Материалы к специальному курсу
для студентов 3, 4, 5 курсов кафедр программистского потока
"Архитектура распределенных систем программного обеспечения "
Лектор: доктор технических наук Карпов Леонид Евгеньевич

Курс посвящен знакомству с основными свойствами распределенных систем программного обеспечения (гетерогенность, прозрачность, открытость, масштабируемость), рассматриваются механизмы реализации этих свойств, используемые при разработке программного обеспечения распределенных систем. Подробно изучаются базовые методы взаимодействия распределенных систем – удаленный вызов процедуры и удаленное обращение к методу объекта, исследуются проблемы, связанные с прозрачностью вызова: преобразование данных при передаче параметров и результатов, согласование протоколов, синхронизация, отработка исключительных ситуаций.

В основе курса – изучение промежуточного (интеграционного) слоя программного обеспечения распределенных систем (middleware) и различных вариантов его реализации: мониторов транзакций, брокеров и мониторов объектов, брокеров сообщений.

Изучаются методы интеграции приложений на базе технологий Интернета. Исследуются концептуальные основы построения сетевых служб, архитектуры сетевых служб, изучаются их базовые компоненты: протокол доступа к объектам, язык описания службы, регистратор сетевых служб. Описываются методы композиции и скоординированной работы сетевых служб.

Содержание курса:

Введение

Понятие распределенных систем программного обеспечения. Виды и свойства распределенных систем программного обеспечения. Виды архитектуры распределенных систем. Управление взаимодействием разнородных приложений (middleware).

Основные механизмы

Понятие удаленной процедуры (модель RPC). Транзакционные мониторы. Алгоритмы подтверждения транзакций. Удаленное обращение к методам объектов (модель RMI). Брокеры объектов (спецификация CORBA). Взаимодействие на основе обмена сообщениями (модель MOM). Очереди сообщений и транзакционные очереди. Модель взаимодействия "точка-точка".

Проблемы интеграции приложений

Комплексная интеграция приложений (EAI). Брокеры сообщений. Модель взаимодействия "публикация/подписка". Системы управления рабочим потоком (WorkflowMS). Серверы приложений.

Технологии Интернета

Понятие сетевой службы (Web Service). Сервисные службы и интеграция приложений. Базовые компоненты сетевых служб. Протоколы и стандартизация. Проблемы публикации данных и поиска сетевых служб. Координация взаимодействия сетевых служб. Композитные сетевые службы.

Основная литература

1. Andrew S. Tanenbaum, Maarten van Steen. "Distributed Systems. Principles and paradigms". Prentice Hall, Inc., 2002 (Э. Таненбаум, М. ван Стеен. "Распределенные системы. Принципы и парадигмы". СПб.: Питер, 2003)
2. Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju. "Web Services. Concepts, Architectures and Applications". Springer-Verlag, 2004.
3. *http://www-128.ibm.com/developerworks/webservices/standards/*

Дополнительная литература

4. John Barkley. "Comparing Remote Procedure Calls", Oct 1993 (*http://hissa.nist.gov/rbac/5277/titlerpc.html*).
5. Philip A. Bernstein. "Middleware - A model for Distributed System Services". Communications of the ACM, v. 39, No 2, February, 1996. (Ф. Бернштейн. "Middleware: модель сервисов распределенной системы". Открытые системы, Системы управления базами данных, № 2, 1997, *http://www.osp.ru/dbms/1997/02/41.htm*).
6. Eric Newcomer. "Understanding Web Services: XML, WSDL, SOAP and UDDI", Addison-Wesley, 2002 (Эрик Ньюкомер. "Веб-сервисы. Для профессионалов", СПб.: Питер, 2003).
7. Robert Orfali, Dan Harkey, Jeri Edwards. "Instant CORBA". Wiley Computer Publishing, John Wiley & Sons, Inc., 1997 (Р. Орфали, Д. Харки, Д. Эдвардс, "Основы CORBA", М., МАЛИП, 1999).
8. Natanya Pitts. "XML In Record Time™", Sybex Inc., 1999 (Натания Питс. "XML за рекордное время", М.: "Мир", 2000).
9. W. Richard Stevens. "UNIX Network Programming. Networking APIs", Prentice Hall PTR, 2nd edition, 1998 (У. Стивенс "Разработка сетевых приложений", СПб.: Питер, 2004).
10. А. А. Цимбал, М. Л. Аншина. "Технологии создания распределенных систем. Для профессионалов". СПб.: Питер, 2003.

Вспомогательная литература

11. *http://www.corba.org/*
12. *http://www-128.ibm.com/developerworks/webservices/library/specification/ws-tx/*
13. *http://www-128.ibm.com/developerworks/library/specification/ws-bpel/*
14. *http://www.sei.cmu.edu/str/descriptions*
15. Oracle Message Broker Administration Guide. Release 2.0.1.0. Part Number A65435-01 (for SPARC Solaris & Windows NT). Доступ в Интернете по адресу *http://cs.ifmo.ru/education/documentation/doc_817/ois.817/a65435/toc.htm*
16. "OSF DCE 1.2.2 Application Development Guide – Core Components", The Open Group, 1997.
17. Jon Siegel. "Quick CORBA™ 3". Wiley Computer Publishing, John Wiley & Sons, Inc., 2001 (Джон Сигел, "CORBA 3", М., МАЛИП, 2002).
18. B. Viveney. "DCE and Object Programming". In W. Rosenberry (ed.) "DCE Today", pp. 251 – 264. Upper Saddle River, NJ, Prentice Hall Inc., 1998.
19. Л. А. Калиниченко, М. Р. Когаловский, "Стандарты OMG: Язык определения интерфейсов IDL в архитектуре CORBA", Системы Управления Базами Данных, № 2, стр. 115-129, 1996 (*http://www.tts.tomsk.su/personal/~sas/DBMS/96_2/source/115.htm*).
20. А. Касаткин. "Средства middleware и их классификация". PCWeek, № 19 (193), 1999.
21. М. Мамаев. "Телекоммуникационные технологии (Сети TCP/IP)". Владивостокский госуниверситет экономики и сервиса. Владивосток, 2001. Доступ в Интернете по адресу *http://athena.vvsu.ru/net/book/index.html*.
22. И. Ш. Хабибуллин. "Создание распределенных приложений на Java 2". СПб.: БХВ-Петербург, 2002.
23. А. А. Цимбал. "Технология CORBA для профессионалов". СПб.: Питер, 2001.

Конспект лекций для самостоятельной подготовки

1. ***Характеристика основных свойств распределенных систем программного обеспечения***. *Распределенная система как набор независимых компьютеров, представляющихся их пользователям единой системой. Сокрытие различий между компьютерами и способов связи между ними, приложения единообразно работают в распределенных системах. Решение задачи облегчения доступа к удаленным ресурсам и контроля их совместного использования. Прозрачность (доступа к данным, сохранности данных, поломки системы или ее части, местоположения, смены местоположения, динамической смены местоположения, репликации, параллельного использования). Открытость (интерфейсы, способность к взаимодействию, гибкость, отделение описаний от определений). Масштабируемость (размер, географическое положение, административное устройство). Централизованные и децентрализованные алгоритмы. Гомогенные и гетерогенные системы.*

Распределенная система – набор независимых компьютеров, представляющихся их пользователям единой системой (определение вольное, неверное, но пригодное). Пользователи и приложения единообразно работают в распределенных системах, независимо от того, где и когда происходит их взаимодействие. Основная задача – облегчение доступа к удаленным ресурсам и контроль совместного использования этих ресурсов, как виртуальных, так и традиционных – компьютеров, файлов. Основные требования:

- Прозрачность.
    - Прозрачность доступа к данным.
    - Прозрачность местоположения.
        - Прозрачность смены местоположения.
        - Прозрачность динамической смены местоположения.
    - Прозрачность репликации ресурсов.
    - Прозрачность параллельного использования.
    - Прозрачность поломки системы.
    - Прозрачность сохранности данных.

  Полностью скрыть распределенность не удается. Прозрачность и производительность.
- Открытость – использование синтаксических и семантических правил, основанных на стандартах. Правильный интерфейс обеспечивает возможность правильной совместной работы одного произвольного процесса, нуждающегося в интерфейсе, с другим произвольным процессом, представляющим интерфейс. Самодостаточность и нейтральность. Переносимость характеризует, насколько приложение, сделанное для одной системы, может работать в составе другой. Способность к взаимодействию характеризует, насколько две разные реализации системы в состоянии работать совместно.
- Гибкость – легкость конфигурирования системы, состоящей из различных компонентов, и легкость подключения новых компонентов.
- Масштабируемость по отношению к размеру (подключение дополнительных пользователей и ресурсов), к географическому положению (пользователи и ресурсы в пространстве), к административному устройству (управление в административно независимых организациях). Проблемы: узкие места по обслуживанию (один сервер для множества клиентов), по данным (один файл с общей информацией), по алгоритмам (централизованный алгоритм и перегрузка коммуникационной сети). Свойства децентрализованных алгоритмов:

    - никто не обладает полной информацией о системе;
    - решения принимаются на основе локальной информации;
    - сбой в одном месте не вызывает нарушения работы алгоритма;

- существования единого времени не требуется.

Требование масштабируемости есть препятствие на пути распространения систем, реализованных для локальных сетей, на уровень сетей глобальных. При синхронной связи клиент, сделавший запрос к серверу, блокируется до получения ответа. В глобальных сетях задержки превышают локальные задержки, поэтому используется асинхронная связь. Разница сетей и в степени надежности связи в локальных и глобальных сетях. В локальных сетях службы фиксированы по компьютерам. В глобальных сетях местоположение службы заранее неизвестно.

Следствие масштабируемости: аппаратные решения могут быть гетерогенными, сами системы могут существовать продолжительное время, но отдельные части могут время от времени выходить из строя. При этом пользователи и приложения не уведомляются о том, что эти части заменены или починены, что добавлены новые части.

2. ***Логические слои программного обеспечения распределенных систем. Виды архитектуры распределенных систем. Одно- и двухярусные системы***. *Презентационный слой, слой прикладной логики и слой управления ресурсами. Монолитность и обмен экранами. Клиент/сервер, "тонкие" и "толстые" клиенты. Концептуальные последствия: удаленный вызов процедуры и прикладной программный интерфейс. Сложности поддержки нескольких клиентов и работы с несколькими серверами.*

Распределенные системы программного обеспечения строятся послойно: (1) презентационный слой, (2) слой прикладной логики и (3) слой управления ресурсами. Слои могут быть абстрактными, но могут быть четко видимы в программном обеспечении в виде отдельных подсистем.

(1) Системы программного обеспечения должны общаться с другими системами. Значительная часть общения связана с преобразованием информации и представлением ее для пользователей. Компоненты системы программного обеспечения, обеспечивающие эту деятельность, формируют презентационный слой. Клиенты могут быть полностью внешними по отношению к системе и независимыми от нее, тогда они не являются ее презентационным слоем (сетевой навигатор – это клиент, он лишь показывает документ, презентационным слоем является сетевой сервер). Клиент и презентационный слой могут быть слиты воедино, что типично для систем типа клиент/сервер, где имеется программа, которая одновременно исполняет роль презентационного слоя и клиента (аплет).
(2) Распределенные системы осуществляют обработку данных программой, реализующей операции, запрошенные клиентом через презентационный слой. Иногда эти программы называются службами, предлагаемыми распределенными системами. В зависимости от сложности выполняемой логики этот слой может называться бизнес процессом, бизнес логикой, бизнес правилами или просто сервером. Все эти имена относятся только к конкретным реализациям.
(3) Для работы распределенная система (как и любая информационная система) нуждается в данных, размещающихся в базах данных и файловых системах. Слой управления ресурсов имеет дело с различными источниками данных, независимо от конкретной природы этих источников. Это дает возможность рекурсивно строить распределенные системы, состоящие из других распределенных систем, как из компонентов.

В практических реализациях концептуальные конструкции слоев могут комбинироваться различными способами. В этих случаях говорят не о слоях, а о ярусах.

Известны 4 основных типа распределенных систем, отличающихся количеством входящих в них ярусов: одно-, двух-, трех- и многоярусные системы.

Одноярусные архитектуры возникли под влиянием систем, включавших в себя большой вычислительный модуль и терминалы, отображавшие результаты работы. Все три слоя располагались на одном ярусе. Клиентом был терминал, имевший только клавиатуру для ввода команд оператора и экран. Если переносимость не является целью, системы создают на языках ассемблеров. В таких системах минимизированы расходы на переключение контекста, а также на преобразования данных. Недостаток одноярусных систем: монолитная структура, затрудняющая их сопровождение. Развитие программирования идет в сторону от одноярусных систем.

Двухъярусные архитектуры появились после возникновения ПЭВМ, на которых можно не только отображать информацию, но и обрабатывать ее, снимая часть нагрузки с других слоев. Презентационные модули стали независимыми, такая архитектура называются архитектурой "клиент/сервер". "Тонкого" клиента проще переносить из одного окружения в другое. "Толстые" клиенты требуют больше ресурсов. Если сервер имеет хорошо определенный, стабильный прикладной программный интерфейс, для него можно разработать все виды клиентов и производить модернизацию сервера без изменения в клиентах. Преимущества перед одноярусными системами: отсутствие переключения контекста, переносимость. Проблемы: ограниченность связи со многими клиентами одновременно, которая требует переключать контекст, трудность работы с разными серверами, когда клиент начинает зависеть от нескольких систем, а изменения в него надо вносить при изменениях любого из серверов. За интеграцию начинает отвечать клиент, который должен комбинировать данные от нескольких серверов, отрабатывать все исключительные ситуации от всех своих серверов, координировать доступ ко всем серверам и т. д.

3. ***Трехъярусные и многоярусные архитектуры распределенных систем***. *Выделение промежуточного слоя системной поддержки. Интеграция разнородных ресурсов, масштабирование и надежность. Стандартизация интерфейсов слоя управления ресурсами.*

Трехъярусные архитектуры решают проблему интеграции всех серверов локальной информационной сети введением слоя программ между клиентом и сервером. Все слои в трехъярусной системе разделены. Презентационный слой размещается в клиенте, как в двухъярусной архитектуре. Прикладная логика размещается в среднем ярусе, который называется промежуточным слоем (middleware) или слоем системной поддержки. Слой управления ресурсами располагается на третьем ярусе и состоит из всех серверов, которые интегрируются в системе. Преимущество: возросшие возможности по масштабированию. Каждый слой может работать на отдельной ЭВМ. Прикладной слой может быть распределен по разным компьютерам или кластерам. Прикладная логика более независима от управления ресурсами, переносимость и переиспользуемость возрастает.

Управление ресурсами трехъярусной системы должно подчиняться четким интерфейсам, которыми должны пользоваться программы прикладной логики, находящиеся в промежуточном слое. Трехъярусные системы привели к стандартизации интерфейсов слоя управления ресурсами.

Преимущества трехъярусной архитектуры проявляются при интеграции разнородных ресурсов. Современные программы промежуточных слоев содержат функциональность, необходимую для введения в эти слои дополнительных свойств: транзакционных гарантий для различных видов ресурсов, балансировки загрузки оборудования, возможностей по регистрации событий, репликации, сохранности данных и многого другого. Стандартизованы глобальные свойства и интерфейсы между разными системными платформами на основе объектно-ориентированного подхода (пример: спецификация

CORBA). Использование системной поддержки позволяет при создании моделей взаимодействия не программировать все самостоятельно.

Основное преимущество трехярусных систем в выделении места для интеграционной логики. Потери в производительности компенсируются гибкостью и поддержкой, которую они оказывают прикладному слою. Потери в производительности при взаимодействии с управлением ресурсами компенсируются возможностями по распространению единой модели на разные сетевые узлы, влияя на масштабируемость и надежность. Ограниченность модели трехярусных систем проявилась при попытках интегрировать несколько трехярусных систем, а также при выходе на уровень Интернета. Проблема - в недостаточной стандартизации.

Многоярусные архитектуры - обобщение трехярусной модели с учетом важности доступа к данным через Интернет. Многоярусные архитектуры разрабатываются для двух основных применений: объединение разнородных систем и подключение к Интернету. Слой управления ресурсами включает в себя не только простые ресурсы, но также двух- и трехярусные системы. Большинство современных систем построено по принципу многоярусности. Основной недостаток многоярусных систем – слишком много промежуточных слоев, часто с избыточной функциональностью, сложных, дорогих в разработке, регулировке и поддержке. Число систем с ростом ярусов растет почти экспоненциально.

Удаленные клиенты связываются с системой посредством Интернета сквозь межсетевые экраны. Их запросы пересылаются на кластер машин, которые в совокупности составляют сетевой сервер (кластеры ЭВМ – типичная конфигурация для трех- и многоярусных систем, они отказоустойчивы и имеют более высокую пропускную способность, чем одиночные компьютеры с той же мощностью процессора).

Внутри системы могут быть дополнительные клиенты, обслуживаемые, как и внешние клиенты, либо через сетевой сервер, либо непосредственным доступом к прикладной логике, реализованной в промежуточном слое. Часто прикладная логика реализуется на кластерах. Внутри системы могут сосуществовать несколько промежуточных платформ, каждая для своего приложения и своей функциональности. За ними могут быть простые файловые серверы или базы данных, работающие на большой ЭВМ, и состоящие из двух-, трех- и многоярусных систем.

С введением в систему новых ярусов достигается повышение ее гибкости, растет функциональность. Одновременно возрастает стоимость взаимодействия между ярусами, возникают проблемы с производительностью системы.

4. ***Способы взаимодействия в распределенных системах****. Синхронное и асинхронное, блокирующее и неблокирующее взаимодействие. Сохранность сообщений. Подтверждение приема, подтверждение доставки, получение ответа.*

Основной характеристикой способа взаимодействия является его синхронность или асинхронность. Блокирующим называется взаимодействие, если вовлеченные стороны прежде, чем переходить к следующим работам, должны дожидаться окончания взаимодействия. Параллельная работа фрагментов систем не имеет никакого отношения к асинхронности и неблокирующему взаимодействию. Сервер, получив запрос от одного из своих клиентов, может, обрабатывая запрос, работать параллельно другим своим клиентам, синхронность же относится к работе запрашивающего обслуживания клиента и того процесса в сервере, который этот запрос обрабатывает.

Синхронное взаимодействие проще асинхронного. Сервер уверен, что состояние клиента не изменится. Программы обращения к серверу и обработки его ответа сильно связаны друг с другом и обычно размещаются рядом. Синхронное взаимодействие применяется в большинстве систем. Синхронное взаимодействие приводит к существенным

потерям времени, а значит и производительности. Поддерживать синхронность в разнородных системах трудно.

Пример асинхронной связи – электронная почта. Письма накапливаются в почтовых ящиках, откуда получатель забирает их, когда хочет. Отправитель не ждет ответа. Это позволяет программе выполнять другие работы и делает ненужной координацию между сторонами взаимодействия. Асинхронная связь наиболее всего эффективна, если взаимодействие не имеет форму запрос/ответ. Взаимодействие может быть построено на регистрации событий и сигналов, либо на модели публикация/подписка, когда одни компоненты извещают систему о постоянно доступной информации (публикуют), а другие о своей потребности в ней (подписываются).

Для асинхронного взаимодействия сообщения должны запоминаться в некотором промежуточном месте, откуда они впоследствии могут извлекаться сервером. Построенные так современные брокеры сообщений способны реализовывать сложные стратегии распространения, обрабатывая форматы или содержание сообщений, пока те находятся в очередях. Это важно для многоярусных систем, которые могут помещать обработчики сообщений в очереди, а не в адаптеры и не в сами компоненты, что дает возможность менять способы преобразования и доставки сообщений без изменения самих компонентов, генерирующих и получающих их.

В системах сохранной связи сообщения не пропадают, если компьютер пользователя выключен, а хранятся в коммуникационной системе до тех пор, пока его не передадут получателю. Кроме сохранной связи существует связь без сохранения сообщений. В таких системах сообщения сохраняются только в период работы приложений, которые их отправляют и получают. На практике применяются различные комбинации взаимодействия. В случае сохранной асинхронной связи сообщение сохраняется в буфере либо локального прикладного комплекса, либо коммуникационного сервера. В случае сохранной синхронной связи сообщения хранятся только на принимающем прикладном комплексе. Отправитель блокируется на все время, пока сообщение не попадет в этот буфер. Усеченный вариант сохранной синхронной связи заключается в том, что блокировка отправителя осуществляется до доставки сообщения коммуникационному серверу получателя.

Асинхронная несохранная связь характерна для транспортного протокола UDP. Если получатель в момент прихода сообщения на принимающий комплекс этого получателя не активен, передача обрывается. Другой пример этого вида связи – асинхронный удаленный вызов процедуры.

Синхронная несохранная связь существует в нескольких вариантах. В самой слабой форме, основанной на подтверждениях приема сообщений, отправитель блокируется до тех пор, пока сообщение не окажется в буфере принимающего комплекса. После получения подтверждения отправитель продолжает свою работу. Другая форма этой связи, ориентированная на доставку, предполагает, что отправитель должен быть заблокирован до момента доставки сообщения самому получателю, который продолжит свою часть работы по его обработке. Наиболее жесткий вариант синхронной несохранной связи, ориентированный на ответ, предполагает блокировку отправителя до получения ответного сообщения с другой стороны, как при работе систем клиент/сервер в режиме запрос/ответ. Эта же схема взаимодействия характерна и для систем, построенных на базе моделей удаленного вызова процедуры и удаленного обращения к методу.

5. ***Основные формы реализации системной поддержки распределенных систем. Краткая характеристика****. Системы на базе RPC. Транзакционные мониторы. Легкие и тяжелые TP-мониторы. Брокеры объектов. Мониторы объектов. Системы на базе обмена сообщениями. Сохранные очереди. Брокеры сообщений.*

Взаимодействие в модели ISO OSI подразделяется на семь уровней (физический, канальный, сетевой, транспортный сеансовый, представлений, прикладной). Каждый уровень

предоставляет интерфейс для работы со следующим уровнем, который состоит из набора операций. На практике все, что выше транспортного уровня, собирается в единый прикладной уровень. Иногда между прикладным и транспортным уровнями выделяется еще один уровень, предназначенный для разных прикладных задач. На этом уровне могут решаться задачи аутентификации (удостоверения личности), авторизации (обеспечение доступа только к тем ресурсам, на которые имеются права), подтверждения транзакций.

Удаленный вызов процедуры призван скрыть четыре первых уровня сетевого взаимодействия, что позволяет полностью отвлечься от необходимости беспокоиться о канале связи, об ошибках, возникающих при передаче, о согласованности действий двух прикладных систем, работающих на разных ЭВМ, о разнице в форматах представления данных на разных ЭВМ. Все, что требуется – сформулировать запрос в виде обращения к процедуре с параметрами.

В инфраструктуру системной поддержки входят: язык описания интерфейсов, транслятор с этого языка и библиотеки, реализующие функциональность, необходимую для удаленного вызова процедуры. Часть этой инфраструктуры используется на стадии разработки программ, использующих удаленный вызов процедур, другая часть необходима в момент выполнения вызова. Все это относится не только к удаленному вызову процедур, но и к другим видам взаимодействия. Системная поддержка может образовывать следующие формы:

- Системы на базе удаленного вызова процедур. Наиболее общая форма, содержащая программы для прозрачного преобразования обычных вызовов процедур в удаленные. Системы удаленного вызова процедур лежат в основе почти всех других форм системного программного обеспечения, включая и сетевые службы.
- Транзакционные мониторы – самая надежная и стабильная технология интеграции прикладных систем. Их можно рассматривать как системы удаленного вызова процедур с транзакционными расширениями. В зависимости от реализации в двух- или трехярусной архитектуре транзакционные мониторы делятся на "легкие" и "тяжелые". Легкие мониторы обеспечивают интерфейс удаленного обращения к базам данных, а тяжелые представляют собой системные платформы, сравнимые с операционными системами.
- После возникновения объектно-ориентированного подхода на базе систем удаленного вызова процедур появились брокеры объектов. Они более развиты, чем системы удаленного вызова, но, по-существу, от них мало отличаются. Наиболее известны брокеры объектов, построенные на основе подхода CORBA, предложенного группой OMG.
- Большая часть функциональности брокеров объектов уже была реализована в транзакционных мониторах. Однако возникла необходимость перевести транзакционные мониторы, основанные на процедурных языках, на объектно-ориентированные языки программирования. Эти факторы привели к сближению транзакционных мониторов и брокеров объектов, новые системы стали называться мониторами объектов.
- Синхронное взаимодействие не всегда обязательно. Сначала появились асинхронные вызовы процедур, в дальнейшем - транзакционные мониторы с сохранными очередями. Такие системы очередей стали называть системами на основе обмена сообщениями. Эти платформы обычно обеспечивают транзакционный доступ к очередям, сохранные очереди и большое количество примитивов для чтения и записи в локальные и удаленные очереди.
- Брокеры сообщений - отдельная форма систем на основе обмена сообщениями, дополнительно обладающая возможностями преобразования и фильтрации сообщений при прохождении через систему очередей. На основе содержания сообщений они могут динамически выбирать их получателей. Единственным

отличием от обычных систем очередей является то, что в брокерах сообщений к очередям можно добавлять дополнительную функциональность, что позволяет асинхронно выполнять сложные виды взаимодействия.

6. ***Принципы реализации удаленного вызова процедур. Основные проблемы реализации***. *Прозрачность. Язык описания интерфейсов (IDL), клиентские и серверные переходники, программные шаблоны и ссылки. Связывание и передача параметров. Маршалинг и сериализация. Синхронизация и отработка ошибочных ситуаций.*

Модель удаленного вызова процедур (RPC): клиент вызывает процедуру, работающую на сервере. Для клиента вызов не отличим от вызова локальной процедуры. При применении RPC синхронизация, установление связи, передача параметров и результата делаются скрытно от клиента (прозрачность). Сложности процесса: разные адресные пространства клиента и сервера, необходимость передачи параметров и результатов, необходимость обработки сбоев и отказов оборудования, передача параметров по значению и по ссылке.

Первый шаг: определение интерфейса процедуры с помощью языка описания интерфейсов (IDL), то есть описание параметров процедуры, передаваемых ей до выполнения и возвращаемых после завершения. Второй шаг: трансляция созданного описания и создание (1) клиентского переходника с программами поиска сервера, форматирования данных (маршалинг – упаковка данных в формат сообщения, сериализация – преобразование сообщения в последовательность байтов), взаимодействия с сервером, передачи параметров и получения ответа, (2) серверного переходника, реализующего серверную часть вызова и состоящего из программ получения запроса от клиента, форматирования данных (десериализация и демаршалинг), вызова реальной процедуры, реализованной на сервере, возврата результатов клиенту, а также (3) программных шаблонов и ссылок (например, файлов-заголовков *.h).

При вызове удаленной процедуры выполняется локальный вызов процедуры, являющейся частью переходника. Вместо вызова реальной процедуры переходник разыскивает сервер, форматирует необходимые данные, устанавливает соединение с сервером. Клиент блокируется и ждет ответа, а на сервере запускается серверный переходник, преобразующий сообщение в параметры локальной процедуры. Сервер видит вызов как прямое обращение к его локальной процедуре с нужными параметрами. Результаты работы процедуры упаковываются серверным переходником в сообщение для клиента. Клиент выводится из состояния ожидания, его переходник распаковывает результаты и передает управление клиенту.

Связывание есть процесс, посредством которого клиент создает соединение с сервером для обращения к удаленной процедуре. При статическом связывании информация о сервере закодирована в программах клиента. Преимущества: простота и эффективность. Недостаток: тесная связь клиента и сервера (если сервер не работает, клиент тоже). Смена адреса сервера приводит к перекомпиляции клиента с новым переходником с новым адресом. Использование дополнительных серверов для повышения производительности невозможно. Балансировку нагрузки на сеть приходится выполнять на стадии разработки клиента. При динамическом связывании создается специализированная служба, ответственная за поиск серверов. Динамическое связывание повышает гибкость системы за счет ее производительности. Клиент не знает, является связывание статическим или динамическим.

Если машины клиента и сервера разные, приходится предпринимать дополнительные действия. Преобразование данных зависит от их типа. Проблемы возникают даже при скалярных значениях параметров, еще более сложна передача параметров по ссылке. Прямое решение – передавать массивы по значению. Вывод: при RPC обе стороны должны следовать

согласованным протоколам, а интерпретация форматов должна быть однозначной. В протоколе надо также согласовывать протокол транспортного уровня (например, TCP/IP).

Стандартный вызов RPC – это полная блокировка клиента до получения ответа от сервера. Если ответ не нужен, клиент замирает навсегда. В таких случаях используют асинхронные вызовы. Сервер сразу отвечает клиенту о своем запуске, а работают потом оба. Первичный ответ содержит только информацию о начале работы сервера, но блокировка клиента снимается.

Основное назначение RPC – скрыть от клиента удаленность, то есть сам факт взаимодействия с сетью и удаленной ЭВМ. Внешне эти вызовы выглядят как локальные. Проблема достижения прозрачности – в возможных ошибках взаимодействия. Существуют разные виды запросов. Некоторые можно повторять многократно (идемпотентность). Правильный выход – строить запросы так, чтобы они все были идемпотентными (нумерация запросов, отметка повторных запросов).

7. ***Основные свойства транзакционного взаимодействия***. *Все или ничего. Атомарность, непротиворечивость, изолированность, долговечность. Плоские, вложенные и распределенные транзакции.*

Транзакции призваны защитить общие ресурсы от одновременного доступа. Транзакции превращают процессы доступа и модификации множества элементов данных в одну операцию. Если в процессе выполнения транзакции будет определено, что дальнейшее ее выполнение невозможно (по любой причине), все данные восстанавливаются с теми значениями и в том состоянии, в котором они были до начала транзакции.

Модель транзакции: процесс объявляет, что намерен начать транзакцию с одним или несколькими другими процессами, происходит согласование условий, выполняются операции, инициатор предлагает всем подтвердить, что работа завершена. Если подтверждение выдают все процессы-партнеры, результаты утверждаются и становятся постоянными. Если хотя бы один процесс-партнер отказывается выдать подтверждение, ситуация возвращается к тому состоянию, которое имело место до начала транзакции. Это свойство называется "все или ничего".

Атомарность гарантирует, что транзакция либо полностью выполняется, либо полностью не выполняется, то есть с точки зрения окружающих систем транзакция выполняется как одна неделимая операция. Пока транзакция находится в процессе выполнения, другие системы не могут наблюдать никаких ее промежуточных состояний. До полного завершения транзакционной вставки информации в файл другие пользователи будут видеть и читать этот файл только в его исходном состоянии. После завершения транзакции размер файла мгновенно изменится, и в то же мгновение в нем появится новая информация.

Непротиворечивость есть соблюдение инвариантов системы. Для каждой системы такие инварианты свои. Изолированность или сериализуемость – это отсутствие влияния на параллельно выполняемые транзакции. Если какие-либо транзакции выполняются параллельно, итог будет таким же, как если бы все транзакции выполнялись последовательно в определенном порядке. Результаты транзакции – долговечны Никакие сбои после завершения операции не могут привести к отмене результатов транзакции.

Плоские транзакции обладают всем набором свойств. Это наиболее простой и наиболее часто используемый тип транзакции. Ограничение: нет частичного результата при прерывании.

Принцип вложенности транзакций позволяет разделять транзакции верхнего уровня на серию иерархически вложенных параллельно работающих транзакций. Параллельность эта может быть вполне реальной: вложенные транзакции могут выполняться на других машинах, но может быть и виртуальной, то есть выполняемой для ускорения или упрощения программирования. Каждая из вложенных транзакций может делиться на транзакции аналогичным образом.

Вложенные транзакции не в полной мере обладают всеми свойствами. Свойство долговечности применимо только к транзакциям самого верхнего уровня (результаты вложенных транзакций могут быть отменены, если не сможет быть выполнена какая-либо из других вложенных транзакций). Вложенные транзакции делят исходную транзакцию логически, а логическое разделение транзакций не обязательно означает распределенности. Распределенные транзакции логически представляют собой плоские неделимые транзакции, работающие с распределенными данными. Основная проблема распределенных транзакций в том, что для блокировки данных и подтверждения транзакции необходимы отдельные распределенные алгоритмы. Подтверждение о возможности проведения транзакции должны получить либо все участники процесса, либо ни один из них.

8. ***Протоколы подтверждения завершения транзакции***. *Однофазное, двухфазное и трехфазное подтверждение. Участники и координаторы. Проблема блокировки координатора.*

Распределенное подтверждение реализуется при помощи координаторов. В простейшей схеме координатор запрашивает все остальные процессы, участвующие в транзакции, в состоянии ли они подтвердить запрашиваемую операцию, а те отвечают на запрос. Эта схема известна под названием однофазного подтверждения. Недостаток: если один из участников на самом деле не может осуществить операцию, он не в состоянии сообщить об этом координатору.

Двухфазное подтверждение (Two-Phase Commit, 2PC) строится из двух фаз, каждая из которых включает в себя два шага. Первая фаза называется фазой голосования, вторая фаза есть фаза решения. Сначала координатор рассылает всем участникам запрос голосования, а участники возвращают координатору свои голоса "за" или "против" подтверждения своей части транзакции. Собрав ответы всех участников, координатор посылает всем участникам глобальное подтверждение. Если хотя бы один участник отказался подтвердить свою часть транзакции, координатор вместо подтверждения рассылает всем указание отмены транзакции. Если участник, голосовавший за подтверждение, получает подтверждение от координатора, он подтверждает транзакцию. В случае же получения отмены выполнение транзакции прекращается.

Проблемы возникают, если при использовании протокола двухфазного подтверждения в системе возникают ошибки. Координатор и другие участники имеют такие состояния, в которых они блокируются и ждут поступления сообщений извне. Если в работе координатора возникнет ошибка, работа протокола нарушается, поскольку другие процессы будут бесконечно ожидать сообщений. Для исправления ситуации вводится механизм тайм-аута.

Отказоустойчивость алгоритма 2PC достигается ведением журналов состояний. Чтобы гарантировать, что процесс действительно восстановился, нужно чтобы он сохранял свое состояние при помощи средств длительного хранения данных. Выбор содержания и момента времени записи в журнал являются очень важными с точки зрения производительности, поскольку выполняются для каждой транзакции и поскольку ведение журнала предполагает запись на диск (а это длительная операция).

Для протокола 2PC характерно наличие ситуации, когда участнику приходится блокироваться до восстановления работоспособности координатора. Эта ситуация возникает, когда все участники успевают получить запрос голосования, но после этого координатор выходит из строя. В этом случае участники даже совместно не состоянии решить, каким образом продолжить работу далее. По этой причине протокол 2PC называется протоколом блокирующего подтверждения. Часто в таких случаях решение о разблокировке передается системному администратору, который подтверждает или отвергает транзакцию и согласовывает состояние системы после восстановления координатора. На практике вероятность отказа координатора именно в данной точке очень мала.

Существуют методы, позволяющие избежать блокировки участников в случае поломки координатора. Один из методов состоит в использовании примитива групповой рассылки, когда получатель немедленно рассылает ответное сообщение всем остальным процессам. Существует вариант протокола 2PC, названный протоколом трехфазного подтверждения (3PC), который предотвращает блокировку процессов при появлении ошибок аварийной остановки. Этот протокол после фазы голосования вводит еще одну фазу подготовки к подтверждению. Он используется редко, поскольку условия, в которых блокируется протокол 2PC, возникают достаточно редко. Большинство систем используют алгоритм 2PC: протокол 3PC более дорог и в практических ситуациях может оказаться блокирующим. Трехфазное подтверждение необходимо сопровождать системами сборки мусора, иначе информация о многочисленных состояниях протокола слишком быстро разрастается. Многие системы соглашаются с риском блокировки подтверждения при возникновении ошибок, другие соглашаются со снижением непротиворечивости, которое возникает при сбоях в системе.

9. ***Функциональность и архитектура транзакционных мониторов. Мониторы объектов***. *Поддержка распределенных транзакций. Объектно-ориентированный подход мониторов объектов. Транзакционный удаленный вызов процедуры. Транзакционные скобки. Ведение журнальных записей, восстановление, блокировка. Управление процессами, приписка приоритетов, балансировка нагрузки, репликация. Менеджер транзакций, интерфейс, программный поток, маршрутизатор, менеджер взаимодействия, оболочки, службы.*

Для реализации транзакций применяются транзакционные мониторы, разработанные для обеспечения надежного мультиплексного доступа к большому количеству ресурсов для большого количества параллельных пользователей. Транзакционный монитор есть альтернативная операционная система. Облегченный вариант процесса, используемый транзакционными мониторами, работает по одной (разделяемой) программе и видит все данные, которые были видны в точке создания. При создании облегченный процесс получает собственный идентификатор, собственный набор регистров, стек для локальных переменных, что позволяет ему работать независимо от других процессов. Рост сложности и функциональности привел к появлению облегченных транзакционных мониторов, предоставлявших функциональность монитора (транзакционный удаленный вызов процедур) в составе систем управления базами данных. Эти системы были расширением по отношению к системам баз данных и имели существенно ограниченную функциональность по отношению к традиционным мониторам.

Цель транзакционного монитора – поддержка выполнения распределенных транзакций на основе транзакционного RPC. Традиционные вызовы удаленных процедур независимы, что для транзакций не подходит. Если вызывается сервер, который, выполняя удаленную процедуру, вызывает другой сервер, нет способа отличить, где произошла ошибка – в первом или втором сервере. Семантика транзакционного вызова: если группа вызовов процедур внутри транзакции подтверждается (успешно завершается), имеются гарантии, что каждый из вызовов завершился успешно. Если возникло прерывание выполнения группы вызовов, общий эффект будет таким, как если бы ни один из вызовов не выполнялся. Процедурные вызовы, заключенный в транзакционные скобки, рассматриваются как единое целое, а инфраструктура RPC гарантирует их атомарность.

Функциональность транзакционных мониторов достаточна для разработки, выполнения, управления и сопровождения транзакционных распределенных систем. Эта функциональность включает в себя язык IDL, именование, безопасность и аутентификацию, компиляторы переходников, поддержку работы с транзакционными вызовами (транзакционные скобки, обратные вызовы), ведение журнальных записей, восстановление,

блокировку, управление процессами и приоритетами, балансировку нагрузки, репликацию, управление ресурсами.

Архитектура транзакционных мониторов включает клиентский интерфейсный компонент и поддержку прямого доступа через терминал. Программный поток исполняет процедуры, написанные на языке данного монитора, которые содержат операции над именованными логическими ресурсами. Маршрутизатор сопоставляет операции и вызовы, которые могут относиться к ресурсам (например, БД) или локальным службам самого монитора. В состав маршрутизатора входит специализированная БД, содержащая определения соответствий между именами логических ресурсов и физическими устройствами. При изменении системы администратор исправляет это соответствие: клиентское приложение модифицировать не требуется, клиент знает только логические имена. Взаимодействие с ресурсами осуществляется через менеджер взаимодействия (например, систему сообщений, обеспечивающую откат и гарантию доставки). Разнородность ресурсов, связанных с монитором, скрывают оболочки. Выполнение транзакции проводит менеджер транзакций, исполняющий протокол 2PC и гарантирующий транзакционные свойства процедур, исполняемых транзакционным монитором.

Примитивы транзакционных скобок есть обращения к клиентскому или серверному переходнику. При работе открывающей скобки клиентский переходник от менеджера транзакций получает имя транзакции и создает контекст последовательности вызовов. При обращении клиента к одному из серверов, серверный переходник извлечет контекст, уведомит менеджера транзакций, что стал участником транзакции и преобразует удаленный вызов в локальный. При выполнении закрывающей скобки клиент обращается к менеджеру транзакций, тот инициирует протокол 2PC со всеми серверами и принимает решение о завершении транзакции или об отказе. После завершения 2PC менеджер транзакций сообщает об этом клиентскому переходнику, который возвращает управление программе клиента, показывая, как завершилась транзакция.

Полезность транзакционных мониторов и последующее появление брокеров объектов привело к построению объектно-ориентированных транзакционных мониторов, то есть мониторов объектов (иногда совместимыми со спецификацией CORBA).

10. ***Объектно-ориентированный подход к распределенной обработке информации***. *Объекты. Создание объектов при компиляции и выполнении. Сохранные объекты. Привязка к клиенту. Статическое и динамическое обращение к методам. Передача параметров по ссылке и по значению. Инкапсуляция.*

Объектно-ориентированный подход в программировании привел к использованию этого подхода в распределенных системах. Объекты инкапсулируют данные, называемые состоянием, и операции над этими данными, называемые методами. Для доступа к состоянию объекта нужно использовать методы, обращение к которым осуществляется через интерфейсы. Для распределенных систем разделение на интерфейсы и объекты позволяет помещать интерфейсы на одну машину, а сами объекты на другую. Когда клиент выполняет привязку к распределенному объекту, в его адресное пространство загружается реализация интерфейса объекта. Заместитель клиента аналогичен переходнику при RPC (маршалинг параметров с упаковкой в сообщения при обращении к методам и демаршалинг данных из ответных сообщений с результатами). Сами объекты находятся на сервере. Входящий запрос на обращение к методу сначала попадает в серверный переходник (скелетон), преобразующий его в обращение к методу. Скелетон отвечает за маршалинг параметров в ответных сообщениях и их пересылку заместителю клиента. Если объект физически распределен по нескольким машинам, это скрывается от клиентов за интерфейсами.

Для работы с объектами, описанными явно, интерфейсы компилируются в клиентские и серверные переходники, позволяющие обращаться к удаленным объектам. Такие объекты зависят от языка программы. Динамически создаваемые объекты не зависят от языка

программирования, способ их реализации открыт, но методы должны быть доступны удаленно. Часто используются адаптеры объектов. Сохранность - одно из важнейших свойств объекта. Такие объекты не зависят от сервера. Сервер может прекратить работу, а затем снова прочитать состояние сохранного объекта и приступить к обработке обращений к нему. Несохранные объекты существуют, пока ими управляет сервер. В системах распределенных объектов для повышения прозрачности могут создаваться уникальные ссылки на объекты, которые свободно передаются между процессами. Перед обращением к методу объекта по его ссылке сначала выполняется привязка (явная или неявная), в результате создается заместитель объекта, реализующий интерфейс с методами, к которым обращается процесс. Для выполнения привязки система находит сервер, управляющий объектом, и помещает заместитель объекта в адресное пространство клиента.

Удаленное обращение к методам (RMI) в части маршалинга и передачи параметров напоминает RPC, описание интерфейсов объектов проводится на языке определения интерфейсов. Обращение к методу может быть статическим (интерфейсы известны при разработке) или динамическим.(параметры собираются перед обращением к методу). При обращении к методам в качестве параметров используются ссылки на объекты, которые передаются по значению. Привязка к объекту выполняется, как только это понадобится. Для простых объектов это приводит к потере эффективности. Копирование ссылки происходит только для удаленных объектов, а локальные объекты копируются целиком, что повышает эффективность, но снижает прозрачность и затрудняет программирование.

Инкапсуляция позволяет поставщику услуг менять детали реализации услуги, не требуя изменений со стороны клиента. Клиентские и серверные объекты не требуется реализовывать на одном языке программирования, в рамках одинаковых ОС. Все, что нужно знать клиенту о сервере, есть в спецификации сервера. Параметры обращений преобразуются в общее представление, не зависящее от языка программирования и ОС, а затем преобразуются назад в скелетоне еще до обращения к методам, зависящим от языка. Это позволяет менять логику реализации и язык клиентских и серверных программ. Использование стандартизованных спецификаций и компиляторов дает гарантию, что реализация объектов переносима между разными системами, независимо от языка программирования.

Другое проявление инкапсуляции – независимость от размещения объектов. Когда нужно обратиться к службе, клиент обращается к ядру системы, которое выдает ссылку на объект (непрозрачный идентификатор), то есть логическое имя объекта, приписываемое ему в момент создания. Ссылка действительна до момента, когда объект уничтожается, даже если в течение жизни объект меняет свое местоположение. Объекты могут не только менять свои физические адреса, они могут оказываться доступными посредством разных брокеров объектов.

11. ***Брокеры объектов. Архитектура брокеров объектов на известном примере****. Брокер объектов. Службы и средства. Язык описания интерфейсов (IDL), скелетоны и переходники. Интерфейс динамического обращения и репозиторий интерфейсов. Динамическое обращение к службе.*

На основе модели RMI создано множество реализаций брокеров объектов, облегчивших создание объектно-ориентированных распределенных приложений. Стандарт CORBA (Common Object Request Broker Architecture) – это архитектура и спецификация для создания и управления объектно-ориентированными приложениями, распределенными в вычислительной сети. В настоящее время выработано несколько версий стандарта CORBA. Спецификацией не определяются ни языки программирования разрабатываемых объектно-ориентированных приложений, ни операционные системы, в которых они должны работать. Другими известными примерами являются брокер объектов Distributed Component Object Model (DCOM) и появившийся позднее COM+, инкорпорированный в операционные

системы фирмы Microsoft. Известны платформы .NET (Microsoft) и Java 2 Enterprise Edition (J2EE).

Система, подчиняющаяся спецификации CORBA, состоит из трех основных частей: брокера запросов объектов, набора служб, доступных с помощью стандартизованного прикладного программного интерфейса, и набора средств и инструментов. Брокер объектов содержит базовые функции взаимодействия объектов. Службы предоставляют функции сохранности, управления жизненным циклом, безопасности и многие другие. Средства CORBA предоставляют службы верхнего уровня, необходимые приложениям (управление документами, поддержка мобильных агентов), а также службы вертикальных рынков (образования, здравоохранения).

Чтобы обратиться к объекту, этот объект должен сначала опубликовать свой интерфейс. Интерфейсы описываются на языке описания интерфейсов (IDL). В дополнение к описанию методов, в отличие от систем на базе RPC, язык IDL CORBA поддерживает множество объектно-ориентированных концепций, например, наследование и полиморфизм. Запись на IDL передается компилятору, который формирует переходник (заместитель объекта, скрывающий распределенность) и скелетон. Для клиента вызовы выглядят не удаленными, а локальными. Программа переходника содержит в себе описание методов, предоставляемых реализацией объекта, она загружается вместе с программой клиента. Скелетон защищает от проблем распределенности сервер, поэтому сервер может разрабатываться так, как если бы вызовы к нему поступали из локального окружения. Все, что требуется знать программисту, это IDL-интерфейс сервера. Семантика интерфейсов методов не формализуется, она описывается другими средствами, например, в виде комментариев или в составе сопроводительной документации. Стандарт CORBA 3 поддерживает IDL для Си, Си++, Java, Smalltalk, Ады, Кобола, Лиспа, PL/1 и других языков.

Клиент может быть статически привязан к интерфейсу, компилятор IDL может строить переходник для конкретного сервисного интерфейса. Модель RMI допускает динамическое обнаружение новых объектов и построение обращений к ним в процессе работы без переходника. Это базируется на двух компонентах: репозитории интерфейсов и интерфейсе динамического обращения. Репозиторий интерфейсов хранит определения всех объектов, известных брокеру. Приложения используют репозиторий для поиска, редактирования или удаления интерфейсов. Один брокер может иметь несколько репозиториев, и несколько брокеров могут иметь доступ к одному репозиторию, но каждый брокер должен иметь хотя бы один репозиторий.

Чтобы клиенты могли идентифицировать нужную им службу, в спецификации CORBA ссылки на сервисные объекты выдаются только службой именования и справочной службой. Служба именования извлекает ссылки на объекты, отталкиваясь от их имени, а справочная служба ищет службы, основываясь на их свойствах. Все службы вносят сведения о своих свойствах в справочник. Используя его, клиенты могут искать объекты, реализующие тот или иной интерфейс, а также объекты, свойства которых имеют заданные значения (например, книги по археологии).

На практике построение динамических обращений весьма сложно, не столько с технической точки зрения, сколько семантически. Одна проблема заключается в том, что для поиска службы клиентский объект должен понимать смысл свойств службы, что в свою очередь требует распространения знаний среди клиентов и поставщиков служб. Если клиент не был заранее реализован с возможностью взаимодействовать с конкретной службой, трудно придать ему возможность выяснять, какие операции может выполнять вновь обнаруженная служба, каков подлинный смысл ее параметров, в каком порядке надо обращаться к ним, чтобы добиться нужной функциональности.

12. ***Основные службы спецификации CORBA***. *Служба именования и справочник. Службы транзакций и сохранности. Служба параллельного доступа. Служба событий. Службы свойств, отношений, контейнеров и запросов. Служба*

жизненного цикла. Служба времени. Служба экстернализации Служба безопасности. Служба лицензирования.

Спецификация CORBA позволяет пользователям систем, построенных на ее основе, организовывать свои программы в виде служб, предоставляющих услуги другим программам, то есть таким же службам или более традиционно построенным программам пользователей. Однако обычно (в некоторой аналогии с библиотеками стандартных программ) вместе с базовой системой (самим брокером CORBA) могут распространяться программы служб, спецификации которых также стандартизованы. Некоторые из этих служб являются обязательными и распространяются всегда, другие же службы, несмотря на стандартность интерфейса, имеют более ограниченное применение (они могут применяться в редко встречающихся ситуациях) и распространяются по отдельным соглашениям с пользователями.

Служба именования используется для сопоставления имен со ссылками на объекты, группирования и поиска имен для получения доступа к ссылкам на объекты. Имена объектов могут быть составными, причем в составных именах все имена, кроме последнего являются именами контекстов имен, а именем объекта является самый последний компонент. Именам приписываются атрибуты, которые никак не интерпретируются, но могут использоваться в программах. Справочная служба ищет объекты не по имени, а по совокупности свойств. Службы предварительно регистрируются в справочнике, сообщая о себе классификационную информацию. Обычно в справочник записывается ссылка на интерфейс, предоставляющий услугу, наименование типа службы и ее свойства. Тип службы содержит информацию об именах операций, на которые служба может реагировать, типы параметров и возвращаемых значений. Свойства представляют собой пары имя-значение, которые описывают возможности службы. Справочник ведет репозиторий типов служб, который позволяет расширять одни типы наследованием свойств других типов. Можно организовывать динамические свойства, которые справочник получает в ответ на запросы от зарегистрированных служб. Справочные службы связанных между собой брокеров могут самостоятельно взаимодействовать друг с другом. Оптимизация поиска может проводиться при помощи стратегий, ограничений и предпочтений. Стратегии задают диапазон поиска. Можно ограничить количество справочных связей, ограничить поиск одним справочником, указать, с какого справочника надо начать поиск. Ограничения указывают критерии поиска с помощью языка ограничений (например, SQL). Предпочтения позволяют задать порядок, в котором выдаются ответы на запросы о поиске. Можно вводить предпочтения выбора максимума, минимума или выбора с ограничениями на основе SQL-операторов.

Служба свойств сопоставляет с объектами их свойства в виде пар "имя объекта – значение свойства". Свойства объектов не зависят от их IDL-описаний и не являются частью типов объектов. Свойства могут создаваться, изменяться и уничтожаться динамически. Служба отношений динамически устанавливает связи между объектами. Отношения могут иметь произвольную сложность особенно при выполнении групповых операций (перемещении, копировании, удалении). Все службы строятся на основе IDL-описаний и имеют свои интерфейсы, но только для службы жизненного цикла реализации методов создаются клиентами, которые знают семантику создаваемых ими объектов и операций над ними (создание, копирование, удаление).

Служба событий (в новых версиях – служба уведомлений) рассылает уведомления о событиях в системе объектам системы. Уведомлением называется сообщение, которое объект посылает объектам для информирования о наступлении события. Поставщики поставляют события, а получатели обрабатывают их с помощью обработчиков. В push-модели активной стороной является поставщик событий, а получатели заранее регистрируются в канале событий для указания интереса к событиям данного типа. Получатель может отсоединиться от канала и прекратить прием событий. В pull-модели получатель сам запрашивает у поставщика данные о событии через обращение к методам

канала, а регистрацию проходят поставщики. Получатель может организовать опрос объектов о наличии событий, а поставщики могут отсоединяться от канала, прекращая поступление к себе мешающих им запросов. Канал событий поддерживает обе модели, что позволяет множеству поставщиков взаимодействовать с множеством получателей асинхронно и без каких-либо дополнительных сведений друг о друге.

Служба сохранности объектов обеспечивает механизм сохранения состояний объектов, как в реляционных, так и в объектных базах данных. Для одноуровневых хранилищ (объектных СУБД) клиент не должен знать, где находится объект – в памяти или на диске, объекты в двухуровневых хранилищах (реляционных СУБД) различаются по месту их размещения. Клиент может пользоваться возможностями автоматического управления сохранностью данных или управлять сохранностью самостоятельно. Служба не нарушает принцип инкапсуляции, но позволяет увидеть некоторые детали, то есть определить, когда объект сохранен, а когда – восстановлен.

Служба экстернализации строит образы объектов, взаимодействующих со стандартными потоками ввода-вывода, что позволяет обрабатывать объекты другими системами, не связанными с брокерами объектов.

Служба транзакций (OTS) взаимодействует непосредственно с самим брокером. Совместная задача брокера и OTS – автоматически передавать транзакционный контекст всем участвующим в транзакции объектам. Эти две службы обеспечивают единую среду для работы всех существенных компонентов системы. Поддерживается протокол двухфазного подтверждения, вложенные и межброкерные транзакции. Транзакционный клиент обращается к методам серверных объектов, заключая свои вызовы в транзакционные скобки. Брокер, получив указание о начале транзакции, создает новый контекст, который размножает всем зарегистрированным участникам транзакции. Транзакционный сервер – это один или несколько объектов, чье поведение определяется транзакцией, но чьи состояния и ресурсы не могут быть восстановлены самостоятельно. Транзакционный сервер не принимает участия в подтверждении транзакции, но может быть причиной отката. Серверы восстановления – объекты, чьи состояния зависят от подтверждения или отката транзакции. Восстановимые объекты – это транзакционные объекты с защищенными ресурсами (транзакционные файлы, базы данных), они регистрируются в OTS, сообщая о связи с транзакцией, контекст которой размножался брокером.

Служба совместного доступа управляет общими ресурсами, координируя параллельные транзакции, в том числе вложенные. При блокировании ресурсов служба может работать двумя способами: от имени транзакции или от имени нетранзакционного клиента. В первом случае снятием блокировок при завершении или прерывании транзакции управляет служба транзакций, во втором – сам клиент. Существует понятие блокировочного набора, объединяющее блокировки, относящиеся к одному ресурсу. Блокировки создаются только на блокировочных наборах, которыми можно управлять как группами.

Служба безопасности работает с полным соблюдением правил прозрачности: все действия выполняются автоматически. Реально со службой взаимодействует только администратор. Служба безопасности решает проблемы идентификации пользователей, определения прав доступа к объектам, управления режимами делегирования полномочий в цепочках последовательных вызовов объектами друг друга, проведения аудита, защиты информации при передачах, ведения достоверной истории взаимодействия объектов.

При обеспечении безопасности в системах CORBA приходится учитывать дополнительные сложности, связанные с распределенной природой защищаемых объектов. Объекты могут быть одновременно и клиентами и серверами. Распределенные объекты меняются со временем, реализации объектов, а также связи между объектами строятся в процессе работы программы. Многие аспекты взаимодействия распределенных объектов скрыты из-за инкапсуляции. Гибкость объектно-ориентированной модели создает для безопасности особые трудности. Объекты полиморфны, легко заменить один объект другим,

имеющим такой же интерфейс. Распределенные объекты способны к беспредельному росту. Сами распределенные системы постоянно изменяются.

Служба времени предназначена для синхронизации часов компьютеров, на которых функционируют части системы. Основной частью службы является использование универсального времени, которое на практике проводится очень часто. С помощью службы времени можно получить текущее время с оценкой ошибки измерения, определить порядок, в котором происходят события, возбуждать события, привязанные к определенным моментам времени, вычислять интервалы времени между двумя событиями. Служба лицензирования используется для удобства разработчиков, которые могут отслеживать степень использования объекта или ограничивать его применение, как по субъектам, так и по времени использования.

Служба коллекций нужна для создания групп объектов и управления этими группами. Служба вводит несколько видов стандартных коллекций – множества, наборы, последовательности и другие. На их базе определены более сложные интерфейсы – очередь, двусторонняя очередь, очередь с приоритетом, стек, список, массив, дерево. Для каждого вида коллекции объектов определен свой интерфейс создания коллекции и свои средства организации перебора элементов коллекции. В коллекции можно добавлять элементы, менять их на другие элементы, извлекать элементы, удалять элементы из коллекций. Коллекции могут иметь свойства упорядоченности элементов, доступа по ключу, сравнимости и уникальности элементов.

Служба запросов предназначена для поиска объектов, соответствующих заданным критериям, записываемым на расширенном языке SQL, либо на объектном языке запросов OQL.

13. ***Распределенная обработка информации на основе обмена сообщениями***. *Асинхронная сохранная связь. Модель очередей сообщений. Основные составляющие сообщений. Базовый интерфейс. Транзакционные очереди.*

Сообщения, которыми обмениваются клиенты и поставщики служб, характеризуются типом и параметрами (набором пар <имя, значение>). Тип сообщения определяется конкретной системой (чаще используются типы языка XML). Асинхронное взаимодействие хорошо подходит для решения интеграционных задач и сетевых служб. Системы очередей сообщений создают расширенную поддержку асинхронной сохранной связи. Они предоставляют возможность промежуточного хранения сообщений, не требуя активности во время передачи от отправителя и получателя. Разница между "клиентом" и "сервером" исчезает, что отличается от других форм взаимодействия, где клиенты обращаются к методам, предоставляемым серверами.

Модель обмена сообщениями формирует основу, на которой разрабатываются концепции и свойства, упрощая разработку взаимодействующих приложений и управляя сбоями. Приложения общаются друг с другом путем помещения сообщений в специальные очереди. Эти сообщения передаются по цепочке серверов к месту назначения, даже в том случае, если получатель в момент отправки неактивен. Каждое приложение имеет свою очередь, куда другие приложения могут посылать сообщения. Несколько приложений могут совместно использовать одну очередь. Отправитель гарантирует попадание сообщения в очередь получателя. Когда сообщение будет прочитано и обработано, зависит от получателя. Это - слабосвязанное взаимодействие.

Сообщения должны быть правильно адресованы и могут содержать любые данные. Адресация осуществляется путем указания имени очереди, в которую направляется сообщение. Базовый интерфейс, предоставляемый приложениям, состоит из неблокирующего вызова Put (передает сообщение базовой системе, где оно помещается в очередь), блокирующего примитива Get (извлекает из очереди самое старое сообщение, включая блокировку в случае пустой очереди, может использовать приоритеты или образцы

сообщений) и неблокирующего варианта этого же действия – примитива Poll. Большинство систем очередей сообщений, поддерживают функции обратного вызова, которые вызываются при попадании сообщения в очередь. Обратные вызовы используются для автоматического запуска процесса, который будет забирать сообщения из очереди. При слабой связанности с отправителем получатели сами выбирают момент обработки сообщений. Извлечение сообщений из очереди производится только тогда, когда получатель может или должен их обработать. Системы очередей устойчивее к системным сбоям, чем системы RPC или RMI, поскольку от них не требуется поддерживать работоспособность в момент отправки сообщения. Сообщения могут иметь связанные с ними временные интервалы действия. Если сообщение не извлекается до истечения указанного интервала, оно уничтожается. Очереди могут быть разделяемыми среди нескольких приложений, что позволяет распределить нагрузку и повысить производительность.

Системы очередей предоставляют пользователям прикладной программный интерфейс, который базируется на ранее описанных примитивах, но более удобен для использования в привычном программном окружении. Есть возможность построения транзакционных очередей, в которых даются гарантии, что, если сообщение было отправлено, оно будет доставлено один и только один раз, даже если сама система обмена сообщениями выйдет из строя в период между объявлением о сообщении его доставкой. Транзакционные очереди обеспечивают защиту от сбоев. Получатель может объединять набор чтений и уведомлений в одном атомарном действии. В случае сбоя в момент, когда не все операции из атомарного набора выполнились, происходит откат всех завершенных действий. Откат операции извлечения сообщения соответствует возврату сообщения в очередь, чтобы оно снова могло быть прочитано. Откат уведомления о сообщении приводит к удалению сообщения из сохранной памяти. Однако возврат сообщения в очередь гарантирует лишь, что оно не будет уничтожено и что оно обязательно будет обработано снова, давая возможность поставщикам услуги преодолеть кратковременные проблемы. Если невозможность правильной обработки сообщения связана не с временным сбоем в системе, возврат сообщения в очередь не поможет исправить проблему и получить обратную связь, которая бы могла помочь отправителю узнать о невозможности обработать запрос.

14. ***Интеграция приложений в рамках предприятия***. *Интеграция серверов. Распространение понятия "службы". Основные проблемы интеграции приложений (разные ОС, разные интерфейсы, разная функциональность, разные форматы данных, разные требования по безопасности, разные модели взаимодействия и протоколы). Адаптеры приложений.*

Когда интегрируемые системы совместимы друг с другом, сравнимы функционально и не включают в себя чрезмерно много различных платформ, традиционные системы могут успешно использоваться для интеграции серверов. Когда эти системы отличаются друг друга по устройству и функциональности, использование традиционных платформ нерационально. Системы комплексной интеграции предприятий – эволюционный шаг в развитии системных платформ, расширивший их возможности по интеграции приложений. Расширение возможностей связано с переходом к асинхронному взаимодействию.

Трехярусные системы с централизованной платформой позволили отделить прикладную логику от управления ресурсами и строить системы не из групп серверов, а из кластеров ЭВМ. Эти системы объединяли в себе различные виды ресурсов. Интеграция сдерживалась разнородностью ресурсов и недостатком стандартов взаимодействия. При комплексной интеграции предприятий важно автоматизировать передачу информации от одного подразделения другому. Практическая жизнь сложилась так, что системы автоматизации, базы данных, форматы данных разных прикладных систем разные. При комплексной интеграции предприятий приходится сталкиваться с тем, что разные прикладные системы работают в рамках разных операционных систем, разные системы

поддерживают разные интерфейсы и функциональность, разные системы имеют разные форматы данных, разные требования по безопасности, используют разные модели взаимодействия и протоколы (RPC, TRPC, система на базе спецификации CORBA). При комплексной интеграции приходится сталкиваться со многими нетехническими проблемами: каждая система обычно принадлежит и эксплуатируется разными подразделениями компании. Каждое подразделение управляется автономно и выполняет много специфических функций, которые не согласуются с тем, что получится при интеграции.

Модель интегрированного предприятия базируется на двух фундаментальных компонентах: адаптерах и системе обмена сообщениями. Адаптеры призваны согласовывать гетерогенные форматы данных, интерфейсы и протоколы с выбранной моделью и форматами. Для каждого приложения, которое включается в интеграционный процесс, должен быть создан свой адаптер. Система обмена сообщениями представляет собой инструмент для взаимодействия с адаптерами и с интегрируемыми системами.

Если выбрана интегрированная платформа автоматизации и реализованы адаптеры для всех интегрируемых компонентов, интеграция приложений сводится к разработке приложения и конфигурированию адаптеров. Приложение взаимодействует с системой обмена сообщениями путем публикации и подписки на доставку сообщений. Конфигурирование адаптеров нужно, чтобы дать им возможность подписаться на соответствующие сообщения и выполнять соответствующие действия в системах, интегрированных с помощью описанного процесса.

Наиболее пригодной системной платформой для проведения комплексной интеграции предприятий признаны брокеры сообщений, но транзакционные мониторы также предлагают некоторый ограниченный набор средств интеграции функциональности приложений. Однако брокеры сообщений разрабатывались специально для целей интеграции и предлагают более полные наборы функций, включая многочисленные и разнообразные адаптеры, средства их настройки и приспособления, а также инструментарий для разработки адаптеров. В результате интеграция с помощью брокеров сообщений дает много преимуществ, которые зависят от конкретных сценариев и реализуемой функциональности, но часто приводят к снижению затрат на разработку, снижению затрат на нововведения, уменьшению усилий на поддержку. Интеграция упрощается, поскольку интегрируемые системы слабо связаны, и основные интеграционные усилия вкладываются в адаптеры. Сокращение сроков интеграции и ее удешевление влияет на возможность подключения новых услуг и расширение возможностей предприятия. Возникает возможность быстрого реагирования на требования введения новых услуг, а значит, этих новых услуг может предлагаться больше, чем при использовании альтернативных решений (например, систем на базе RPC). Использование адаптера приводит к локализации взаимодействия с внешней системой в отдельном модуле – адаптере. Если появляются новые версии систем для них надо создавать новый адаптер, но приложение, выполняющее интеграционную логику, при этом будет затронуто весьма незначительно.

15. ***Модели взаимодействия "точка/точка" и "публикация/подписка". Брокеры сообщений***. *Особенности взаимодействия с помощью брокеров сообщений. Подписчики и издатели. Типы и параметры сообщений. Фильтрация сообщений. Комбинирование брокеров сообщений.*

Традиционные системы на базе RPC или обмена сообщениями создают между приложениями соединения типа "точка-точка", поэтому слишком статичны. В таких системах ответственность за определение получателя сообщения ложится на отправителя, поэтому. если отправителю потребуется начать взаимодействовать с новой системой, его приложение должно быть изменено. Эта схема адресации может стать трудно управляемой, поскольку число отправителей и получателей постоянно растет, а окружение, в котором работает система, становится более динамичным. Снять ограничения удается с помощью

брокеров сообщений, которые действуют как посредники между системными сущностями. Брокеры сообщений обеспечивают гибкую маршрутизацию и обходят ограничение, передавая задачу определения пути пересылки сообщения от отправителя в ядро системы.

Отправители могут не указывать получателя сообщений. Брокер сообщений, выполняя правила, заданные пользователем, сам определит получателя. Каждое правило содержит логическое условие, зависящее от значений данных, находящихся в сообщении. Правила могут определяться в брокере сообщений, либо на уровне очереди. Если они определяются в брокере, то относятся ко всем сообщениям. Если логика связана с очередью, она определяет, в получении каких сообщений эта очередь заинтересована. Брокеры сообщений полностью развязывают отправителей и получателей сообщений. Отправители не указывают и не беспокоятся о том, кто получит отправленное ими сообщение, получатели могут не беспокоиться о том, какое приложение присылает им сообщения. Брокеры сообщений можно комбинировать с разделяемыми очередями: сообщения будут доставляться в несколько очередей, в зависимости от логики маршрутизации (развязка получателей и отправителей), разные приложения могут совместно извлекать сообщения из очередей и обрабатывать их (балансировка нагрузки).

Поскольку в системах обмена сообщениями взаимодействие приложений проходит через центральное звено системы, в этом слое можно концентрировать больше функциональности, чем просто правила маршрутизации. Приложения единой системы, имеющие разные форматы данных могут определять правила преобразования данных и связывать их с очередью. Приложения делаются более устойчивыми, но очереди становится очень трудно сопровождать. Если правил будет слишком много, может стать проблемой производительность брокеров.

Брокеры сообщений могут поддерживать разные модели взаимодействия, основанные на обмене сообщениями: "точка/точка" или "публикация/подписка". В модели "публикация/подписка" отправители не указывают получателей: они публикуют сообщение в ядре системы. Если приложение заинтересовано в получении сообщений данного типа, оно должно подписаться. Как только издатель посылает сообщение данного типа, система доставляет копию каждому из подписавшихся на него. Подписчики могут определять нужные их сообщения с помощью их типа или с помощью логических условий, вычисляемых над параметрами сообщений. Системы могут вводить структурные имена типов на основе иерархии типов/подтипов произвольной глубины.

В состав брокеров сообщений входит поддержка администратора, имеющего право определять типы сообщений, которые можно отправлять и получать, и пользователей, которым разрешено получать и/или получать сообщения и настраивать для себя логику маршрутизации. Администраторы брокеров более важны из-за слабой связи между отправителями и получателями. Системы "публикация/подписка" разрешают издателям фиксировать ограничения на набор пользователей, которые могут получать определенные сообщения.

Архитектуры брокеров сообщений могут приспосабливаться к работе в разных административных зонах, ведущих интенсивный обмен сообщениями. Если клиент одного из брокеров хочет получить сообщение, посланное клиентом другого брокера, он должен подписаться у своего брокера на такую услугу. В свою очередь, сам брокер подписывается на то же самое сообщение у своего соседа. Когда один из клиентов публикует сообщение нужного типа, оно будет переслано подписавшемуся брокеру, а тот, получив сообщение, доставит его всем своим клиентам, подписавшимся у него на это сообщение. С точки зрения одного брокера другой выглядит точно так же, как и любой другой клиент. Единственным отличием является то, что они относятся к разным административным зонам, и, следовательно, администратор одного брокера должен устанавливать разрешения на подписку и публикацию для этого клиента.

16. ***Системы управления рабочим потоком***. *Административные и производственные рабочие потоки. Графы потоков. Виды узлов (рабочие, маршрутизирующие, начала и завершения). Мотор рабочего потока. Модельные абстракции и системная поддержка (опережающее и обратное восстановление, языки управления исключительными ситуациями, истечение предельного срока). Архитектура современного интегрированного предприятия (комплексная интеграция приложений на основе брокера сообщений, управление рабочим потоком).*

Системы управления рабочим потоком (СУРП) могут выполнять диспетчерскую роль, передавая документы от одного участника процесса другому, и определять бизнес логику, необходимую для интеграции разнородных распределенных систем. Бизнес процессом называется совокупность действий, осуществляемых пользователем, которые вместе приводят к достижению конкретных целей производственного процесса. СУРП поддерживает проектирование, разработку, выполнение и анализ рабочих процессов. Обычно рабочий поток (РП) описывается направленным графом (называемым графом потока), определяющим порядок выполнения стартовых, рабочих, маршрутизирующих и конечных узлов процесса. Комбинируя и конфигурируя различные узлы, разработчики могут описывать работы, порядок, в котором они должны выполняться, кому (или чему) работа должна быть приписана. РП может быть запущен несколько раз. Несколько запусков одного или разных РП могут исполняться параллельно.

С РП могут быть связаны переменные, значения которых локальны для каждого его запуска. Переменные используются для обмена данными между рабочими узлами и для определения значений условий маршрутизации. Предлагается много моделей: на диаграммах активности унифицированного языка моделирования UML, на сетях Петри, на графах состояний и активности или с помощью иерархической декомпозиции активности. Внутри систем рабочие потоки часто описываются на языке XML.

РП выполняется мотором, который составляет расписание работ и приписывает работы исполнителям. При запуске РП, мотор извлекает его определение и выясняет узлы, которые надо включить в работу, то есть те, с которыми соединен выход стартового узла. Если запускаемый узел – маршрутизирующий, мотор просто вычисляет условие и выясняет, какой выход (какой следующий узел) должен быть активирован. Если запускается рабочий узел, мотор определяет ресурс, к которому узел должен быть приписан, а затем помещает работу в очередь работ выбранного ресурса. Как только ресурс будет готов приступить к очередной работе, он извлечет из своей рабочей очереди указание о работе, выполнит его и возвратит результат мотору. Ресурсы не обязаны извлекать работы из очереди работ в порядке их попадания туда. Мотор непрерывно просматривает входную очередь для обработки сообщений рабочих узлов о завершении работы.

РП характеризуется вызовом нескольких функций в порядке, определяемом некоторой потоковой логикой. Эффект циклов и условных операторов моделируется вставкой маршрутизирующих узлов. РП может иметь переменные, которые могут передаваться как входные или получаться как выходные данные некоторых вызовов рабочих узлов. Эти переменные используются для оценки условий маршрутизации и для передачи информации между узлами. Однако РП обычно состоят из крупных структурных блоков активностей и приложений, которые могут работать часами или сутками, например, из больших программных модулей или целых приложений, а часто из сложных многоярусных систем.

Обеспечение транзакционности РП более сложно, чем это же действие в приложениях баз данных. Откат для некоторых операций может приводить к потере части работ и является очень сложным. Из-за длительности операций рабочего потока невозможно блокировать необходимые ресурсы базы данных: это может привести к задержкам в работе других приложений, пользующихся тем же ресурсом. СУРП хранит состояние запущенного потока в сохранной памяти для опережающего восстановления. При таком подходе теряются только

работы, выполнявшиеся узлами, активными в момент сбоя и не успевшими сообщить о своих результатах. Обратное восстановление СУРП выполняют в тех случаях, когда нельзя завершить выполнение потока и требуется повторно выполнить часть работ. С каждым рабочим узлом ассоциируется компенсационная активность, выполнение которой семантически аннулирует эффект рабочего узла. Некоторые СУРП предлагают дополнительные возможности по управлению исключениями, то есть механизмы регистрации событий, которые могут возникать асинхронно по отношению управляющему потоку. СУРП могут приписывать каждому шагу процесса допустимое время исполнения. Если указанное время истекает, система предпринимает корректирующее действие. Многие СУРП имеют встроенные брокеры ресурсов, которые исполняют ресурсные правила. Функциональность СУРП достаточна для кодирования логики процессов при интеграции приложений, включая стыковки крупноблочных приложений и работ, выполняемых вручную.

17. ***Особенности обмена информацией в Интернете. Поддержка удаленных клиентов в Интернете***. *Универсальный клиент и протокол HTTP(S). Аплеты. Общий шлюзовой интерфейс (CGI). Сервлеты.*

Первыми стандартами для обмена информацией по Интернету были протокол Telnet и простой протокол передачи писем SMTP. Эти протоколы определяют различные способы соединения различных систем, не зависящие от работающих операционных систем и компьютеров. Протокол передачи файлов FTP предназначен для пересылок файлов между сайтами Интернета и обеспечивает публикацию файлов, размещая их на FTP-сервере. В основе сетевых технологий в настоящее время находятся протокол HTTP, язык HTML, сетевые серверы и сетевые навигаторы. Передачей файлов по сети управляет протокол передачи гипертекстов. Это обобщенный протокол, его можно использовать для доступа к другим протоколам – FTP и SMTP. Протокол HTTP поддерживает язык разметки гипертекстов HTML, который определяет стандартный набор текстовых индикаторов, показывающих, как отображаются страницы.

Информационный обмен протоколом HTTP осуществляется в форме документов, которые определяются с помощью унифицированного идентификатора (URI). Документы могут быть статическими или динамическими, содержимое которых генерируется при обращении к ним. Каждый ресурс доступен с помощью адреса (URL), который идентифицирует местоположение ресурса. Адрес ресурса полностью описывает способ доступа к ресурсу, то есть с его помощью можно узнать протокол, который надо использовать для доступа к ресурсу, адрес машины, где размещен ресурс, и иерархическое строение местоположения ресурса в машине.

Механизм протокола HTTP основан на модели клиент/сервер. Клиент HTTP открывает соединение с HTTP сервером и посылает сообщение, состоящее из метода запроса, URL и версии протокола, за которой следует само сообщение. Сервер возвращает ответ, состоящий из строки состояния и сообщения с запрошенным документом, и закрывает соединение. Сервер должен устанавливать сохранное соединение, что снижает расходы на открытие и закрытие соединений.

Между клиентом и сервером могут находиться один или более промежуточных объектов – отправляющих агентов, принимающих агентов (шлюзов) и тоннелей. Отправляющий агент действует и как сервер и как клиент, его цель выполнять запросы по поручению других клиентов. Шлюз действует как промежуточный сервер для некоторого другого сервера. Тоннель выполняет действия по передаче между двумя соединениями. Агент и шлюз в состоянии обрабатывать URL и содержимое сообщения, а тоннельная программа этого делать не может. Расширенный протокол HTTPS позволяет серверам и клиентам аутентифицировать друг друга и устанавливать между собой шифрованные

соединения. Протокол HTTP не имеет состояний, он не поддерживает выполнение транзакций, историю вызовов должны хранить прикладные программы.

Чтобы обеспечить доступ внешним клиентам надо иметь специализированные клиентские программы для каждой системы, с которой надо взаимодействовать. Но имеется универсальный клиент для всех систем: стандартизованный сетевой навигатор. Чтобы сетевые навигаторы могли работать с динамическими документами, используются аплеты – программы, встроенные внутрь HTML документов. При загрузке документа виртуальная машина Java исполняет программу, что превращает просмотровую программу в клиента. Однако аплет существует только в течение текущего вызова просмотровой программы и не обладает свойством сохранности.

Для доступа к динамическим документам разработан общий шлюзовой интерфейс (CGI). Этот интерфейс связывает программы с URL, и при обращении к URL автоматически вызывается программа. Параметры и вся другая необходимая для вызова информация пересылаются как часть вызываемого URL. Программы шлюзового интерфейса обычно помещают в выделенные каталоги, чтобы сетевой сервер мог распознать их в качестве программ и отличить от статических документов. Шлюзовой интерфейс не обладает достаточной производительностью. Альтернативой ему служат сервлеты. Выполнение сервлета управляется тоже с помощью URL. Однако сервлеты вызываются непосредственно с помощью информации, содержащейся в HTTP-запросе и зависящей от конкретного сервлета. Сервлеты выполняются как параллельные ветви одного процесса виртуальной машины Java. Накладные расходы при этом сильно снижаются. Процесс Java сервера играет роль сохранной памяти для сервлетов. С помощью сервлетов легче реализовать и проще масштабировать отслеживание сеансов, соединения с базой данных и прочие аналогичные действия.

18. ***Архитектура серверов приложений на известном примере***. *Общая схема построения. Поддержка прикладного слоя. Основные компоненты и варианты их реализации (поддержка состояний объектов и работа с несколькими клиентами, управление сообщениями). Поддержка слоя управления ресурсами. Адаптеры ресурсов. Поддержка презентационного слоя. Виды поддерживаемых клиентов (сетевые навигаторы, традиционные приложения, интеллектуальные внешние устройства, системы электронной почты, клиенты сетевых служб).*

Серверы приложений эквивалентны традиционным системам поддержки распределенной обработки информации, отличаясь использованием глобальной сети, как основного канала доступа к системным службам. Презентационный слой этих систем играет более важную роль, чем в традиционных системах. Основная функциональность серверов приложений связана с поддержкой взаимодействия и презентации, поддержкой интеграции приложений, поддержкой доступа к ресурсам.

В сервере приложений J2EE для поддержки взаимодействия и презентации предназначены сервлеты, а также язык тегов JSP и его интерпретатор, прикладной интерфейс для работы с XML (JAXP), система электронной почты (Java Mail), служба аутентификации и авторизации (JAAS). Поддержка интеграции приложений обеспечивается компонентами EJB, интерфейсом именования и каталогов (JNDI), службой сообщений (JMS) и транзакционным интерфейсом (JTA). Поддержка доступа к ресурсам осуществляется компонентами обеспечения связи с базами данных (JDBC) и подключения архитектур (J2CA). Система включает и другие прикладные интерфейсы, нужные для интеграции приложений.

Целью поддержки прикладного слоя является создание единого окружения для всех видов прикладной логики, работающей в глобальной сети и без нее. В комплексе J2EE поддержка прикладной логики концентрируется в трех основных компонентах: EJB, JNDI и JMS. Компонент EJB размещается на серверной стороне и обеспечивает функциональность,

специфическую для данного приложения, например, подготовку прайс-листов в ответ на запрос покупателя о покупке. В состав некоторого приложения могут входить несколько компонентов EJB одного из трех типов, в зависимости от метода, выбранного для управления состоянием и сохранностью. Сессионный вариант управляет сессией с клиентом. Имеются модификации, отслеживающие состояния и не делающие этого. Если состояния не отслеживаются, один и тот же компонент может использоваться для работы с разными клиентами. Объектовый вариант продолжает существование за пределами одной сессии с клиентом. Он имеет состояния, хранящиеся в базе данных или в другой сохранной памяти. Разработчик может сам писать SQL-запросы или другие команды, чтобы запомнить состояние в базе данных, а может управлять сохранностью автоматически. Вариант с управлением сообщениями обеспечивает возможность асинхронного взаимодействия с клиентом. Асинхронность взаимодействия достигается обменом сообщениями по интерфейсам JMS.

Компоненты EJB могут помещаться в контейнер, предоставляющий функциональность, общую для разных типов компонентов по поддержке транзакционности, сохранности и безопасности. Контейнер управляет транзакциями, руководствуясь свойствами, приписанными компонентам EJB во время конфигурирования. Привязка к компоненту EJB производится с помощью службы именования и каталогов (JNDI). Используя службу JNDI, клиенты могут привязываться к серверу, зная только имя объекта. В состав сервера J2EE входит транзакционный прикладной программный интерфейс JTA. Для совместимости с системами, построенными на основе спецификации CORBA, в сервер J2EE добавлен набор транзакционных интерфейсов JTS.

Для решения проблем работы со слоем управления ресурсами сервер приложений J2EE использует два стандарта: JDBC и J2CA. Стандарт JDBC определяет прикладной интерфейс, дающий разработчику возможность доступа к источнику табличных данных, а стандарт J2CA определяет, как надо строить адаптеры ресурсов.

Сервер J2EE имеет в своем составе язык JSP, позволяющий вставлять в обычные документы HTML дополнительные теги, расширяющие возможности стандартных страниц. С помощью тегов JSP можно задавать параметры страниц HTML, включать в страницы файлы, адреса которых указываются в тегах, выполнять определения объектов Java, вычислять значения выражений Java и выполнять встроенные фрагменты программ Java.

Серверы приложений развивают возможности шлюзов и поддерживают сетевые навигаторы, работающие с простыми страницами HTML и с аплетами, приложения, такие же, как и в традиционных системах, сложные устройства, например, мобильные телефоны, программы электронной почты, а также клиентов сетевых служб, то есть приложения, взаимодействующие с сервером через стандартные протоколы сетевых служб.

19. ***Сетевые технологии для интеграции приложений***. *Обмен B2B. Междоменные соединения (протоколы GIOP и IIOP). Прямая интеграция систем. Межсетевые экраны и проблемы традиционных видов взаимодействия (удаленный вызов процедуры, удаленное обращение к методу, междоменные протоколы). Туннелирование.*

Допустимые стратегии взаимодействия трехярусных систем определяются всеми возможными комбинациями связей между этими слоями: связь может вестись на уровне клиента, промежуточного слоя или ресурсов. До появления Интернета реальными были только два варианта: использование обмена сообщениями на уровне специализированных клиентов и прямое соединение системных слоев с прямыми обращениями к удаленным системам. Возникновение Интернета добавило вариант универсального клиента всех систем (сетевой навигатор).

Большинство системных платформ спроектированы для работы в одной отдельно взятой локальной вычислительной сети (ЛВС). Однако сейчас необходима автоматизация

взаимодействия между предприятиями. Технически электронное взаимодействие означает вызов службы, размещенной в другой компании. Расширение на Интернет достигается соединением нескольких систем друг с другом. В системах с брокерами объектов это делается с помощью обобщенного межброкерного протокола GIOP, который определяет, каким образом вызовы от одного брокера передаются другому и как назад отправляется ответ на вызов. Этот протокол был расширен и превращен в межброкерный протокол Интернета IIOP, в котором определено, как транслировать вызовы протокола GIOP в вызовы протокола TCP/IP, которые уже можно посылать в Интернет.

На практике брокеры соединены с Интернетом через межсетевые экраны, которые сильно ограничивают взаимодействие. Межсетевой экран – барьер на пути нежелательного сетевого трафика, который блокирует многие коммуникационные каналы, в том числе почти все виды взаимодействия, предлагаемые традиционными продуктами интеграции предприятий. В Интернете нельзя рассчитывать на использование удаленных вызовов процедур, обращений к методам и протоколов GIOP/IIOP. Другим препятствием является различие определений интерфейсов и форматов данных, применяемых в их приложениях. Для помощи в поисках вызываемых служб необходим единый справочный сервер, для которого трудно найти подходящее место. При наличии межсетевых экранов невозможно осуществлять прямое взаимодействие интегрируемых систем. Для проникновения сквозь экраны применяются специальные трюки, известные как туннелирование. Протоколы, которые могли бы быть заблокированными межсетевыми экранами, прячутся под протоколами (чаще всего под HTTP), которые этими экранами допускаются. В этом случае посредническая программа упаковывает исходное сообщение в документ на языке HTML, посылает его по протоколу HTTP, а после прихода документа к получателю извлекает сообщение.

В традиционных системах представление данных скрыто в языке IDL, который выполняет две задачи: определение интерфейса и введение промежуточного машинно-независимого представления данных. В настоящее время в разнородной сети Интернет используется язык разметки XML. Этот язык ориентирован на описании синтаксиса представления данных, а не семантики. Язык XML предоставляет стандартные правила определения структуры документов, пригодной для автоматического разбора. Четко определенная структура помогает разобраться с семантикой отдельных частей документа, а стандартизация способа кодирования структуры позволяет разрабатывать инструментарий для просмотра, разбора документов, а также для извлечения информации из них. Программным способом можно идентифицировать, что документ содержит элементы заказчик, идентификатор продукта, дата поставки, и извлечь содержимое, связанное с этими элементами, а затем создать объект.

Другой формой автоматической поддержки XML-документов является валидация. Стандарт языка XML определяет правила, которым должен удовлетворять документ, чтобы считаться правильным. Можно определить тип документа и потребовать, чтобы документ соответствовал некоторому типу. Ограничения на структуру документов могут специфицироваться в разделе определения типа документа DTD или в XML-схеме. После определения типов документов возникает возможность описывать документы данного типа. Проверять структуру и содержимое документа на соответствие предписаниям типа или схемы документа можно автоматически. При этом наличие определений типов или схемы не дает никакой семантической информации: то, что в документе должен присутствовать тот или иной элемент, никак не разъясняет семантику документа и то, как информация будет восприниматься получателем.

20. ***Определение и общая характеристика сетевых служб. Сетевые службы и системы интеграции предприятий***. *Способы и проблемы интеграции систем в открытой глобальной сети (доверительные отношения, соединения "точка-точка", гетерогенность окружения, проблемы подтверждения транзакций).*

Консорциум UDDI определяет сетевую службу как самодостаточное модульное бизнес приложение, имеющее открытый стандартизованный интерфейс, ориентированный на Интернет. Определение консорциума W3C: сетевая служба есть приложение, идентифицируемое с помощью URI, интерфейс и способ связывания которого могут быть определены, описаны и выявлены как артефакты XML. Этим определением поясняется, что значит доступность службы (определение, описание, выявление), а что значит ее ориентированность на Интернет. Здесь же указывается, что сетевая служба должна быть "службой" в смысле, похожем на трактовку этого термина традиционными системами. Сетевые службы – это компоненты, которые могут интегрироваться в более сложные распределенные приложения. Консорциум W3C настаивает на том, чтобы язык XML был включен в определение сетевых служб. Этот язык может рассматриваться в качестве части сетевой технологии, он определяет форматы данных для взаимодействия через Интернет.

Традиционные подходы к построению распределенных систем и традиционные методы интеграции приложений направлены на интеграцию автономных систем и автоматизацию бизнес процессов, распространяющихся на несколько таких систем. Применение традиционных подходов требует от участников взаимодействия достижения соглашения по использованию и совместному управлению конкретной системной платформой, а также по реализации "глобального рабочего потока". Этот подход не всегда пригоден, поскольку трудно достичь необходимого уровня доверия между участниками. Другой причиной непригодности традиционного подхода является неверность предположений, делавшихся при интеграции предприятий (длительность взаимодействия в сети Интернет препятствует применению традиционных протоколов типа 2PC, они блокируют ресурсы на слишком большой срок, делая невозможным параллельное выполнение других операций).

Развитие глобальной сети привело к появлению новых стандартов, протоколов (HTTP), форматов (XML). В основе работы сетевой службы лежит предположение, что функциональность, открываемая некоторым предприятием для взаимодействия с его партнерами, будет проявляться как "услуга", "служба". С точки зрения использования сетевые службы не отличаются от служб обычных программных систем, но к ним можно обращаться через Интернет. Службы оказываются слабо связанными системами, поскольку они определяются, разрабатываются и управляются разными компаниями. Не все, что доступно через Интернет, представляет собой сетевую службу. Сетевая служба это не набор страниц в Интернете, а приложение с общеизвестным и стабильным программным интерфейсом.

Протоколы, с которыми работают сетевые службы, должны быть пригодны для работы без выделенных серверов. Традиционные протоколы (2PC) работают с центральным транзакционным координатором, который обладает возможностями блокировать ресурсы. Протокол 2PC и другие протоколы взаимодействия и координации должны быть модифицированы, чтобы работать в децентрализованном режиме в отсутствии доверительных зон и гибко проводить блокировки.

Общий подход к интеграции приложений (B2B) в Интернете с помощью сетевых служб таков: каждая сторона представляет свои внутренние операции как некоторую (сетевую) службу, являющуюся точкой входа в локальную информационную систему. Работа независимых приложений осуществляется в режиме равноправного взаимодействия, но некоторые компоненты могут централизовываться. Обмены информацией проводятся на основе единых протоколов, разработанных так, чтобы децентрализованно обеспечивать свойства традиционных протоколов. Сетевые службы сами исполняют эти протоколы и скрывают от программистов все сложные проблемы интеграции приложений.

Сетевые службы это аналог сложных оболочек, которые инкапсулируют одно или несколько приложений, создавая для них единый интерфейс и обеспечивая доступ через Интернет. С точки зрения клиента интеграции подлежат именно оболочки, только они видны

интегрируемому приложению. Гомогенность компонентов существенно снижает трудность интеграции. Сетевые службы создают фундамент, на котором строится программное обеспечение, поддерживающее интеграцию приложений в Интернете. К сетевым службам не обязательно обращаться только через Интернет. Они также могут быть доступными клиентам локальных сетей.

21. ***Основные технологии сетевых служб. Языки. Интерфейсы. Бизнес протоколы***. *Основные направления развития средств интеграции систем автоматизации в глобальной сети Интернет (развитие архитектуры программных систем, развитие протоколов взаимодействия, разработка дополнительных стандартов). Режим равноправного взаимодействия (без выделенных серверов). Бизнес протоколы (поддержка "разговоров"), горизонтальные протоколы (системная поддержка) и вертикальные (отраслевые) стандарты.*

Описание традиционной программной службы основывается на интерфейсе и языке описания интерфейса. Семантика различных операций и порядок, в котором к ним надо обращаться, предполагаются известными разработчикам клиентских программ заранее. В сетевых службах неявный контекст отсутствует, поэтому описания должны быть точнее.

При описании сетевых служб применяется стек языков описания, в котором каждый элемент более высокого уровня, использует и уточняет описание нижнего уровня. В качестве общего метаязыка используется язык XML, который широко распространен, имеет гибкий синтаксис и позволяет определять языки описания и протоколы служб. При описании интерфейсов сетевой службы дополнительно указывается ее адрес и транспортный протокол. Наиболее используемым является язык описания WSDL. Обмены между службами, проходящие в определенном порядке, называются разговорами. Набор правил, регулирующих разговоры, называется бизнес протоколом. Для описания разговоров используются языки WSCL и BPEL.

Решения об использовании службы принимаются на основании их описаний, доступных благодаря спецификации универсальных средств описания, обнаружения и интеграции (UDDI). В этой спецификации показано, как организуется информация о сетевой службе и как строить репозитории, где эта информация может регистрироваться. Языки описания интерфейсов и свойства служб не стандартизуют содержание службы и ее семантику. Для того, чтобы определить конкретные интерфейсы, протоколы, свойства и семантики, предлагаемые сетевыми службами в конкретных приложениях, необходимы вертикальные стандарты. Например, стандарты RosettaNet описывают автоматизированный обмен коммерческой информацией. Эти стандарты связаны с определенными приложениями, ими руководствуются при написании клиентских приложений.

Для обеспечения доступности сетевых служб их описания заносятся в справочники, которые позволяют разработчикам регистрировать новые службы, а пользователям отыскивать их. Поиск службы может осуществляться во время разработки или динамически. Для взаимодействия с единой справочной службой или для взаимодействия между локальными справочниками определяются протоколы и программные интерфейсы опубликования и поиска информации.

Каждый уровень стандартов взаимодействия сетевых служб характеризуется одним или несколькими протоколами, которые применимы ко всем сетевым службам. Протоколы взаимодействия невидимы для разработчиков, что позволяет сосредотачиваться на бизнес логике. Транспортные протоколы скрывают от сетевых служб коммуникационные сети. Наиболее часто используется протокол HTTP. На базе транспортных протоколов можно определять форматы и методы упаковки информации. Для сетевых служб используется простой протокол доступа к объектам SOAP. Он не детализирует свойства конкретного обмена информацией, а задает шаблон обобщенного сообщения. Для указания конкретных

свойств над протоколом SOAP делаются дополнительные надстройки. Например, протокол WS-Security описывает, как с помощью SOAP осуществлять безопасные обмены.

Бизнес протоколы связаны с конкретными приложениями, но многое из нужной им поддержки может быть реализовано обобщенными компонентами. Эти компоненты могут хранить сведения о состоянии разговора между клиентом и службой, ассоциировать сообщения с теми или иными службами, верифицировать сообщения на соответствие правилам, определенным в протоколах. Например, перед началом взаимодействия клиенты и службы должны выбрать протокол и назначить ответственного за его выполнение. Метапротоколы, способы использования языка WSDL и протокола SOAP определяются в спецификации WS-Coordination.

Сетевые службы должны обладать всеми свойствами традиционных служб (надежность, транзакционность), и их свойства реализуются с помощью децентрализованных (горизонтальных) протоколов. Эти протоколы называются горизонтальными, поскольку они применимы ко многим сетевым службам. Первым протоколом такого рода был протокол транзакционного взаимодействия сетевых служб WS-Transaction.

Сетевая служба (базовая служба) может получать запросы от других сетевых служб (композитных служб). Базовые и композитные службы неразличимы: это просто сетевые службы, описываемые и реализуемые одинаковыми способами. Наиболее продвинутым стандартом композитных служб является BPEL.

22. ***Внутренняя и внешняя архитектура сетевых служб***. *Дополнительный ярус программного обеспечения. Централизованная и децентрализованная поддержка. Поставщики служб и публикация служб. Децентрализованные протоколы и композиция служб.*

Сетевые службы получают запросы и передают их другим системным программам (внутренняя инфраструктура). Сетевые службы обеспечивают интеграцию различных служб между собой (внешняя инфраструктура). Внешняя архитектура включает централизованные брокеры (обеспечивают важнейшие свойства взаимодействий - журнализацию, транзакционные гарантии, надежность, службы именования и справочников), набор компонентов, координирующих взаимодействие сетевых служб друг с другом, в частности, реализующие децентрализованные протоколы, инфраструктуру композиции служб. Разделение на внутреннюю и внешнюю архитектуры может быть проведено даже, если сетевые службы используются внутри предприятия.

Сетевые службы играют ту же роль, что и традиционные промежуточные слои программного обеспечения, но имеют другой масштаб. Сетевые службы это оболочки, которые обращаются к внутренним службам, реализующим нужную прикладную логику. Системная поддержка сетевых служб связана с упаковкой и распаковкой сообщений, пересылаемых между сетевыми службами, а также с преобразованием их в формат внутреннего программного обеспечения. Эти преобразования приводят к росту накладных расходов на выполнение операций, поэтому сетевые службы чаще используются в крупноблочных приложениях.

К внешней архитектуре сетевых служб относится та сторона их работы, которая напоминает работу брокеров сообщений и систем управления рабочим потоком. Самой большой проблемой построения такой архитектуры является проблема поиска наиболее удобного места для размещения системной поддержки. Первое решение – реализация децентрализованной поддержки, когда все участники кооперируются и вместе выполняют функции службы именования. Это очень правильный подход, но с ним трудно добиться нужного уровня надежности и доверия. Другое решение – введение посредников (брокеров), выполняющих нужные функции. В настоящее время существует только один тип брокера сетевых служб, стандартизованного и используемого на практике: сервер именования.

Внешняя архитектура сетевых служб в ее централизованном варианте пригодна для всех видов систем, включая системы RPC. Поставщики служб создают сетевые службы и определяют интерфейсы, а также генерируют описания этих служб и делают их известными путем публикации в реестре служб. Информация из описания служб используется реестром для создания каталогов служб и поиска в этих каталогах по запросам от пользователей. Реестр должен восприниматься всеми, как сетевая служба, адрес и интерфейс которой известен всем заранее.

При централизованном подходе управление транзакциями, имеющееся в транзакционных мониторах, и службы брокеров объектов размещаются в общеизвестных местах. Размещение транзакционного брокера в общеизвестном месте требует внедрения стандартного способа выполнения транзакций, принятого всеми и не нарушающего транзакционной семантики. Транзакционная семантика в каждой конечной точке транзакции должна полностью определяться базовой системной платформой, что требует абсолютной стандартизации транзакционного взаимодействия со всеми базовыми платформами. Серьезным ограничением является необходимость полного доверия брокеру со стороны всех участников.

Альтернатива есть только в реализации децентрализованного транзакционного брокера. В этом случае все, кто обращается к сетевым службам, имеют собственное управление транзакциями, которое несет полную ответственность за соблюдение транзакционной семантики при обращении к сетевым службам. Аналогичные решения предлагаются и для других компонентов, которые обычно реализуются в централизованной манере, поэтому децентрализованные протоколы и инфраструктуры, поддерживающие выполнение этих протоколов, используются чаще. Эта инфраструктура является частью внешней архитектуры, но обычно принадлежит и контролируется поставщиками служб и теми, кто к этим службам обращается, а не третьими сторонами.

Еще одной составляющей внешней архитектуры сетевых служб является инструментарий для композиции служб. Композиция относится к внешней архитектуре, поскольку она связана с интеграцией других служб. Технически композиция может быть централизованной, но поскольку реализации являются частными и конфиденциальными, эта инфраструктура должна находиться в распоряжении поставщиков служб, а не у посторонних участников.

23. ***Механизм взаимодействия сетевых служб по протоколу SOAP***. *Метасинтаксис и структура сообщения. Промежуточные узлы доставки сообщений (передача на конечный пункт, пропуск без обработки, обработка на каждом узле).*

Для всех спецификаций необходимо иметь общий синтаксис их описания, для сетевых служб – XML. Все стандарты основаны на XML, а структуры данных и форматы описываются как XML-документы. Механизм, позволяющий удаленным объектам взаимодействовать друг с другом, имеет общий формат данных для сообщений, которые будут использоваться при обменах, соглашение по поддержке разных форм взаимодействия (посылка сообщений или удаленный вызов процедуры) и правила работы с сообщениями в терминах транспортного протокола.

Сетевые службы могут пользоваться разными транспортными протоколами. Часто подходит TCP/IP, для туннельного проникновения через межсетевые экраны необходим протокол HTTP, а для асинхронной отправки сообщений применяется протокол SMTP. Механизм взаимодействия должен уметь работать с разными транспортными протоколами, а сообщения надо уметь преобразовывать под правила любого из них. Механизм взаимодействия должен оставлять все участвующие приложения слабо связанными, поэтому он базируется на обмене сообщениями. Сетевые службы основывают взаимодействия на простом протоколе доступа к объектам – SOAP.

Сообщения протокола SOAP состоят из заголовка и тела сообщения. Заголовок может отсутствовать, но тело сообщения – обязательно. Заголовок и тело состоят из блоков. У каждого сообщения есть отправитель, конечный получатель и произвольное число промежуточных узлов, которые обрабатывают сообщение и переправляют его получателю. Основная информация размещается в теле сообщения, заголовок служит для обработки на промежуточных узлах или для дополнительных служб (транзакционное взаимодействие, безопасность). Протокол SOAP может использоваться двумя разными способами: в стиле документов и в стиле RPC. В первом случае происходит асинхронный обмен документами, второй случай отличается тем, как построено тело сообщения. В нем непосредственно содержатся имя вызываемого метода и его параметры. Дополнительные свойства удаленной процедуры (указание транзакционного контекста) содержатся в заголовке. Сообщения, пересылаемые по протоколу SOAP, подчиняются правилам кодирования, определяющим, каким образом конкретная структура будет представлена в XML. Клиент и сервер должны договариваться о выбранном способе представления данных. Разные методы кодирования приводят к разному представлению сообщений в XML. Различия возникают на стадии принятия решения о том, как передаются параметры сообщений (удаленных процедур) – в виде вложенных элементов или значениями атрибутов исходного элемента.

Промежуточные узлы, которые поочередно обрабатывают SOAP-сообщения по мере их доставки получателю, представляют собой разные ярусы системной поддержки сетевой службы. В блоках заголовка сообщения можно указывать, для выполнения какой роли этот блок предназначен. Если блоку приписана роль "никто", это означает, что блок не обрабатывается ни на одном узле на пути сообщения (если это нужно для обработки других блоков, блок может читаться). Если блоку приписана роль "конечный получатель", ни один промежуточный узел обработкой блока заниматься не будет. Если блоку приписана роль "следующий", блок может (не обязательно) обрабатываться на каждом узле, куда попадает сообщение (в том числе и на узле конечного получателя). Тело сообщения не имеет приписанной роли, оно всегда обрабатывается конечным получателем. Обработка блоков может быть разной (удаление из заголовка, запись в журнал узла, внесение добавочной информации). Блок может содержать признак, который требует обязательной обработки узлом с указанной ролью. Тело сообщения такого признака не имеет, но конечный получатель все равно должен его обработать или выработать ошибку.

Реализация взаимодействия на основе SOAP очень напоминает реализацию RPC. Клиент делает вызов процедуры, являющийся обращением к процедуре-заместителю, размещенной в переходнике. Переходник перенаправляет вызов в подсистему SOAP, где вызов преобразуется в XML-документ. Полученное сообщение преобразуется к формату HTTP и передается на удаленный сервер. Там происходят обратные преобразования: модуль HTTP, получив сообщение, передает его в подсистему SOAP, где снимается оболочка HTTP, извлекается XML-документ, а его содержимое передается серверному переходнику, который вызывает нужную процедуру. Обратно от сервера клиента тем же порядком передается результат работы процедуры.

Протокол SOAP может работать и в асинхронном режиме. При этом запросы отправляются с помощью системы очередей, а в качестве транспортного протокола используется SMTP.

24. ***Описание сетевых служб. Спецификация WSDL.*** *Сходства и отличия языка WSDL от традиционных языков IDL. Структура спецификации сетевой службы. Абстрактная и конкретная части спецификации. Типы базовых операций (односторонние, уведомление, запрос/ответ и просьба/ответ). Типы данных и портов. Интерфейсное связывание, порты, службы. Основные цели, достигаемые использованием WSDL (описание сетевой службы, входные данные для трансляторов переходников, предположения о семантике сетевой службы).*

Роль, отведенную в традиционных системах языкам IDL, играет для сетевых служб основанный на XML язык описания WSDL. Интерфейс описывается в терминах методов, которые поддерживаются сетевой службой, а каждый метод может принимать одно сообщение на входе и возвращать другое сообщение на выходе. В терминах удаленного вызова процедуры эти сообщения содержат входные и выходные параметры вызовов процедур. Файлы с описаниями транслируются в соответствующий язык программирования, при этом генерируются переходники и программы-посредники, делающие обращения к сетевым службам прозрачными. Язык WSDL прячет сетевые службы за некоторым интерфейсом, то есть вызовом метода. Инфраструктура сетевой службы использует WSDL и SOAP для конструирования заместителей объектов на сторонах поставщика и потребителя, поэтому разработчики могут программировать свои приложения, как если бы они выполняли локальные вызовы. Кроме спецификации операций, предлагаемых службой (как в IDL), необходимо описывать механизм доступа к этой службе. Сетевые службы делаются доступными посредством протоколов. Отсутствие централизованной системной платформы приводит к необходимости описания места, где размещена служба.

Спецификации WSDL делят на две части – абстрактную и конкретную (определяющую протоколы связывания и другую информацию). Абстрактная часть построена на определениях типов портов, аналогичных традиционным интерфейсам. Типы портов есть логические наборы операций, определяющих обмен сообщениями. Для правильной интерпретации данных на обоих концах взаимодействия необходимо определить их типы. Сообщения в WSDL представляют собой текстовые документы, разделенные на части, каждая из которых характеризуется именем и типом, который ссылается на тип XML. Вызывая процедуру с двумя параметрами, надо определить сообщение из двух частей, одна из которых будет содержать первый параметр, а другая – второй.

В WSDL имеется четыре типа базовых операций: односторонние, уведомление, запрос/ответ и просьба/ответ. Первые два типа операций связаны с одиночными сообщениями, выполняемыми асинхронно, а последние два – с синхронными парными сообщениями. Сообщения типа уведомление и просьба/ответ инициируются службой, двух других типов – клиентом. Определение абстрактного интерфейса завершается в WSDL привязкой операций к определениям типов портов. Типы портов могут опираться на ранее определенные типы портов. В таких случаях они содержат все операции, входящие во вложенные типы, и дополнительные операции. В перечисленных определениях не определяются ни конкретный транспортный протокол, ни тип кодирования информации, ни сама служба, которая реализует набор типов портов. Разделение на абстрактную и конкретную части спецификации WSDL позволяет переиспользовать абстрактные спецификации. Одни и те же абстрактные спецификации могут использоваться с различными привязками к протоколам и по разным сетевым адресам. Конкретная часть интерфейса WSDL определяется с помощью трех конструкций. Интерфейсное связывание определяет тип кодирования сообщений и привязку к протоколу для всех операций и сообщений, заданных для данного типа порта. Можно определить, что данная операция имеет стиль удаленного вызова процедуры или стиль документа. Интерфейсное связывание может определять, что сообщения некоторой операции должны пересылаться с использованием протокола SOAP с привязкой к протоколу HTTP. Здесь же специфицируются правила кодирования, на основе которых сообщения сериализуются в XML. Порты соединяют информацию интерфейсного связывания с адресами сети (URI). В традиционных системах это не нужно, но для сетевых служб – необходимо. Службы являются логическими группами портов. Это означает, что WSDL-служба может оказаться доступной в Интернете по разным адресам. На практике наиболее частой оказывается ситуация, в которой в одной службе группируются близкие по семантике порты, расположенные в одном месте. Часто в службу включаются порты одного типа, но имеющие привязки к разным протоколам.

Язык описания интерфейсов WSDL может использоваться и в качестве языка описания службы, и в качестве входных данных для трансляторов переходников и других

программных инструментов, которые по описаниям на WSDL могут генерировать переходники и другие дополнительные данные. В будущем описания на WSDL смогут содержать информацию о семантике сетевой службы, но пока семантика находится вне WSDL, поэтому для понимания семантики служб надо привлекать информацию, не входящую в описания службы на WSDL.

25. ***Проблемы регистрации сетевых служб. Реестр UDDI***. *Назначение реестров сетевых служб. Способы классификации информации в реестрах (по алфавиту, по тематике, по способам вызова). Структура информации в реестре (поставщик, служба, шаблон использования, техническая модель). Виды интерфейсов реестров (интерфейс запросов, интерфейс публикации, интерфейс безопасности, интерфейс надзора и передачи прав владения, интерфейс подписки, интерфейс репликации).*

Чтобы сетевые службы имели глобальный характер, процесс опубликования сведений о них и их поиск должны быть стандартизованы. Примером является проект универсальных средств описания, обнаружения и интеграции (UDDI), состоящий из реестра и прикладного программного интерфейса. Реестр эквивалентен серверу именования, прикладной интерфейс UDDI определяет, как опубликовать службу, что необходимо для регистрации, как делать запросы к службе. Информация в реестре используется для написания клиентских программ и для обеспечения динамического поиска службы. Информация, занесенная в реестр UDDI, группируется там в алфавитном порядке имен предприятий, поставляющих сетевые службы для использования, по тематике, к которой относится как деятельность поставщиков, так и сами сетевые службы, и по способам вызова сетевых служб (в виде ссылок на документы, хранящиеся вне реестра).

При занесении в реестр сетевая служба сопровождается информацией о предприятии (имя, адрес), о наборе сетевых служб предприятия (оно может поставить несколько служб, а служба поставляется одним предприятием), о шаблоне использования (адрес сетевой службы, набор ссылок на документы с описаниями интерфейсов, одна служба может содержать несколько шаблонов, шаблон относится к одной службе), о технических моделях (интерфейсы служб, классификация, протоколы взаимодействия, сведения о семантике служб). На технические модели могут ссылаться любые объекты реестра (предприятия, службы, шаблоны), но они не привязаны ни к одному из них. Для опубликования службы в реестре сначала определяют технические модели, затем публикуют сведения о предприятии, общую информацию о службах, затем техническую информацию о каждой реализации и доступе к службам со ссылками на технические модели.

Фактическое описание находится в документах, ссылки на которые приводятся в технической модели. Каждая техническая модель, зарегистрированная в реестре, имеет уникальный ключ, по которому разработчики имеют возможность организовывать динамический поиск служб, ссылающихся на них. Технические модели могут использоваться для классификации служб. Описание классификации помещается в обзорные документы, в технических моделях остаются только признаки, которые можно использовать как поисковые ключи. Технические модели могут ссылаться на другие технические модели.

Работу с реестром могут проводить поставщики служб, клиенты и другие реестры. Для разных типов пользователей реестры поддерживают разные точки входа (URI), взаимодействие с которыми осуществляется обменом XML-документами, обычно по протоколу SOAP. Реестр имеет разные виды прикладного программного интерфейса. Интерфейс запросов реестра содержит операции для поиска записей и операции, позволяющие получить описания объекта. Интерфейс используется разработчиками и клиентами для динамической привязки. Интерфейс публикации реестра служит для поставщиков служб. Интерфейс безопасности реестра позволяет пользователям проходить аутентификацию. Интерфейс надзора и передачи прав владения реестра позволяет передавать часть своих прав от одного поставщика служб другим. В любых условиях

владельцем записей в реестрах всегда является реестр, в котором запись была первоначально создана. Модификации записей проводятся только в реестрах-владельцах, но права можно явно передать другим реестрам. Интерфейс подписки на реестр позволяет подписываться на новые или модифицированные службы. Интерфейс репликации помогает реплицировать информацию, обеспечивая синхронность различных реестров.

Использование реестров описания служб, которые делаются разработчиками на WSDL. Определения WSDL-интерфейса регистрируются в качестве технических моделей. Пользователь может отправить сообщение о поиске технической модели (с классификационными признаками), в ответ на которое будет получен список всех ключей технических моделей. Затем можно отправить сообщение о выдаче технической модели, ответом на которое будут технические модели искомого описания интерфейса. После этого можно изучить поле обзорного документа прочитанной технической модели и извлечь содержимое документа с описанием WSDL-интерфейса службы.

Держателями собственных реестров являются многие компании, в том числе IBM (https://uddi.ibm.com/ubr/registry.html) и Microsoft (http://uddi.microsoft.com/default.aspx).

26. ***Координация работы сетевых служб. Координационные протоколы. Ролевые протоколы***. *Понятия "разговора" и "роли". Модели описания разговоров (диаграмма последовательности, диаграмма активности). Ролевой фрагмент протокола. Горизонтальные (системная поддержка, транзакционность, безопасность, надежность) и вертикальные (отраслевые) протоколы.*

Работа сетевых служб проводится следующим образом. Текст ее описания на WSDL хранится у поставщика службы. Компилятор языка WSDL создает серверный переходник (обычно в виде сервлета) и регистрирует его на маршрутизаторе SOAP. Маршрутизатор при обращении к определенному ресурсу (URI) должен вызвать зарегистрированный переходник, который будет вызывать исходный объект. Этим реализация службы завершается. Теперь надо опубликовать в реестре техническую модель, ссылающуюся на автоматически полученное описание на WSDL, а затем опубликовать в реестре полноценную запись, в которой должны содержаться сведения об адресе, по которому служба доступна, и ссылка на техническую модель. Привязка к портам (статическая или динамическая) выполняется реестрами служб.

Для организации взаимодействия служб друг с другом надо следовать протоколам координации сетевых служб, которые строятся на основе понятия "разговора". Для описания протоколов координации служб важным является понятие "роли". Участники взаимодействия "играют" свои роли, обмениваясь сообщениями. Протокол накладывает ограничения на порядок, в котором такой обмен может происходить. Термин "разговор" относится ко всякой последовательности операций (обменов сообщениями), возникающей между клиентом и службой в процессе обращения к службе. Термин "координационный протокол" относится к спецификации набора допустимых разговоров. Существующие стандарты, в частности, стандарт языка описания разговоров сетевых служб WSCL пока не получили достаточной поддержки, поэтому для описания координационных протоколов применяются различные модели. Среди используемых моделей выделяются две: модель диаграмм последовательности и модель диаграмм активности. При описании координационных протоколов указывают роли участников.

Диаграммы последовательности распространены ввиду их простоты и интуитивной ясности, однако, чем сложнее протокол, тем сложнее становится разобраться в этих диаграммах. Иногда прорисовывают несколько диаграмм (как при описании протокола TCP), но диаграмм может стать слишком много. Часто применяют диаграммы активности, с помощью которых показываются альтернативные и параллельные ветви разговоров.

Зная координационный протокол и роль, которую сетевая служба играет в этом протоколе, разработчики проектируют приложения, взаимодействующие с этой службой.

Координационные протоколы помогают организовать динамическую привязку к службе, поскольку приложения, разработанные для участия в некотором протоколе, могут ограничить поиск сетевых служб в реестрах только теми, которые играют определенные роли. Координационные протоколы обычно хранятся в реестрах в виде технических моделей, а роли, исполняемые службой, могут указываться в шаблонах использования, ссылающихся на модели.

Координационные протоколы не раскрывают деталей реализации, что облегчает внесение изменений в программы взаимодействия служб, пока изменения в программах не затрагивают протокол. Ролевые протоколы являются подмножествами полных протоколов, но содержат только те операции и сообщения, которые необходимо знать исполнителю определенной роли. Координационные протоколы сетевых служб делятся на вертикальные и горизонтальные. Вертикальные протоколы относятся к отдельным отраслям (и называются бизнес протоколами). Обычно в них описывается, каким образом следует строить конкретные бизнес транзакции, определяются форматы соответствующих документов, семантика содержимого этих документов.

Многие важные детали реализации в вертикальных протоколах отсутствуют, которые концентрируются на семантике обменов, а также на наборах правильных разговоров. С деталями имеют дело горизонтальные протоколы, в которых определяется общая инфраструктура, не зависящая от прикладной области. Их цель – наделить обмены, проходящие между службами высокоуровневыми абстракциями, реализованными в системном слое сетевых служб прозрачно для разработчиков самих служб. Сложность сетевых служб в том, что они предназначены для взаимодействия не внутри, а между предприятиями, и многие ранее разработанные методы для них не пригодны. В частности, из-за длительности взаимодействия протоколы подтверждения типа 2PC использоваться не могут, так как они в момент этого подтверждения блокируют ресурсы. Следует разрабатывать новые стандарты (WS-Coordination, WS-Transaction).

27. ***Централизованная и децентрализованная (распределенная) координация***. *Контроллер "разговоров" (идентификация объектов, верификация "разговоров" на соответствие протоколу). Координаторы и участники "разговоров". Координационный протокол, координационный тип, координационный контекст. Типы портов координатора и участника.*

Программы, предназначенные для выполнения разговоров служб, называются контроллерами разговоров. Они маршрутизируют разговоры и верифицируют их соответствие протоколу. Обычно контроллеры разговоров включаются в состав маршрутизаторов SOAP. Маршрутизация разговоров связана с проблемой диспетчеризации сообщений: необходимо, чтобы эти сообщения попадали нужным внутренним объектам. Если служба реализована как единый объект, и этот объект должен управлять всеми разговорами, то реализация объекта должна обеспечить отслеживание различных состояний и содержать программы, необходимые для понимания, какому разговору принадлежит поступившее сообщение. Альтернативой является создание объекта для каждого разговора и передача функций по управлению разговорами системным программам. Контроллер разговоров может включать в заголовки каждого сообщения уникального для данного разговора идентификатора. Такой идентификатор должен генерироваться каждый раз в начале нового разговора, и при получении контроллером сообщения должен в нем отыскиваться, указывая нужный объект (например, компонент EJB сервера приложений J2EE). При обнаружении несоответствий контроллер может возбуждать сообщение об ошибке.

Стандарт WS-Coordination определяет метод передачи уникальных идентификаторов сетевым службам, взаимодействующим между собой, в частности, определяется координационный контекст и то, как он включается в заголовки сообщений SOAP. С

помощью интерфейса регистрации портов определяется метод передачи контроллерам разговоров сведений о портах участников разговоров. С помощью интерфейса активации определяется метод передачи контроллерам разговоров сведений об их ролях в разговоре. Взаимодействие между координаторами и участниками разговоров может осуществляться централизованно и распределенно. Координационный протокол есть набор правил управления разговорами между координатором и участниками (пример – 2PC). Координационный тип есть набор связанных друг с другом координационных протоколов. Координационный тип атомарных транзакций может включать в себя группу из 2PC и протокола уведомления, который выполняется между участниками, желающими получить информацию о результате 2PC. Координационный контекст есть структура данных, используемая для отметки сообщений, относящихся к одному разговору.

При активации участник требует от координатора создать новый координационный контекст. Новые контексты создаются всякий раз, когда участник создает новый экземпляр координационного типа (разговор). Участник регистрируется у координатора как участник координационного протокола, после чего координатор и участники обмениваются сообщениями, специфичными для данного прикладного протокола. Взаимодействия, проводимые для активации и регистрации, не зависят от типа координации (горизонтальны).

Для активации определяются два типа портов: порт активации координатора и порт активации участника. Через порты таких типов службы просят своих координаторов создавать новые координационные контексты, а те возвращают службам ссылки на созданный контекст. Во время регистрации все участники, желающие участвовать в протоколе, должны зарегистрироваться у своих координаторов. Для регистрации служба посылает сообщение на соответствующий порт координатора, указывая имя протокола и ссылку на свой интерфейс, по которому будет доступен регистрируемый протокол. В ответ координатор возвращает ссылку на свой интерфейс этого же протокола (например, 2PC). После проведения активации и регистрации координатору известно, кто будет принимать участие в координации, а также порты, на которые надо посылать сообщения. О частных протоколах делается предположение, что все они имеют парный характер, то есть для каждого из них определен тип порта участника и тип порта координатора.

Отличие децентрализованной координации от централизованной заключается в том, что участники могут регистрироваться у разных координаторов (каждый создает свой контекст), а координаторы должны регистрироваться друг у друга (один из них объявляет себя координатором всего взаимодействия, а другие участвуют в качестве посредников). Таким образом могут строится цепочки произвольной сложности. В соответствии со стандартом WS-Coordination координаторы могут транслировать сообщения одного протокола, получаемые от их участников, в сообщения другого протокола, передаваемые другому координатору. Это требует реализации зависящих от конкретных протоколов компонентов, выполняющих активацию и регистрацию.

28. ***Транзакции в сетевых службах****. Откат и компенсация. Бизнес активности и атомарные транзакции. Подтверждение с нулевой фазой.*

Сохранение базовых свойств транзакций при выполнении транзакций сетевыми службами невозможно. Для сетевых служб не существует определений терминов "ресурс", "блокировка", "подтверждение" и "откат". Работа с сетевыми службами приводит к ослаблению строгости свойств транзакций и переходу к компенсационным механизмам. Если транзакцию надо отменить, служба выполнит компенсационную операцию и семантически отменит результаты транзакции.

Стандарт WS-Transaction определяет набор протоколов, требующих скоординированной работы нескольких сторон и строится на базе протокола WS-Coordination. Стандарт WS-Transaction предполагает существование набора сетевых служб, участвующих в транзакции и одного или нескольких координаторов, централизованных или

распределенных. Стандарт WS-Transaction также определяет структуру координационного контекста и стандартные WSDL-интерфейсы, которые должны реализовываться участниками и координаторами. Стандарт WS-Transaction определяет стандартный протокол для долгих транзакций, называемых бизнес активностями. Для работы с короткими транзакциями в доверительных зонах стандарт WS-Transaction определяет также набор спецификаций, называемых атомарными транзакциями.

Тип атомарных транзакций состоит из нескольких координационных протоколов, исполняемых сетевыми службами участниками или координаторами, в зависимости от того, что должно делаться на протяжении той или иной фазы распределенной транзакции. В этот координационный тип входят протоколы завершения, 2PC, завершения с уведомлением, нулевой фазы и уведомления о результате. Протокол завершения нужен для информирования координатора о необходимости инициировать протокол 2PC и опросить участников об успешности транзакции. В некоторых случаях координатор до 2PC проводит с участником обмен по протоколу нулевой фазы, оповещая его о предстоящем начале 2PC. Участник может провести подготовительные работы, (сбросить информацию из буферов). Участник может запросить координатора о результате транзакции, выполняя протокол уведомления. Протокол завершения с уведомлением может инициироваться сетевой службой вместо обычного протокола завершения в тех случаях, когда нужно, чтобы координатор хранил результат транзакции, не уничтожая, до получения специального уведомления от сетевой службы о том, что этот результат ею получен. Для каждого из пяти протоколов стандарт вводит по два типа портов, которые должны быть реализованы координатором. Один из пары типов портов позволяет координатору выполнять протоколы в качестве координатора (для организации координационных цепочек), а второй – в качестве участника. Сетевые службы всегда являются только участниками взаимодействий.

Стандарт WS-Transaction определяет структуру транзакционного контекста. Эта структура возвращается координатором в ответ на запрос о создании координационного контекста. Она же передается как часть заголовков сообщений SOAP и показывает, что сообщение посылается в рамках некоторого разговора. Семантика понятий "подтверждение" и "отказ от подтверждения" определяется неформально.

Для управления длительными бизнес транзакциями без блокировки ресурсов используется протокол 2PC и координационный тип, называемый бизнес активностью, который содержит два протокола: бизнес соглашение с завершением участника и бизнес соглашение с завершением координатора. Протокол бизнес соглашения с завершением участника инициируется участником для информирования координатора о состоянии выполнения (прервано, завершено, завершено с ошибкой). Координатор отвечает всем участникам сообщениями типа закрыть, завершить, компенсировать или ошибка. Протокол бизнес соглашения с завершением координатора отличается тем, что координатор предварительно дает возможность участнику подготовиться к операциям завершения или компенсации. Координатор в соответствии со стандартом имеет две пары типов портов (одна пара - для соглашения с завершением участника, а вторая для соглашения с завершением координатора).

Для каждой бизнес активности и каждой сетевой службы может определяться только одна компенсационная операция. Это означает, что если сетевая служба в рамках некоторой бизнес активности выполняет в некотором порядке серию заданий, все эти задания должны отменяться в совокупности. Откат серии заданий в обратном порядке, например, отправкой серии компенсационных сообщений по каждому заданию, никак не поддерживается, поскольку в противном случае от координатора потребовалось бы знать порядок, в котором службы обмениваются операционными сообщениями.

29. ***Композиция сетевых служб***. *Назначение композиции. Основные элементы системной поддержки композиции сетевых служб (композиционная модель и язык, окружение разработки, окружение выполнения). Композиция и координация.*

Координация служб позволяет нескольким службам участвовать в одном разговоре между собой, их композиция обеспечивает службам возможность одновременно вести несколько разговоров с разными службами. Различные сетевые службы могут поддерживать разные протоколы, клиент сам реализует все протоколы, необходимые для обращения к службам.

Композиция сетевых служб итеративна, добавлением составных компонентов она позволяет создавать сколь угодно сложные приложения. Клиентом может быть сетевая служба, которая может рассматриваться в качестве компонента более высокоуровневой (композитной) службы.

Композиция сетевых служб не предполагает физической интеграции компонентов. Сетевые службы это не библиотеки прикладных программ, а интерфейсы. Композиция сетевых служб эквивалентна указанию, к каким службам надо обратиться, в каком порядке это сделать, как надо управлять исключительными ситуациями. Базовые компоненты остаются отделенными от композитных служб.

Чаще всего программирование композитных сетевых служб ведется на традиционных языках. Для проведения композиции в системах TRPC в транзакционных мониторах применяются расширенные языки Транзакционный Си и Транзакционный Си++. Эти расширения, обычно принимающие форму библиотек и процедурных переходников, вводят языковые средствами, необходимые для композиции. Стандарты сетевых служб тоже приводят к такому подходу, поскольку протокол SOAP превращает вызовы сетевых служб в вызовы удаленных процедур. Композитная сетевая служба при этом реализуется на уровне традиционных системных платформ как самая обычная (не композитная) служба. В получающихся программах бизнес логика оказывается перемешанной с низкоуровневыми деталями, что привело к созданию моделей высокого уровня, сходных с понятием рабочего потока. Для композиции сетевых служб предложены различные языки: XL, WSFL, BPML, BPEL.

Композиция служб определяет, как надо реализовывать сетевую службу, соединяя между собой другие службы. Системная поддержка должна заключаться в определении абстракций и инструментария, которые помогут четко описать и исполнить запрос к сетевой службе, позволив разработчикам концентрироваться не на низкоуровневых деталях, а на бизнес логике. Композиционную модель и язык композиции позволяют описывать объединяемые службы, порядок в котором к ним надо обращаться, и способ определения параметров обращений к службам (способ построения сообщений). Спецификация композитной службы, выраженная на языке композиции, называется композиционной схемой. Схема определяет бизнес логику композитной сетевой службы, она может выглядеть как программа, написанная на языке, специально разработанном для композиции. Окружение разработки обычно характеризуется графическим пользовательским интерфейсом, с помощью которого разработчики могут создавать композиционные схемы и графы зависимостей, обозначая порядок, в котором надо обращаться к службам. Графы и другая описательная информация транслируются в текстовые спецификации (композиционные схемы). Окружение выполнения (композиционный мотор) выполняет бизнес логику композитной службы, обращаясь к службам компонентам, определенным в схеме. Системной поддержки композиции позволяет реализовывать композитные службы на системном уровне сетевых служб, а не на уровне традиционной системной поддержки.

Композиция служб связана с реализацией операций внутри сетевых служб. Спецификации не сообщают клиентам и нигде не регистрируют. Они предназначены для системных слоев сетевых служб, которые автоматизируют композицию, обращаясь к операциям, предлагаемым другими сетевыми службами в соответствии с композиционной схемой. С точки зрения клиента все равно, является сетевая служба композитной или нет.

Имеется четкое различие между внутренней композицией и внешней координацией сетевых служб. Координационные протоколы – это общедоступные документы, создаваемые

на основе стандартных языков. Эти документы регистрируются в реестрах с целью поддержки поиска при разработке и привязки при выполнении. Разговоры, подчиняющиеся координационным протоколам, поддерживаются контроллерами разговоров, цель которых связана не с выполнением бизнес логики, а с диспетчеризацией сообщений, приходящих для внутренних объектов, и с верификацией правил протоколов.

30. ***Виды композиционных моделей***. *Компонентная модель (типы компонентов и предположения о них), оркестровая модель, модель данных и доступа к данным (данные приложений и данные управляющего потока, передача данных – журнальный подход и явный поток), модель выбора службы (статическое связывание, динамическое связывание по ссылке, динамическое связывание с поиском, динамический выбор операции), транзакции (атомарные области, саги), управление исключительными ситуациями (включение в общий поток управления, ассоциирование с активностями, введение правил).*

Терминология, применяемая при описании композиционных моделей, близка к терминологии СУРП. Компонентная модель определяет объединяемые элементы в терминах предположений о таких компонентах. Оркестровая модель определяет абстракции и языки, используемые для определения порядка вызова служб. Модель данных и доступа к данным определяет методы описания данных и обмена данными между компонентами. Модель выбора службы определяет способ статической или динамической привязки. Транзакционная модель определяет транзакционную семантику, которая ассоциирована с композицией. Управление исключениями определяет способ управления исключительными ситуациями в работе службы.

Компонентная модель. Одни службы отличаются от других типами компонентов и предположениями об этих компонентах. Модель может предполагать, что компоненты реализуют некоторый набор стандартов (HTTP, SOAP, WSDL и WS-Transaction). Такие предположения снижают гетерогенность системы. Предположения можно ограничить, приняв, что компоненты обмениваются XML-сообщениями. Преимущество: более общая модель, недостаток: усложнение работы из-за гетерогенности. Могут применяться промежуточные решения, приводящие к использованию сложных языков и сложных систем с множеством форматов и протоколов. Язык композиции BPEL предполагает, что компонентами являются службы, описанные на WSDL.

Оркестровая модель. Оркестровка позволяет различным службам организоваться в единое целое. В ней описывается порядок вызова служб и условия, при которых определенная служба может вызываться или не вызываться. Применяются разные модели описания оркестровок: диаграммы активности, сети Петри, π-исчисление, диаграммы состояний, иерархии активностей, оркестровка на основе правил.

Модель данных и доступа к данным. Данные, используемые при композиции служб, делятся на данные приложений и управляющие данные. Прикладные данные – параметры, посылаемые или получаемые в сообщениях. Управляющие данные используются для вычисления условий перехода, они используются композиционным мотором при выполнении процесса. Обычно переменных процесса мало, их типы ограничены строковыми, целыми, вещественными, а иногда и составными типами. Значения управляющих данных обычно извлекаются из сообщений, получаемых сетевыми службами. Прикладные данные более сложны и разнообразны.

Для передачи данных между активностями, существуют два подхода: журнальный и подход явного потока данных. Журнальный подход аналогичен языкам программирования: все данные композитной службы именуются и перечисляются явно. Модификации переменных выполняются при получении сообщения "атомарно", прежние значения переменных стираются. Активности могут иметь разный уровень доступа к переменным (чтение/запись или только чтение). Каждый запуск активности заводит отдельный журнал.

Подход явного потока данных требует, чтобы в дополнение к потоку управления явным элементом композиции был поток данных. На диаграммах активностей прорисовываются линии, и разработчик указывает, что входные данные одной активности (используемые, например, для сообщений, составляющих операции) должны браться из результатов ранее выполненной активности. Потоки данных гибче журналов, но сложнее: они создают неявные управляющие зависимости, поскольку активности, являющиеся источниками данных должны завершаться до начала работы активностей, эти данные получающих.

Модель выбора службы. Композиционная схема описывает посылаемые и получаемые сообщения, а также порядок, в котором проводятся обмены. Для выполнения композиционной логики мотор должен в дополнение к схеме знать еще, какая служба является получателем сообщения. В композиционных схемах эта информация указывается абстрактно (вместо номера порта его тип или роль получателя). Перед отправкой сообщения все типы портов должны заменяться номерами. Способы привязки к службе: статическое связывание, динамическое связывание по ссылке, динамическое связывание с поиском и динамический выбор операции. Вставка адреса службы в спецификацию композитной службы полезна при построении прототипов и тестировании. Недостаток: трудность отслеживания изменений URI служб, а также то, что все отдельные запуски композитной службы всегда обращаются к одной и той же службе. При использовании подхода ссылок URI служб берутся из переменных процесса. Присваивание указателя переменной происходит в результате выполнения предыдущей операции, его можно извлекать из указателя клиента, обратившегося к композитной службе, он может быть явно задан при установке службы на машине. Если указатель должен быть динамически извлечен из справочника, этому действию надо сопоставить отдельную активность, то есть операцию интерфейса сетевой службы некоторого реестра, которая определяет указатель службы и записывает его в некоторую переменную. Динамическое связывание с поиском позволяет для каждой активности описывать запрос к справочнику. Результат обработки запроса используется для определения службы, которую надлежит вызывать, при этом необходимо формулировать критерии выбора службы из списка. Композиционные модели могут проводить динамическое связывание не только на уровне целых служб, но и на уровне отдельных операций. Такой подход называется динамическим выбором операции. Этот подход усложняет оркестровку, которой при росте вариантов выбора, становится трудно управлять. Динамические операции полезны в тех случаях, когда набор параметров операции меняется, в зависимости от выбранной службы.

Транзакционное поведение композитных служб определяется внедрением в оркестровую схему атомарных областей. На диаграммах такие области окружают наборы активностей, обладающие свойством "все или никто". Атомарность достигается выполнением протоколов 2PC, которые реализуются системой, не требуя от разработчика программирования. Решение проблемы менее строгой транзакционной семантики связано с применением компенсаций, когда результаты подтвержденных операций отменяются выполнением других операций. Применение такого подхода означает, что при возникновении ошибки для осуществления частичного отката операций атомарной области системный слой сетевых служб должен инициировать и выполнить протокол компенсации подтвержденных активностей. Транзакционная модель саг позволяет разбивать длительные транзакции на подтранзакции, имеющие традиционные транзакционные свойства и исполняемые в некотором предопределенном порядке. При завершении они подтверждаются, снимая блокировки и показывая результат другим транзакциям. Откат саги выполняется прерыванием всех активных подтранзакций и компенсацией подтвержденных подтранзакций в порядке, обратном выполнению. Ответственность за реализацию компенсации возлагается на разработчика службы, то есть на стороны компонентов, а не на сторону композиции. Если компонент уже имеет компенсационную логику, разработчик композиции освобождается от ее реализации, а обращается к компенсационной операции.

Управление исключениями. Исключения обычно вызываются ошибками в системе или в вызываемых приложениях, либо могут быть ситуациями, предусмотренными семантикой сетевой службы. При использовании подходов, основанных на потоках, в конце каждой операции результат проверяется на наличие ошибок, и, если ошибка есть, выполняются соответствующие действия. В случаях, когда вызванная операция не возвращает никакого результата, для каждой активности вводят таймауты, при срабатывании которых мотор останавливает активность. Методика "попытка-перехват-возбуждение" ассоциирует исключение с группой активностей и вводит логическое условие, связанное с композиционными данными. Если условие выполнено, выполняется часть программы, управляющая исключением. После этого, в зависимости от спецификации исключения, процесс может повторить активность, повторить обработку исключения или просто остановиться. Такая методика полезна, если оркестровая модель допускает определение групп активностей и ассоциирования с этими группами отдельных свойств, или если оркестровка может быть структурирована в иерархии. Преимущества: отделение обычной логики от логики исключений, структурирование определения исключений, возможность описания стратегии продолжения работы после завершения обработки исключения.

Подходы, основанные на правилах, управляют исключениями на основе правил "событие-условие-действие", с помощью которых событие облекается в форму сообщений, посылаемых композитной службе, или в форму таймаутов. Условие есть логическое выражение, которое проверяет, действительно ли событие относится к исключительной ситуации, которую надо обработать вызовом операции или прерыванием транзакции. Правила обычно определяются в некотором текстовом виде, дополняя графическую нотацию, определяемую оркестровкой. Подходы, основанные на правилах, разделяют нормальное и исключительное поведение процесса, но в системе появляется еще один язык, который надо изучать и интерпретировать.

31. ***Виды оркестровых моделей***. *Диаграммы активностей, диаграммы состояний, сети Петри, π-исчисление, иерархии активностей, оркестровка на основе правил.*

Оркестровая модель строится, чтобы помочь различным сетевым службам организоваться в единое целое. В ней описывается порядок вызова служб и условия, при которых служба может вызываться. Применяются разные модели описания оркестровок: диаграммы активности, сети Петри, π-исчисление, диаграммы состояний, иерархии активностей, оркестровка на основе правил.

Модель диаграммы активностей определяет оркестровку с помощью последовательности операций. Активности моделируют сообщения, приходящие к службе и исходящие от нее. Если полученное сообщение вызывает двухстороннюю операцию, композиционная схема включит активность "ответить", которая пошлет ответ клиенту. Получение сообщений, вызывающим односторонние или двухсторонние операции, предлагаемые композитной службой, моделируются блокирующей активностью "получить". Обращения к синхронным (запрос/ответ) операциям, предлагаемым другой сетевой службой, моделируются блокирующей активностью "вызвать". Уведомления о сообщениях другим сетевым службам (вызовы односторонних операций, предлагаемых сетевыми службами) моделируются неблокирующей активностью "послать".

Диаграммы активности содержат определения служб (URL), которым отправляются сообщения, способы построения сообщений на основе результатов предыдущих активностей, способы обработки исключительных ситуаций. В отличие от диаграмм активностей в описаниях коммуникационных протоколов, в этих диаграммах полностью специфицируются все условия и элементы данных, поскольку внутренние спецификации служб не доступны.

Формализм диаграмм состояний основан на расширенной автоматной схеме. Он определяет активности, выполняемые при входе в состояние, при выходе из него или при

нахождении в состоянии, а также описывает события, условия и действия, связанные с переходами, которые нужно делать при возникновении события, если истинно некоторое условие. Имеются расширения в виде составных состояний, составных переходов, параллельных состояний, синхронизации после выполнения параллельных составных состояний. В диаграммах состояний активности "скрыты", зато возникает явное понятие состояния, отсутствовавшее на диаграммах активности. Смысловые имена состояний облегчают получение информации о выполнении служб.

Оркестровая модель на сетях Петри объединяет диаграммы активности и диаграммы состояний. Свойства сетей Петри четко определены и имеют понятную семантику. Для работы с сетями Петри созданы многочисленные системы автоматического анализа, с помощью которых исследуются свойства спецификаций и обнаруживаются потенциально ошибочные места. Переходы отмечаются логическими предикатами, управляющими переключением состояний.

π-исчисление есть алгебра процессов, на которой созданы современные языки композиции, (XLANG, BPEL). Она строится на основе понятий взаимодействия последовательных процессов, алгебры взаимодействующих процессов и абстракций и исчисления параллельных систем. Достоинство: наличие точного и изученного формализма верификации свойств процесса. π-исчисление вводит конструкции для композиции активностей в терминах последовательного, параллельного или условного выполнения.

В модели иерархии активностей процессы описываются с помощью дерева активностей. Конечные узлы представляют собой исполняемые активности, промежуточные узлы определяют порядок следования активностей более низких уровней. Недостаток: необходимость осуществлять "искусственные" шаги и отслеживать хронологический порядок следования шагов. Преимущество: изучение оркестровки с разных уровней абстракции, что делает определение процесса модульным.

Оркестровка на основе правил часто используется в системах, которые отслеживают факты возникновения интересующих их событий, в частности, событий, сигнализирующих о критических условиях. Фиксация события приводит к выполнению действия, управляющего ситуацией. Для этого правила записываются в виде пар <событие-действие>. Правило может содержать условие, то есть логический предикат над параметрами события, вычисляемый при обнаружении события. Правила позволяют также моделировать асинхронные события, то есть события, которые могут произойти на любой стадии процесса, что делает их пригодными для определения логики управления исключительными ситуациями, которые по своей природе асинхронны. Недостаток: трудность понимания логики процессов, описанных большим числом правил.

32. ***Координация композитных служб***. *Основные отношения между координационными протоколами и композицией сетевых служб. Процесс проектирования композитной службы (описание ролевых протоколов, .определение процесса обмена сообщениями, сопоставление сообщений и активностей, абстрактные протоколы). Проблема маршрутизации и корреляционная информация.*

Определение координационного протокола накладывает ограничения на композиционную схему сетевой службы, реализующей логику протокола. Композиционная схема сетевой службы должна включать активности, получающие и отсылающие сообщения, предписанные протоколом.

Чтобы создать сетевую службу, надо создать ролевой фрагмент протокола, который должен включать все обмены сообщениями, затрагивающие данную роль, а затем перейти от ролевой части протокола к определению процесса обмена сообщениями, предписанному ролью и включающему активности, отправляющие и получающие сообщения протокола. Для получения композиционной схемы, необходимо добавить бизнес логику, способную в заданном протоколе играть назначенную роль. Для каждой вызываемой операции,

отмеченной в роли, надо поставить в соответствие активность (или пару активностей) процесса. Двухсторонние операции (запрос/ответ), вызываемые некоторой ролью в отношении роли создаваемой сетевой службы, соответствуют активностям "получение" и "ответ". Односторонние операции, вызываемые в отношении создаваемой службы другими ролями, моделируются активностями "получение". Односторонние операции, вызываемые создаваемой службой в отношении других ролей, моделируются активностями "отправка". Двухсторонние операции (запрос/ответ), инициированные создаваемой службой, моделируются как активности "вызов".

Созданный абстрактный процесс есть эквивалентное представление ролевого фрагмента, но вместо описания того, что внешняя служба посылает сообщение, в нем указывается, что создаваемая служба должна получить, отправить, вызвать, ответить на сообщение. Абстрактный процесс определяет протокол, поскольку информация, содержащаяся в описании процесса та же, что и в ролевом фрагменте протокола. Вместо описания внутренней логики и условий здесь определяется видимое поведение сетевой службы, поэтому процессы называются открытыми. Выполняться абстрактный процесс не может, его определение может только передаваться контроллеру разговоров, который проверяет, что обмен сообщениями происходит в соответствии с протоколом. Композиционный мотор не сможет с ним работать потому, что ему нужно знать, как строить сообщения и как вычислять условия ветвления.

Преимущества абстрактных процессов в том, что они облегчают понимание того, как надо ограничить композицию и как определить композиционную схему, реализующую протокол. Расширение абстрактного протокола необходимыми деталями легко приведет разработчиков к композиционной схеме. На практике сетевые службы должны поддерживать несколько протоколов одновременно и вести сразу несколько разговоров. Простого решения для этого не существует, поскольку различные протоколы не зависят друг от друга и только реализация может установить зависимости разговоров. Некоторые современные языки (BPEL, ebXML) могут описывать и внешнее поведение (абстрактные процессы) и внутреннюю реализацию (выполняемые процессы).

Разработка архитектуры композитной службы на основе композиционного мотора сталкивается с проблемами маршрутизации. Контроллер проверяет соответствие сообщений протоколу и направляет их в мотор. С точки зрения контроллера мотор представляет собой внутренний объект, реализующий разговор. Поскольку один мотор выполняет множество композиционных запусков, к которым поступают все сообщения, относящиеся к этим запускам, мотор должен уточнять, к какому конкретно запуску надо направить каждое конкретное сообщение, то есть решать задачу, аналогичную задаче маршрутизации. Маршрутизатор SOAP при передаче сообщений композиционному мотору может оставлять их информационные заголовки, и для определения места назначения можно использовать координационный контекст. Если же контроллер отсекает заголовки из сообщений SOAP, то можно включать в композиционную схему корреляционную информацию, на основе параметров сообщений определяя логику, по которой сообщения могут быть ассоциированы с композиционными запусками. Если каждое сообщение из разговора несет некоторый "номер" в качестве одного из параметров, а каждый разговор имеет уникальный "номер", разработчик композиции может установить корреляцию сообщений и тех запусков, для которых значение переменной "номер" соответствует "номеру", содержащемуся в сообщении. Это приводит к потере прозрачности маршрутизации, а выполнимо только, когда сообщения включают информацию, которая годится для идентификации разговора. В будущем контроллеры и композиционные моторы смогут использовать средства стандартных интерфейсов.