КОРПОРАТИВНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

Вернутся

1 Основы и основные понятия корпорации и КИС

Термин **корпорация** происходит от латинского слова corporatio - объединение. Корпорация обозначает объединение предприятий, работающих под централизованным управлением и решающих общие задачи.

Как правило, корпорации включают предприятия, расположенные в разных регионах и даже в различных государствах (транснациональные корпорации).

В самом общем смысле термин **Корпорация** означает *объединение предприятий, работающих под централизованным управлением и решающих общие задачи. Корпорация является сложной, многопрофильной структурой и вследствие этого имеет распределенную иерархическую систему управления*

. **Корпоративное управление** определяется как система взаимоотношений между акционерами, советом директоров и правлением, определенные уставом, регламентом и официальной политикой компании, а также принципом главенства права на основе принятой бизнесмодели.

Бизнес-модель – это описание предприятия, как сложной системы, с заданной точностью. В рамках бизнес-модели отображаются все объекты (сущности), процессы, правила выполнения операций, существующая стратегия развития, а также критерии оценки эффективности функционирования системы. Форма представления бизнес-модели

и уровень её детализации определяются целями моделирования и принятой точкой зрения.

Предприятия, отделения и административные офисы, входящие в корпорацию, как правило, расположены на достаточном удалении друг от друга. Их информационная связь друг с другом образует коммуникационную структуру корпорации, основой которой является информационная система.

Информационная модель – подмножество бизнес-модели, описывающее все существующие (в том числе не формализованные в документальном виде) информационные потоки на предприятии, правила обработки и алгоритмы маршрутизации всех элементов информационного поля.

Информационная система (ИС) – это вся инфраструктура предприятия, задействованная в процессе управления всеми информационно-документальными потоками, включающая в себя следующие обязательные элементы:

- Информационная модель, представляющая собой совокупность правил и алгоритмов функционирования ИС. Информационная модель включает в себя все формы документов, структуру справочников и данных, и т.д.
- Регламент развития информационной модели и правила внесения в неё изменений.
- Кадровые ресурсы (департамент развития, привлекаемые консультанты), отвечающие за формирование и развитие информационной модели.

- Программное обеспечение, конфигурация которого соответствует требованиям информационной модели (программное обеспечение является основным движителем и, одновременно, механизмом управления ИС). Кроме того, всегда существуют требования к поставщику программного обеспечения, регламентирующие процедуру технической и пользовательской поддержки на протяжении всего жизненного цикла.
- Кадровые ресурсы, отвечающие за настройку и адаптацию программного обеспечения, и его соответствие утвержденной информационной модели.
- Регламент внесения изменений в настраиваемые структуры (специфические настройки, структуры баз данных и т.д.) и конфигурацию программного обеспечения и состав его функциональных модулей.
- Аппаратно-техническая база, соответствующая требованиям по эксплуатации программного обеспечения (компьютеры на рабочих местах, периферия, каналы телекоммуникаций, системное программного обеспечение и СУБД).
- Эксплуатационно-технические кадровые ресурсы, включая персонал по обслуживанию аппаратно-технической базы.
- Правила использования программного обеспечения и пользовательские инструкции, регламент обучения и сертификацию пользователей.

Ресурсы корпораций включают:

1. материальные (материалы, готовая продукция, основные средства)
2. финансовые
3. людские (персонал)
4. знания (ноу-хау)
5. КИС

Система управления любой компании включает *три основные подсистемы:*

1.**Планирование продаж и операций**. Это общий план функционирования предприятия, устанавливающий объемы изготовления готовой продукции. Главным здесь является планирование спроса и оценка ресурсов, необходимых для удовлетворения спроса. Здесь же создается основной производственный план, определяющий, какие изделия, в каком количестве и в какие сроки нужно произвести.

2. **Детальное планирование необходимых ресурсов** (материалов, производственных мощностей, трудовых ресурсов и т.д.). Составленный план определяет время и объем заказов для всех материалов и комплектующих, необходимых для реализации основного производственного плана.

3. **Управление исполнением планов** в процессе производства и закупок (снабжения).

Все эти подсистемы реализуются на основе КИС.

Корпоративные информационные системы (КИС) - ***это интегрированные системы управления территориально распределенной корпорацией, основанные на***

углубленном анализе данных, широком использовании систем информационной поддержки принятия решений, электронных документообороте и делопроизводстве. КИС призваны объединить стратегию управления предприятием и передовые информационные технологии.

Корпоративная информационная система — ***это совокупность технических и программных средств предприятия, реализующих идеи и методы автоматизации.***

Комплексная автоматизация бизнес процессов предприятия на базе современной аппаратной и программной поддержки может называться по-разному. В настоящее время наряду с названием Корпоративные информационные системы (КИС) употребляются, например, следующие названия:

1. Автоматизированные системы управления (АСУ);
2. Интегрированные системы управления (ИСУ);
3. Интегрированные информационные системы (ИИС);
4. Информационные системы управления предприятием (ИСУП).

Главная задача КИС - *эффективное управление всеми ресурсами предприятия (материально- техническими, финансовыми, технологическими и интеллектуальными) для получения максимальной прибыли и удовлетворения материальных и профессиональных потребностей всех сотрудников предприятия.*

КИС по своему составу - это совокупность различных программно-аппаратных

платформ, универсальных и специализированных приложений различных разработчиков, интегрированных в единую информационно-однородную систему, которая наилучшим образом решает в некотором роде уникальную задачу каждого конкретного предприятия. То есть, КИС - человеко-машинная система и инструмент поддержки интеллектуальной деятельности человека, которая под его воздействием должна:

- Накапливать определенный опыт и формализованные знания;
- Постоянно совершенствоваться и развиваться;
- Быстро адаптироваться к изменяющимся условиям внешней среды и новым потребностям предприятия.

Комплексная автоматизация предприятия подразумевает перевод в плоскость компьютерных технологий всех основных деловых процессов организации. И использование специальных программных средств, обеспечивающих информационную поддержку бизнес-процессов, в качестве основы КИС представляется наиболее оправданным и эффективным. Современные системы управления деловыми процессами позволяют интегрировать вокруг себя различное программное обеспечение, формируя единую информационную систему. Тем самым решаются проблемы координации деятельности сотрудников и подразделений, обеспечения их необходимой информацией и контроля исполнительской дисциплины, а руководство получает своевременный доступ к достоверным данным о ходе производственного процесса и имеет средства для

оперативного принятия и воплощения в жизнь своих решений. И, что самое главное, полученный автоматизированный комплекс представляет собой гибкую открытую структуру, которую можно перестраивать на лету и дополнять новыми модулями или внешним программным обеспечением.

Под корпоративной информационной системой будем понимать информационную систему организации, отвечающую следующему минимальному перечню требований:

1. Функциональная полнота системы
2. Надежная система защиты информации
3. Наличие инструментальных средств адаптации и сопровождения системы
4. Реализация удаленного доступа и работы в распределенных сетях
5. Обеспечение обмена данными между разработанными информационными системами и др. программными продуктами, функционирующими в организации.
6. Возможность консолидации информации
7. Наличие специальных средств анализа состояния системы в процессе эксплуатации

Функциональная полнота системы

- выполнение международных стандартов управленческого учета MRP II, ERP, CSRP
- автоматизация в рамках системы решения задач планирования, бюджетирования, прогнозирования, оперативного (управленческого) учета, бухгалтерского учета,

статистического учета и финансового-экономического анализа

- формирование и ведение учета одновременно по российским и международным стандартам
- количество однократно учитываемых параметров деятельности организации от 200 до 1000, количество формируемых таблиц баз данных – от 800 до 3000.

Система защиты информации

- парольная система разграничения доступа к данным и реализуемым функциям управления
- многоуровневая система защиты данных (средства авторизации вводимой и корректируемой информации, регистрация времени ввода и модификации данных)

Инструментальные средства адаптации и сопровождения системы

- изменение структуры и функций бизнес-процессов
- изменение информационного пространства
- изменение интерфейсов ввода, просмотра и корректировки информации
- изменение организационного и функционального наполнения рабочего места пользователя
- генератор произвольных отчетов
- генератор сложных хозяйственных операций
- генератор стандартных форм

Возможность консолидации информации

- на уровне организации – объединение информации филиалов, холдингов, дочерних компаний и т.д.
- на уровне отдельных задач – планирования, учета, контроля и т.д.
- на уровне временных периодов – для выполнения анализа финансово-экономических показателей за период, превышающий отчетный

Специальные средства анализа состояния системы в процессе эксплуатации

- анализ архитектуры баз данных
- анализ алгоритмов
- анализ статистики количества обработанной информации
- журнал выполненных операций
- список работающих станций серверов
- анализ внутрисистемной почты

Наиболее развитые корпоративные ИС (КИС) предназначены для автоматизации всех функций управления корпорацией: от научно-технической и маркетинговой подготовки ее деятельности до реализации ее продукции и услуг. В настоящее время КИС имеют в основном экономическую и производственную направленность.

2 Общие вопросы проектирования и внедрения КИС

Успешное руководство бизнесом невозможно сегодня без постоянной, объективной и всесторонней информации. Для повышения эффективности и минимизации издержек управления (временных, ресурсных и финансовых), разрабатываются и применяются корпоративные информационные системы, помогающие осуществлять контроль бюджетных процессов, рабочего времени сотрудников, выполненных ими работ, хода реализации проектов, документооборота, и других управленческих функций. Доступ к подобного рода данным может быть осуществлён как в локальной сети, так и через Интернет. С помощью эффективной корпоративной информационной системы можно значительно упростить процессы контроля и управления на предприятии любого уровня. Разработка и реализация информационных систем – одно из основных направлений деятельности вашей специальности. Этот процесс начинается с анализа деятельности предприятия и заканчивается внедрением разработанной системы. **Все *этапы* этого процесса:**

1. Проведение предпроектного обследования
2. Формулирование целей и ограничений проекта, разработка стратегии реализации проекта
3. Инжиниринг и реинжиниринг бизнес-процессов Заказчика, консалтинг в различных областях
4. Выбор платформы, разработка системы, интеграция с используемым программным обеспечением
5. Поставка оборудования и программного обеспечения
6. Пусконаладочные работы по вводу системы в эксплуатацию
7. Сопровождение созданной системы в процессе эксплуатации, работы по ее дальнейшему развитию

Так же корпоративные информационные системы сегодня являются важнейшим инструментом внедрения новых методов управления и реструктуризации предприятия**.**

В последнее время интерес к корпоративным информационным системам (КИС) постоянно растет. Если вчера КИСы привлекали внимание довольно узкого круга руководителей, то сейчас проблемы автоматизации деятельности компаний стали актуальными практически для всех. Обусловлено это не только положительной динамикой развития экономики, но и тем, что сегодня предприятия уже обладают значительным опытом использования программных продуктов различного класса.

Основная задача *проектирования и внедрения* корпоративных информационных систем, как результата системной интеграции, - комплексная деятельность по решению бизнес-задач средствами современных информационных технологий. Разработка проекта информационной системы ведется совместно с клиентом, что позволяет создать успешно работающую и удовлетворяющую все потребности заказчика корпоративную информационную систему.

Спектр бизнес-процессов, реализованных в различных КИС, может быть достаточно широк. Среди прочего это и управление продажами в различных формах, например, продажа в кредит или продажа с оплатой встречным обязательством, разнообразные бизнес-процессы, связанные с планированием, закупками, производством, хранением, персоналом, и многое-многое другое.

Информационная система может строиться с применением послойного принципа. Так, в отдельные слои можно выделить специализированное программное обеспечение (офисное, прикладное), непосредственно workflow, систему управления документами, программы поточного ввода документов, а также вспомогательное программное обеспечение для связи с внешним миром и обеспечения доступа к функционалу системы через коммуникационные средства (e-mail, Internet/intranet). Среди преимуществ такого подхода следует отметить возможность внесения изменений в отдельные программные компоненты, расположенные в одном слое, без необходимости коренных переделок на других слоях, обеспечить формальную спецификацию интерфейсов между слоями, поддерживающих независимое развитие информационных технологий и реализующих их программных средств. Причем применение открытых стандартов позволит безболезненно осуществлять переход с программных модулей одного производителя на программы другого (например, замена почтового сервера или СУД). Кроме того, послойный подход позволит повысить надежность и устойчивость к сбоям системы в целом.

2.1 Что даёт внедрение КИС?

Преимущества внедрения корпоративных информационных систем:

1. получение достоверной и оперативной информации о деятельности всех подразделений компании;
2. повышение эффективности управления компанией;
3. сокращение затрат рабочего времени на выполнение рабочих операций;
4. повышение общей результативности работы за счет более рациональной ее организации.

Самый важный вопрос. Давайте на секунду спросим себя: что даёт человеку нервная система? Конечно же, способность управлять собой, сопротивляться неблагоприятным внешним факторам и гибко реагировать на изменения окружающей среды. Если представить компанию в качестве живого организма, то КИС лучше всего подходит на роль его нервной системы, пронизывающей все органы, все частички корпоративного организма.

Повышение внутренней управляемости, гибкости и устойчивости к внешним воздействиям увеличивает эффективность компании, её конкурентоспособность, а, в конечном счёте - прибыльность. Вследствие внедрения КИС увеличиваются объёмы продаж, снижается себестоимость, уменьшаются складские запасы, сокращаются сроки выполнения заказов, улучшается взаимодействие с поставщиками. Но, несмотря на привлекательность приведённых утверждений, вопрос об окупаемости инвестиций в КИС не теряет свою актуальность. Соотношение выгоды от использования системы и ее стоимости является одним из наиболее важных факторов, оказывающих влияние на решение "покупать или не покупать". Любой инвестиционный проект, а внедрение КИС, несомненно, нужно рассматривать как инвестиционный проект, представляет собой своего рода "покупку" и, соответственно, требует оценки его стоимости и ожидаемой выгоды.

Прямую окупаемость КИС посчитать непросто, поскольку в результате внедрения оптимизируется внутренняя структура компании, снижаются трудноизмеримые транзакционные издержки. Сложно определить, например, в какой степени увеличение доходов компании явилось следствием работы КИС (читай - программной системы), а в какой - результатом настройки бизнес-процессов, то есть плодом управленческих технологий. Однако в некоторых аспектах деятельности компании оценка вполне реальна. В первую очередь это касается

логистики, где внедрение КИС приводит к оптимизации материальных потоков и к снижению потребности в оборотных средствах. Постановка на базе КИС системы финансового контроллинга приводит к снижению накладных затрат компании, ликвидации убыточных подразделений и исключению из ассортимента нерентабельных продуктов.

Совсем трудно оценить эффект от ликвидации хаоса. Для того чтобы это сделать, нужно чётко представлять масштабы хаоса, что в силу самой природы беспорядка невозможно. Действительно, можете ли Вы сказать, сколько денег Ваша компания не зарабатывает (читай - теряет) из-за перекосов в ассортименте, или, скажем, из-за срыва сроков исполнения заказов? Какие ресурсы компании оказываются выведенными из оборота вследствие "посмертного" учёта и нестыковки данных в бухгалтерии, на складе и в цехах? А как оценить объём воровства и разбазаривания ресурсов?

В настоящее время для оценки эффективности IT-проектов применяется метод инвестиционного анализа Cost Benefit Analysis **(CBA)** Метод назван так, поскольку в основе лежит оценка и сравнение выгод от осуществления проекта, с затратами на его реализацию.

Глобальная цель внедрения КИС - повышение эффективности компании. Каждая компания определяет ключевые сферы, влияющие на ее эффективность, так называемые "критические факторы успеха" (Critical Success Factor -- CSF). Повышение эффективности происходит за счет реализации задач в каждой из ключевых областей. Поэтому в основе CBA лежат именно бизнес-цели компании, определенные на этапе стратегического планирования.

Но достигнуть цели можно несколькими путями, поэтому второй краеугольный камень CBA - сравнение альтернативных вариантов. При этом одним из возможных является вариант "без КИС", т. е. рассматривается развитие во времени текущей ситуации без внесения в нее каких-либо изменений. Сравнение альтернативных вариантов производится на основании измерения приносимых ими выгод и требуемых для этого затрат. Учитываются как количественные, так и качественные показатели. Анализу качественных показателей в последнее время уделяется особое внимание. Помимо соотношения выгод и затрат, альтернативные варианты также отличаются степенью риска и факторами, которые эти риски определяют. Поэтому анализ влияния таких факторов на соотношение выгод и затрат является еще одной сферой внимания CBA. Это о методах оценки конкретного случая.

Если же говорить о статистических данных, характеризующих эффективность внедрения КИС, могу привести следующие цифры:

- Снижение транспортно-заготовительных расходов на 60%;
- Сокращение производственного цикла по заказным изделиям на 50%;
- Сокращение количества задержек с отгрузкой готовой продукции на 45%;
- Уменьшение уровня неснижаемых остатков на складах на 40%;
- Снижение производственного брака на 35%;
- Уменьшение административно-управленческих расходов на 30%;
- Сокращение производственного цикла по базовым изделиям на 30%;

- Уменьшение складских площадей на 25%;
- Увеличение оборачиваемости средств в расчётах на 30%;
- Увеличение оборачиваемости ТМЗ на 65%;
- Увеличение количества поставок точно в срок на 80%.

Эта статистика собрана на примере западных компаний, где качество управления и так достаточно высокое. Как Вы считаете, на российской почве эффект будет больше или меньше?

2.2 Принципы построения КИС

Концепция построения КИС в экономике предусматривает наличие типовых компонентов:

1. Ядро системы, обеспечивающее комплексную автоматизацию совокупности бизнес-приложений, содержит полный набор функциональных модулей для автоматизации задач управления;
2. Система автоматизации документооборота в рамках корпорации;
3. Вспомогательные инструментальные системы обработки информации (экспертные системы, системы подготовки и принятия решений и др.) на базе хранилищ данных КИС;
4. Программно-технические средства системы безопасности КИС;
5. Сервисные коммуникационные приложения (электронная почта, программное обеспечение удаленного доступа);
6. Компоненты интернет/интранет для доступа к разнородным базам данных и информационным ресурсам, сервисным услугам;
7. Офисные программы - текстовый редактор, электронные таблицы, СУБД настольного класса и др.
8. Системы специального назначения - системы автоматизированного проектирования (САПР), автоматизированные системы управления технологическими процессами (АСУТП), банковские системы и др.

Ядром каждой производственной системы являются воплощенные в ней рекомендации по управлению производством. На данный момент существует несколько сводов таких рекомендаций. Они представляют собой описание общих правил, по которым должны производиться планирование и контроль различных стадий деятельности корпорации. Далее рассмотрены некоторые из существующих технологий управления.

К основным принципам построения КИС относятся:

1. Принцип интеграции, заключающийся в том, что обрабатываемые данные вводятся в систему только один раз и затем многократно используются для решения возможно большего числа задач; принцип однократного хранения информации;
2. Принцип системности, заключающийся в обработке данных в раз личных разрезах, чтобы получить информацию, необходимую для принятия решений на всех уровнях и во всех функциональных под системах и подразделениях корпорации; внимание не только к под системам, но и к связям между ними; эволюционный аспект – все стадии эволюции продукта, в фундаменте КИС должна лежать способность к развитию;

3. Принцип комплексности, подразумевающий автоматизацию процедур преобразования данных на всех стадиях продвижения продуктов корпорации.

2.3 Этапы проектирования КИС:

1. **Анализ**

Обследование и создание моделей деятельности организации, анализ (моделей) существующих КИС, анализ моделей и формирование требований к КИС, разработка плана создания КИС.

2. **Проектирование**

Концептуальное проектирование, разработка архитектуры КИС, проектирование общей модели данных, формирование требований к приложениям.

3. **Разработка**

Разработка, прототипирование и тестирование приложений, разработка интеграционных тестов, разработка пользовательской документации.

4. **Интеграция и тестирование**

Интеграция и тестирование приложений в составе системы, оптимизация приложений и баз данных, подготовка эксплуатационной документации, тестирование системы.

5. **Внедрение**

Обучение пользователей, развертывание системы на месте эксплуатации, инсталляция баз данных, эксплуатация.

6. **Сопровождение**

Регистрация, диагностика и локализация ошибок, внесение изменений и тестирование, управление режимами работы ИС.

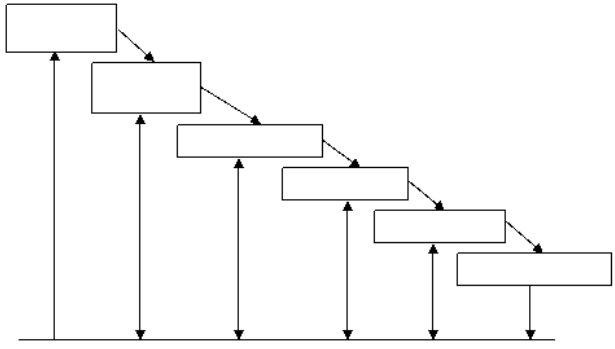
Классический жизненный цикл

Одной из старейших последовательностей шагов разработки программного обеспечения (ПО) является классический жизненный цикл (Автор Уинстон Ройс, 1970).

Чаще классический жизненный цикл называют КАСКАДНОЙ или ВОДОПАДНОЙ моделью, подчеркивая, что разработка рассматривается как последовательность этапов, причем переход на следующий иерархически нижний этап происходит только после полного завершения работ на текущем этапе и возврата к пройденным этапам не предусматривается. (см. рис. ниже)

Рисунок - Классический жизненный цикл разработки ПО.

Приведем краткое описание основных этапов. Разработка начинается на системном уровне и проходит через



- анализ,
- проектирование,
- кодирование (реализация),
- тестирование,
- сопровождение

При этом моделируются действия стандартного инженерного цикла.

Системный анализ определяет роль каждого элемента в компьютерной системе, взаимодействие элементов друг с другом. Анализ начинается с определения требований и назначения подмножества этих требований программному элементу.

На этом этапе начинается решение задачи планирования проекта ПО.

В ходе планирования проекта определяются:

- объем проектных работ,
- риск проектных работ,
- необходимые трудозатраты,
- формируются рабочие задачи,
- формируется план-график работ.

Анализ требований, относящийся к программному элементу, т.е. к ПО, уточняет и детализирует:

- функции ПО,
- характеристики ПО,
- интерфейс ПО.

Все определения документируются в спецификации анализа.

Проектирование создает представления:

- архитектуры ПО,
- модульной структуры ПО,
- алгоритмической структуры ПО,
- структуры данных,

- входного и выходного интерфейса (входных и выходных форм данных).

Кодирование (реализация) состоит в переводе результатов проектирования в текст на языке программирования.

Тестирование – это выполнение программы для выявления дефектов в функциях, логике и форме реализации программного продукта.

Сопровождение – это внесение изменений в эксплуатируемое ПО. Цели изменений:

- исправление ошибок,
- адаптация к изменениям внешней для ПО среды,
- усовершенствование ПО по требованию заказчика.

Сопровождение ПО состоит в повторном применении каждого из предшествующих шагов (этапов) жизненного цикла, т.е. системного анализа, анализа требований, проектирования и т. д., к существующей программе, но не разработке новой программы.

Каждая стадия (этап) завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Достоинствами классического жизненного цикла являются:

- получение плана и временного графика по всем этапам проекта,
- упорядочение хода разработки.

К недостаткам классического жизненного цикла относятся:

- частое отклонение реальных проектов от стандартной последовательности шагов,
- основанность цикла на точной формулировке исходных требований к ПО, тогда как реально в начале проекта требования заказчика определены лишь частично,
- доступность результатов проекта заказчику лишь в конце работы.

Макетирование (прототипирование)

На начальной стадии проекта полностью и точно сформулировать все требования к будущей модели невозможно, поскольку пользователи, как правило, не в состоянии изложить все свои требования и не могут предвидеть, как они изменятся в ходе разработки, и , кроме того, за время разработки могут произойти изменения во внешней среде, которые могут повлиять на требования к системе. Поэтому процесс создания ПО носит скорее итерационный характер, когда результаты очередной стадии разработки могут вызвать необходимость возврата к предыдущим разработкам.

Поэтому ПО создается не сразу, как в случае каскадного подхода, а постепенно с использованием макетирования (прототипирования), когда создается модель требуемого программного продукта.. *Под прототипом* понимается действующий программный компонент, реализующий отдельные функции.

Модель может принимать одну из трех форм:

- бумажный макет или макет на основе ПК (изображает или рисует человеко – машинный диалог),

- работающий макет (выполняет некоторую часть требуемых функций),
- существует программа, характеристики которой затем должны быть улучшены.

Макетирование основывается на многократном повторении итераций, в

которых участвуют заказчик и разработчик.

Поскольку часто заказчик не может определиться в своих требованиях по разрабатываемому продукту, а проектировщик сомневается в полноте и целесообразности требований заказчика, то прототипирование (макетирование) начинается со сбора и уточнения требований к создаваемому ПО.

Совместными усилиями разработчик и заказчик определяют все цели ПО, устанавливают, какие требования известны, а какие предстоит доопределить. Следующим шагом является быстрое проектирование, внимание в котором сосредотачивается на тех характеристиках ПО, которые должны быть видимы пользователю. Макет (прототип), построенный на этапе быстрого проектирования, оценивается заказчиком и используется для уточнения требований к ПО. Итерации повторяются до тех пор, пока макет не выявит все требования заказчика и не даст возможности разработчику понять, что должно быть сделано.

Достоинством макетирования является обеспечение определения полных требований к ПО.

К недостаткам макетирования относятся:

- возможность принятия заказчиком макета за продукт,
- возможность принятия разработчиком макета за продукт

Заказчик, получивший предварительную версию (макет) и удостоверившийся в ее работоспособности, может перестать видеть недостатки и нерешенные вопросы ПО и перестать соглашаться на дальнейшее усовершенствование, требуя скорейшего преобразования макета в рабочий продукт. В тоже время для экономии времени разработки макета, а также возможности показать работающий вариант, разработчик может использовать неэффективные средства. Забывая о причинах, побудивших использовать эти средства, разработчик может интегрировать неэффективный вариант в систему.

Стратегии разработки ПО

Стратегии разработки ПО можно подразделить на три группы:

1. *Линейная последовательность этапов разработки* – однократный проход (водопадная стратегия)
2. *Инкрементная стратегия*, когда сначала определяются все требования (пользовательские и системные), а затем оставшаяся часть разработки выполняется в виде последовательности версий, первая из которых реализует часть запланированных возможностей, а все последующие версии реализуют дополнительные возможности до тех пор, пока не будет получена полная система.
3. *Эволюционная стратегия*.

При этой стратегии начальный этап не содержит полного объема требования, они уточняются в ходе разработки новых последовательных версий.

Инкрементная стратегия

Инкрементная модель является классическим примером инкрементной стратегии разработки ПО, объединяя элементы последовательной водопадной модели с итерационной философией макетирования. Она представляет собой несколько поставок (инкрементов) представляющих собой последовательность анализа, проектирования, кодирования и тестирования.

Разработка первого инкремента позволяет получить базовый продукт, реализующий базовые требования, при этом многие вспомогательные требования остаются нереализованными. План следующих инкрементов предусматривает последовательную модификацию базового продукта, обеспечивающих дополнительные характеристики и функциональность.

По своей природе инкрементный процесс итеративен, но в отличие от макетирования инкрементная модель обеспечивает в конце инкрементной итерации работающий продукт.

Эволюционная стратегия разработки ПО

Эволюционную стратегию рассмотрим на примерах спиральной модели, компонентно-ориентированной модели и тяжеловесных и облегченных процессах проектирования.

Спиральная модель

Спиральная модель (автор Боэм Б, 1988 г.) опирается на лучшие свойства классического жизненного цикла и макетирования, к которым добавляется новый элемент – анализ риска, отсутствующий в этих шагах разработки.

Спиральная модель определяет планирование (определение целей, вариантов, ограничений), анализ риска (анализ вариантов и распознавание/выбор риска), конструирование (разработка продукта следующего уровня), оценивание (оценка заказчиком текущих результатов разработки).

С каждой итерацией по спирали (продвижением от центра к периферии) строятся все более полные версии ПО. В первом витке спирали определяются:

1) начальные цели, варианты и ограничения;
2) распознавание и анализ риска;
3) необходимость использования макетирования;
4) оценка заказчиком конструктивной работы и внесение предложения по модификации;
5) следующая фаза планирования и анализа риска, базируемая на предложениях заказчика.

В каждом цикле по спирали результаты анализа риска формируются в виде «продолжать, не продолжать». Если риск слишком велик, проект может быть остановлен. В большинстве случаев движение по спирали продолжается, с каждым шагом продвигая разработчиков к более общей модели системы. В каждом цикле по спирали требуется конструирование, которое может быть реализовано классическим жизненным циклом или макетированием.

К достоинствам спиральной модели относится:

1) наиболее реальное (в виде эволюции) отображение разработки программного обеспечения,

2) возможность явно учитывать риск на каждом витке эволюционной разработки,

3) включение шага системного подхода в итерационную структуру разработки,

4) использование моделирования для уменьшения риска и совершенствования программного изделия.

Недостатками спиральной модели являются:

1) повышенные требования к заказчику,

2) трудности контроля и управления временем разработки.

Компонентно-ориентированная модель

Компонентно-ориентированная модель является развитием спиральной модели и основывается на эволюционной стратегии разработки ПО. В этой модели конкретизируется содержание конструирования – оно отображает тот факт, что в современных условиях новая разработка должна основываться на повторном использовании существующих программных компонентов.

К достоинствам компонентно-ориентированной модели относится:

1) уменьшение времени разработки ПО;

2) снижение стоимости программной разработки;

3) повышение производительности разработки.

Тяжеловесные и облегченные процессы

Традиционно для упорядочения и ускорения программных разработок использовались строго упорядочивающие так называемые *тяжеловесные процессы*. В этих процессах прогнозируется весь объем предстоящих работ, поэтому они называются *прогнозирующимися процессами.* Порядок, который должен выполнять при этом человек-разработчик, чрезвычайно строг.

В последние годы появилась группа новых *облегченных процессов* разработки ПО. Их также называют *подвижными процессами*. Эти процессы привлекательны отсутствием бюрократизма, характерного для тяжеловесных (прогнозирующих) процессов.

Облегченные процессы разработки ПО воплощают разумный компромисс между строгой дисциплиной и отсутствием ее.

Подвижные процессы требуют меньшего объема документации и ориентированы на человека. Подвижные процессы учитывают особенности современного заказчика, а именно, частые изменения его требований к ПО. Подвижные процессы адаптируют изменения требований (адаптивная природа).

3 Классификация и характеристики КИС

3.1 Классификация КИС

Корпоративные информационные системы можно также разделить на два класса:

финансово-управленческие и производственные.

1. ***Финансово-управленческие системы*** включают подкласс малых интегрированных систем. Такие системы предназначены для ведения учета по одному или нескольким направлениям (бухгалтерия, сбыт, склад, кадры и т.д.)- Системами этой группы может воспользоваться практически любое предприятие.

Системы этого класса обычно универсальны, цикл их внедрения невелик, иногда можно воспользоваться «коробочным» вариантом, купив программу и самостоятельно установив ее на ПК.

Финансово-управленческие системы (особенно системы российских разработчиков) значительно более гибкие в адаптации к нуждам конкретного предприятия. Часто предлагаются «конструкторы», с помощью которых можно практически полностью перестроить исходную систему, самостоятельно или с помощью поставщика установив связи между таблицами БД или отдельными модулями.

2. ***Производственные системы*** (также называемые системами производственного управления) включают подклассы средних и крупных

интегрированных систем. Они предназначены в первую очередь для управления и планирования производственного процесса. Учетные функции, хотя и глубоко проработаны, играют вспомогательную роль, и порой невозможно выделить модуль бухгалтерского учета, так как информация в бухгалтерию поступает автоматически из других модулей.

Эти системы функционально различны: в одной может быть хорошо развит производственный модуль, в другой - финансовый. Сравнительный анализ систем такого уровня и их применимости к конкретному случаю может вылиться в значительную работу. А для внедрения системы нужна целая команда из финансовых, управленческих и технических экспертов. Производственные системы значительно более сложны в установке (цикл внедрения может занимать от 6 - 9 месяцев до полутора лет и более). Это обусловлено тем, что система покрывает потребности всего предприятия, и это требует значительных совместных усилий сотрудников предприятия и поставщиков программ.

Производственные системы часто ориентированы на одну или несколько отраслей и/или типов производства: серийное сборочное (электроника, машиностроение), мелкосерийное и опытное (авиация, тяжелое машиностроение), дискретное (металлургия, химия, упаковка), непрерывное (нефтедобыча, газодобыча).

Специализация отражается как в наборе функций системы, так и в существовании бизнес - моделей данного типа производства. Наличие встроенных моделей для

определенного типа производства отличает производственные системы друг от друга. У каждой из них есть глубоко проработанные направления и функции, разработка которых только начинается или вообще не ведется.

Производственные системы по многим параметрам значительно более жестки, чем финансово-управленческие. Основное внимание уделяется планированию и оптимальному управлению производством. Эффект от внедрения производственных систем проявляется на верхних эшелонах управления предприятием, когда становится видна вся картина его работы, включая планирование, закупки, производство, сбыт, запасы, финансовые потоки и другие аспекты.

При увеличении сложности и широты охвата функций предприятия системой возрастают требования к технической инфраструктуре и программно-технической платформе. Все производственные системы разработаны с помощью промышленных баз данных. В большинстве случаев используются технология клиент-сервер или Internet-технологии.

Для автоматизации больших предприятий в мировой практике часто используется смешанное решение из классов крупных, средних и малых интегрированных систем. Наличие электронных интерфейсов упрощает взаимодействие между системами и позволяет избежать двойного ввода данных.

Также различают виды КИС, такие как *заказные* (уникальные) и

тиражируемые КИС.

Заказные КИС

Под *заказными КИС* обычно понимают системы, создаваемые для конкретного предприятия, не имеющего аналогов и не подлежащие в дальнейшем тиражированию.

Подобные системы используются либо для автоматизации деятельности предприятий с уникальными характеристиками либо для решения крайне ограниченного круга специальных задач.

Заказные системы, как правило, либо вообще не имеют прототипов, либо использование прототипов требует значительных его изменений, имеющих качественный характер. Разработка заказной КИС характеризуется повышенным риском в плане получения требуемых результатов.

Тиражируемые (адаптируемые) КИС.

Суть проблемы адаптации тиражируемых КИС, т.е. приспособления к условиям работы на конкретном предприятии в том, что в конечном итоге каждая КИС уникальна, но вместе с тем ей присущи и общие, типовые свойства. Требования к адаптации и сложность их реализации существенно зависят от проблемной области, масштабов системы. Даже первые программы, решавшие отдельные задачи автоматизации, создавались с учетом необходимости их настройки по параметрам.

Разработка КИС на предприятии может вестись как **“от нуля”, так и на основе референционной модели.**

Референционная модель представляет собой описание облика системы, функций, организованных структур и процессов, типовых в каком-то смысле (отрасль, тип производства и т.д.).

В ней отражаются типовые особенности, присущие определенному классу предприятий. Ряд компаний – производителей адаптируемых (тиражируемых) КИС совместно с крупными консалтинговыми фирмами в течение ряда лет ведет разработку референционных моделей для предприятий автомобильной, авиационной и других отраслей.

Адаптации и референционные модели входят в состав многих систем класса

MRP II / ERP, что позволяет значительно сократить сроки их внедрения на предприятия.

Референционная модель в начале работы по автоматизации предприятия может представлять собой описание существующей системы (как есть) и служит точкой отсчета, с которой начинаются работы по совершенствованию КИС.

Используется также следующая классификация. КИС делятся на три (иногда четыре) большие группы:

1) простые (“коробочные”);
2) среднего класса;
3) высшего класса

Простые (“коробочные”) КИС реализуют небольшое число бизнес-процессов организации. Типичным примером систем подобного типа являются бухгалтерские, складские и небольшие торговые системы наиболее широко представленные на российском рынке. Например, системы таких фирм как 1С, Инфин и т.д.

Отличительной особенностью таких продуктов является относительная легкость в усвоении, что в сочетании с низкой ценой, соответствием российскому законодательству и возможностью выбрать систему “на свой вкус” приносит им широкую популярность. *Системы среднего класса* отличаются большей глубиной и широтой охвата функций. Данные системы предлагают российские и зарубежные компании. Как правило, это системы, которые позволяют вести учет деятельности предприятия по многим или нескольким направлениям:

- финансы;
- логистика;
- персонал;
- сбыт.

Они нуждаются в настройке, которую в большинстве случаев осуществляют специалисты фирмы-разработчика, а также в обучении пользователей.

Эти системы больше всего подходят для средних и некоторых крупных предприятий в силу своей функциональности и более высокой, по сравнению с первым классом,

стоимости. Из российских систем данного класса можно выделить, например, продукцию компаний Галактика, ТБ.СОФТ

К *высшему классу* относятся системы, которые отличаются высоким уровнем детализации хозяйственной деятельности предприятия. Современные версии таких систем обеспечивают планирование и управление всеми ресурсами организации (ERP-системы).

Как правило, при внедрении таких систем производится моделирование существующих на предприятии бизнес-процессов и настройка параметров системы под требования бизнеса.

Однако значительная избыточность и большое количество настраиваемых параметров системы обуславливают длительный срок ее внедрения, и также необходимость наличия на предприятии специального подразделения или группы специалистов, которые будут осуществлять перенастройку системы в соответствии с изменениями бизнес-процессов.

На российском рынке имеется большой выбор КИС высшего класса, и их число растет. Признанными мировыми лидерами являются, например, R/3 фирмы SAP, Oracle Application компании Oracle.

3.2 Классификация автоматизированных систем

Рассмотрим классификацию автоматизированных систем (АС):

- **Классификация систем по масштабу применения**

1. локальные (в рамках одного рабочего места);
2. местные (в пределах одной организации);
3. территориальные (в пределах некоторой административной территории);
4. отраслевые.

- **Классификация по режиму использования**
  1. системы пакетной обработки (первые варианты организационных АСУ, системы информационного обслуживания, учебные системы);
  2. запросно-ответные системы (АИС продажи билетов, информационно-поисковые системы, библиотечные системы);
  3. диалоговые системы (САПР, АСНИ, обучающие системы);
  4. системы реального времени (управление технологическими процессами, подвижными объектами, роботами-манипуляторами, испытательными стендами и другие).

АИС - автоматизированная информационная система

АИС предназначены для накопления, хранения, актуализации и обработки систематизированной информации в каких-то предметных областях и предоставления требуемой информации по запросам пользователей. АИС может функционировать самостоятельно либо являться компонентой более сложной системы (например, АСУ или САПР).

По характеру информационных ресурсов АИС делятся на два вида: фактографические и документальные (хотя возможны и комбинированные АИС). Фактографические системы характеризуются тем, что они оперируют фактическими сведениями, представленными в виде специальным образом организованных совокупностей формализованных записей данных. Эти записи образуют базу данных системы. Существует специальный класс программных средств для создания и обеспечения функционирования таких фактографических баз данных – системы управления базами данных.

Документальные АИС оперируют неформализованными документами произвольной структуры с использованием естественного языка. Среди таких систем наиболее распространенными являются информационно-поисковые системы, которые включают программные средства для организации ввода и хранения информации, поддержки общения с пользователем, обработки запросов и поисковый массив документов. Этот массив часто содержит не тексты документов, а только их библиографическое описание, иногда рефераты или аннотации. Для работы системы используются поисковые образы документов (ПОД) – формализованные объекты, отражающие содержание документов. Запрос преобразуется системой в поисковый образ запроса (ПОЗ), который затем сопоставляется с ПОД по критерию смыслового соответствия. Вариантом информационно-поисковых систем являются библиотечные системы, с помощью которых

создаются электронные каталоги библиотек.

Активно развивающейся в настоящее время разновидностью АИС являются географические информационные системы (ГИС). Геоинформационная система предназначена для обработки пространственно-временных данных, основой интеграции которых служит географическая информация. ГИС позволяет упорядочивать информацию о данной местности или городе как комплекте карт. В каждой карте представлена информация об одной характеристике местности. Каждая из этих отдельных карт называется слоем. Самый нижний слой представляет сетку координатной системы, в которой все карты зарегистрированы. Это позволяет анализировать и сравнивать информацию во всех слоях или в некоторой их комбинации.

Возможность разделить информацию на слои и дальнейшее их комбинирование определяет большой потенциал ГИС как научного инструментария и средства для принятия решения, так как обеспечивается возможность интеграции самой разной информации об окружающей среде и обеспечивается аналитический инструментарий использования этих данных. В ГИС могут быть десятки и сотни слоев карт, которые выстроены в определённом порядке и показывают информацию о транспортной сети, гидрографии, характеристиках населения, экономической активности, политической юрисдикции и других характеристиках природной и социальной сред.

Такая система может быть полезной в широком диапазоне ситуаций, включающих

анализ и управление природными ресурсами, планирование землепользования, инфраструктуры и градостроительства, управление чрезвычайными ситуациями, анализ местоположения и так далее.

Как уже отмечалось во введении, в настоящее время термин информационная система (подразумевается автоматизированная система) часто используют в более широком смысле, замещая им в частности и термин АСУ. При этом под информационной системой понимается любая АС, используемая как средство сбора, накопления, хранения, обработки, передачи и представления информации в целях сопровождения и поддержки какого-либо вида профессиональной деятельности.

САПР - система автоматизированного проектирования

САПР предназначены для проектирования определенного вида изделий или процессов. Они используются для подготовки и обработки проектных данных, выбора рациональных вариантов технических решений, выполнения расчетных работ и подготовки проектной документации (в частности, чертежей). В процессе функционирования системы могут использоваться накапливаемые в ней библиотеки стандартов, нормативов, типовых элементов и модулей, а также оптимизационные процедуры.

Результатом работы САПР является соответствующий стандартам и нормативам комплект проектной документации, в котором зафиксированы проектные решения по

созданию нового или модернизации существующего технического объекта. Наиболее широко такие системы используются в электронике, машиностроении, строительстве.

АСНИ - автоматизированная система научных исследований

В настоящее время эти системы как правило, используются для развития научных исследований в наиболее сложных областях физики, химии, механики и других. В первую очередь - это системы для измерения, регистрации, накопления и обработки опытных данных, получаемых при проведении экспериментальных исследований, а также для управления ходом эксперимента, регистрирующей аппаратурой и так далее. Во многих случаях для таких систем важной является функция планирования эксперимента; целью такого планирования является уменьшение затрат ресурсов и времени на получение необходимого результата.

Кроме того, желательным свойством АСНИ является возможность создания и хранения банков данных первичных результатов экспериментальных исследований (особенно, если это дорогостоящие и трудно повторяемые исследования). Впоследствии могут появиться более совершенные методы их обработки, которые позволят получить новую информацию из старого экспериментального материала.

Как разновидность задачи автоматизации эксперимента можно рассматривать задачу автоматизации испытаний какого-либо технического объекта. Отличие состоит в том, что управляющие воздействия, влияющие на условия эксперимента, направлены на

создание наихудших условий функционирования управляемого объекта, не исключая в случае необходимости и аварийных ситуаций.

Второе направление - это компьютерная реализация сложных математических моделей и проведение на этой основе вычислительных экспериментов, дополняющих, или даже заменяющих эксперименты с реальными объектами или процессами в тех случаях, когда проведение натурных исследований дорого или вообще невозможно. Технологическая схема вычислительного эксперимента состоит из нескольких циклически повторяемых этапов: построение математической модели, разработка алгоритма решения, программная реализация алгоритма, проведение расчетов и анализ результатов. Вычислительный эксперимент представляет собой новую методологию научных исследований, соединяющую характерные черты традиционных теоретических и экспериментальных методов.

Системы используются в электронике, машиностроении, строительстве.

АСУ - автоматизированная система управления.

Как уже выше было отмечено, АСУ предназначена для автоматизированной обработки информации и частичной подготовки управленческих решений с целью увеличения эффективности деятельности специалистов и руководителей за счет повышения уровня оперативности и обоснованности принимаемых решений.

Различают два основных типа таких систем: системы управления технологическими

процессами (АСУ ТП) и системы организационного управления (АСОУ). Их главные отличия заключаются в характере объекта управления (в первом случае – это технические объекты: машины, аппараты, устройства, во втором – объекты экономической или социальной природы, то есть, в конечном счете коллективы людей) и, как следствие, в формах передачи информации (сигналы различной физической природы и документы соответственно).

Следует отметить, что наряду с автоматизированными существуют и **системы автоматического управления (САУ).** Такие системы после наладки могут некоторое время функционировать без участия человека. САУ применяются только для управления техническими объектами или отдельными технологическими процессами. Системы же организационного управления, как следует из их описания, не могут в принципе быть полностью автоматическими. Люди в таких системах осуществляют постановку и корректировку целей и критериев управления, структурную адаптацию системы в случае необходимости, выбор окончательного решения и придание ему юридической силы.

Как правило, АСОУ создаются для решения комплекса взаимосвязанных основных задач управления производственно-хозяйственной деятельностью организаций (предприятий) или их основных структурных подразделений. Для крупных систем АСОУ могут иметь иерархический характер, включать в свой состав в качестве отдельных подсистем АСУ ТП , АС ОДУ (автоматизированная система оперативно-диспетчерского

управления), автоматизированные системы управления запасами, оперативно-календарного и объемно-календарного планирования и АСУП (автоматизированная система управления производством на уровне крупного цеха или отдельного завода в составе комбината).

Самостоятельное значение имеют автоматизированные системы диспетчерского управления (АСДУ), предназначенные для управления сложными человеко-машинными системами в реальном масштабе времени. К ним относятся системы диспетчерского управления в энергосистемах, на железнодорожном и воздушном транспорте, в химическом производстве и другие. В системах диспетчерского управления (и некоторых других типах АСУ) используются подсистемы автоматизированного контроля оборудования. Задачами этой подсистемы является измерение и фиксация значений параметров, характеризующих состояние контролируемого оборудования, а сравнение этих значений с заданными границами и информирование об отклонениях.

Отдельный класс АСУ составляют системы управления подвижными объектами, такими как поезда, суда, самолеты, космические аппараты и АС управления системами вооружения.

3.3 Характеристики КИС

Наиболее значимыми характеристиками КИС являются:

1. Архитектура информационной системы - состав элементов и их

взаимодействие;

2. Сетевые технологии, их масштабы и топология сети;

3. Функциональная структура управления, реализованная в информационной системе (состав подсистем, комплексов задач);

4. Организационная форма хранения информации (централизованная или распределенная база данных);

5. Пропускная способность системы - скорость обработки транзакций;

6. Объем информационного хранилища данных;

7. Системы документов и документооборот;

8. Количество пользователей КИС;

9. Пользовательский интерфейс и его возможности;

10. Типовые информационные технологии процессов сбора, передачи, обработки, хранения, извлечения, распространения информации.

11. Обеспечение полного цикла управления в масштабах корпорации: нормирование, планирование, учет, анализ, регулирование на основе обратной связи в условиях информационной и функциональной интеграции;

12. Территориальная распределенность и значительные масштабы системы и объекта управления;

13. Неоднородность составляющих технического и программного обеспечения

структурных компонентов системы управления;

14. Единое информационное пространство для выработки управленческих решений, объединяющее управление финансами, персоналом, снабжением, сбытом и процесс управления производством;

15. Функционирование в неоднородной вычислительной среде на разных вычислительных платформах;

16. Реализация управления в реальном масштабе времени;

Высокая надежность, безопасность, открытость и масштабируемость информационных компонентов.

4 Архитектура КИС

Опыт последних лет разработки ПО показывает, что архитектура информационной системы должна выбираться с учетом нужд бизнеса, а не личных пристрастий разработчиков. Далее рассматриваются существующие клиент-серверные архитектуры построения информационных систем.

Не секрет, что правильная и четкая организация информационных бизнес-решений является слагающим фактором успеха любой компании. Особенно важным этот фактор является для предприятий среднего и малого бизнеса, которым необходима система, которая способна предоставить весь объем бизнес-логики для решения задач компании. В то же время, такие системы для компаний со средним и малым масштабом сетей часто попадают под критерий “цена - качество”, то есть должны обладать максимальной производительностью и надежностью при доступной цене.

Первоначально системы такого уровня базировались на классической двухуровневой клиент-серверной архитектуре (Two-tier architecture) (рис. 3.1).

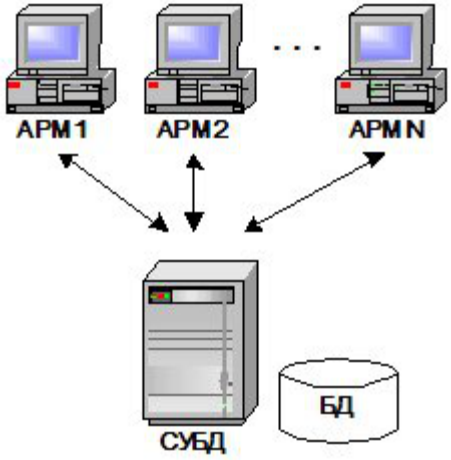


Рисунок 3.1 - Двухуровневая клиент-серверная архитектура

Данная клиент-серверная архитектура характеризуется наличием двух взаимодействующих самостоятельных модулей - автоматизированного рабочего места (АРМа) и сервера базы данных, в качестве которого может выступать Microsoft SQL Server, Oracle, Sybase и другие. Сервер БД отвечает за хранение, управление и целостность данных, а также обеспечивает возможность одновременного доступа нескольких пользователей. Клиентская часть представлена так называемым “толстым” клиентом, то есть приложением (АРМ) на котором сконцентрированы основные правила работы системы и расположен пользовательский интерфейс программы. При всей простоте построения такой архитектуры, она обладает множеством недостатков, наиболее существенные из которых - это высокие требования к сетевым ресурсам и пропускной

способности сети компании, а также сложность обновления программного обеспечения из-за “размазанной” бизнес-логики между АРМом и сервером БД. Кроме того, при большом количестве АРМов возрастают требования к аппаратному обеспечению сервера БД, а это, как известно, самый дорогостоящий узел в любой информационной системе.

Как видим, минусов у такой архитектуры достаточно, а решение тривиально - нужно отделить бизнес-логику от клиентской части и СУБД, выделив ее в отдельный слой. Так и поступили разработчики и следующим шагом развития клиент-серверной архитектуры стало внедрение среднего уровня, реализующего задачи бизнес-логики и управления механизмами доступа к БД (рис. 3.2).

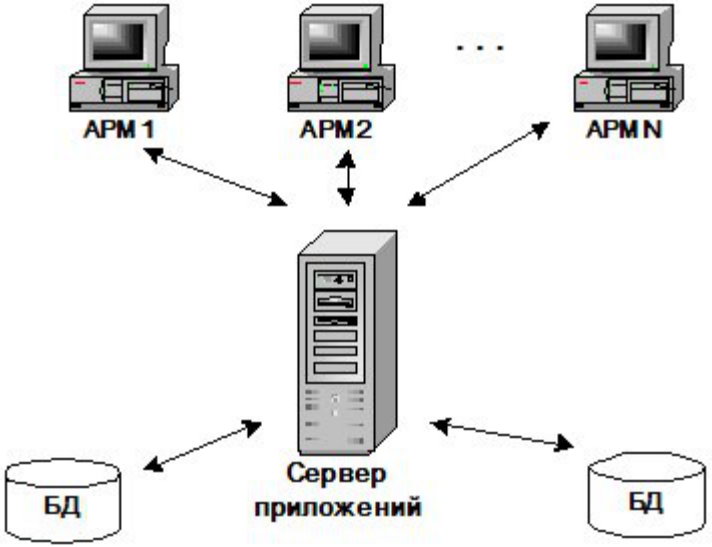


Рисунок 3.2 - Трехуровневая клиент-серверная архитектура (Three-tier architecture)

Плюсы данной архитектуры очевидны. Благодаря концентрации бизнес-логики на сервере приложений, стало возможно подключать различные БД. Теперь, сервер базы данных освобожден от задач распараллеливания работы между различными пользователями, что существенно снижает его аппаратные требования. Также снизились требования к клиентским машинам за счет выполнения ресурсоемких операций сервером приложений и решающих теперь только задачи визуализации данных. Именно поэтому такую схему построения информационных систем часто называют архитектурой “тонкого” клиента.

Но, тем не менее, узким местом, как и в двухуровневой клиент-серверной архитектуре, остаются повышенные требования к пропускной способности сети, что в свою очередь накладывает жесткие ограничения на использование таких систем в сетях с неустойчивой связью и малой пропускной способностью (Internet, GPRS, мобильная связь).

Существует еще один важный момент использования систем, построенных на такой архитектуре. Самый верхний уровень (АРМы), в целом обладающий огромной вычислительной мощностью, на самом деле простаивает, занимаясь лишь выводом информации на экран пользователя. Так почему бы не использовать этот потенциал в работе всей системы? Рассмотрим следующую архитектуру(Рис. 3.3) которая позволяет

решить эту задачу.

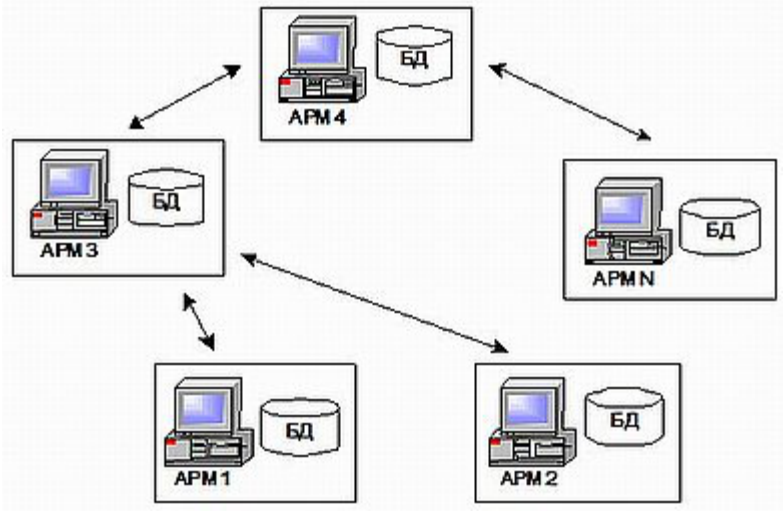


Рисунок 3.3 - Распределенная архитектура системы

Еще два-три года назад реализация такой архитектуры системы для среднего и малого бизнеса была бы не возможна из-за отсутствия соответствующих недорогих аппаратных средств. Сегодня хороший ноутбук обладает мощностью, которой несколько лет назад обладал сервер крупной корпорации, и позволял рассчитывать множество важных и судьбоносных отчетов для всех сотрудников этой корпорации.

Более 95 % данных, используемых в управлении предприятием, могут быть размещены на одном персональном компьютере, обеспечив возможность его независимой работы. Поток исправлений и дополнений, создаваемый на этом компьютере, ничтожен

по сравнению с объемом данных, используемых при этом. Поэтому если хранить непрерывно используемые данные на самих компьютерах, и организовать обмен между ними исправлениями и дополнениями к хранящимся данным, то суммарный передаваемый трафик резко снизиться. Это позволяет понизить требования к каналам связи между компьютерами и чаще использовать асинхронную связь, и благодаря этому создавать надежно функционирующие распределенные информационные системы, использующие для связи отдельных элементов неустойчивую связь типа Интернета, мобильную связь, коммерческие спутниковые каналы. А минимизация трафика между элементами сделает вполне доступной стоимость эксплуатации такой связи. Конечно, реализация такой системы не элементарна, и требует решения ряда проблем, одна из которых своевременная синхронизация данных.

Каждый АРМ независим, содержит только ту информацию, с которой должен работать, а актуальность данных во всей системе обеспечивается благодаря непрерывному обмену сообщениями с другими АРМами. Обмен сообщениями между АРМами может быть реализован различными способами, от отправки данных по электронной почте до передачи данных по сетям.

Еще одним из преимуществ такой схемы эксплуатации и архитектуры системы, является обеспечение возможности персональной ответственности за сохранность данных. Так как данные, доступные на конкретном рабочем месте, находятся только на

этом компьютере, при использовании средств шифрования и личных аппаратных ключей исключается доступ к данным посторонних, в том числе и IT администраторов.

Такая архитектура системы также позволяет организовать распределенные вычисления между клиентскими машинами. Например, расчет какой-либо задачи, требующей больших вычислений, можно распределить между соседними АРМами благодаря тому, что они, как правило, обладают одной информацией в своих БД и, таким образом, добиться максимальной производительности системы.

Таким образом, предложенная модель построения распределенных систем вполне способна решить и реализовать функции современного программного обеспечения для предприятий среднего и малого бизнеса. Построенные на основе данной архитектуры системы будут обладать надежностью, безопасностью информации и высокой скоростью вычислений, что от них в первую очередь и требуется.

5 Требования, предъявляемые к КИС

КИС должны отвечать целому набору обязательных требований:

1. Среди них, в первую очередь, стоит отметить использование архитектуры клиент-сервер с возможностью применения большинства промышленных СУБД
2. Поддержку распределенной обработки информации
3. Модульный принцип построения из оперативно-независимых функциональных блоков с расширением за счет открытых стандартов (API, COM+, CORBA и другие)
4. Обеспечивать поддержку технологий Internet/intranet.
5. **Гибкость**

Гибкость, способность к адаптации и дальнейшему развитию подразумевают возможность приспособления информационной системы к новым условиям, новым потребностям предприятия. Выполнение этих условий возможно, если на этапе разработки информационной системы использовались общепринятые средства и методы документирования, так что по прошествии определенного времени сохранится возможность разобраться в структуре системы и внести в нее соответствующие изменения, даже если все разработчики или их часть по каким-либо причинам не смогут продолжить работу.

Следует иметь в виду, что психологически легче разобраться в собственных разработках, пусть даже созданных давно, чем в чужих решениях, не всегда на первый

взгляд логичных. Поэтому рекомендуется фазу сопровождения системы доверять лицам, которые ее проектировали.

Любая информационная система рано или поздно морально устареет, и станет вопрос о ее модернизации или полной замене. Разработчики информационных систем, как правило, не являются специалистами в прикладной области, для которой разрабатывается система. Участие в модернизации или создании новой системы той же группы проектировщиков существенно сократит сроки модернизации.

Вместе с тем возникает риск применения устаревших решений при модернизации системы. Рекомендация в таком случае одна — внимательнее относиться к подбору разработчиков информационных систем.

6. Надежность

Надежность информационной системы подразумевает ее функционирование без искажения информации, потери данных по «техническим причинам». Требование надежности обеспечивается созданием резервных копий хранимой информации, выполнения операций протоколирования, поддержанием качества каналов связи' и физических носителей информации, использованием современных программных и аппаратных средств. Сюда же следует отнести защиту от случайных потерь информации в силу недостаточной квалификации персонала.

7. Эффективность

Система является эффективной, если с учетом выделенных ей ресурсов она позволяет решать возложенные на нее задачи в минимальные сроки.

В любом случае оценка эффективности будет производиться заказчиком, исходя из вложенных в разработку средств и соответствия представленной информационной системы его ожиданиям.

Негативной оценки эффективности информационной системы со стороны заказчика можно избежать, если представители заказчика будут привлекаться к проектированию системы на всех его стадиях. Такой подход позволяет многим конечным пользователям уже на этапе проектирования адаптироваться к изменениям условий работы, которые иначе были бы приняты враждебно.

Активное сотрудничество с заказчиком с ранних этапов проектирования позволяет уточнить потребности заказчика. Часто встречается ситуация, когда заказчик чего-то хочет, но сам не знает чего именно. Чем раньше будут учтены дополнения заказчика, тем с меньшими затратами и в более короткие сроки система будет создана.

Кроме того, заказчик, не являясь специалистом в области разработки информационных систем, может не знать о новых информационных технологиях. Контакты с заказчиком во время разработки для него информационной системы могут подтолкнуть заказчика к модернизации его аппаратных средств, применению новых методов ведения бизнеса, что отвечает потребностям как заказчика, так и

проектировщика. Заказчик получает рост эффективности своего предприятия, проектировщик — расширение возможностей, применяемых при проектировании информационной системы.

Эффективность системы обеспечивается оптимизацией данных и методов их обработки, применением оригинальных разработок, идей, методов проектирования (в частности, спиральной модели проектирования информационной системы, о которой речь пойдет в следующих главах).

Не следует забывать и о том, что работать с системой придется обычным людям, являющимся специалистами в своей предметной области, но зачастую обладающим весьма средними навыками в работе с компьютерами. Интерфейс информационных систем должен быть им интуитивно понятен. В свою очередь, разработчик-программист должен понимать характер выполняемых конечным пользователем операций. Рекомендациями в этом случае могут служить повышение эффективности управления разработкой информационных систем, улучшение информированности разработчиков о предметной области.

Имеет смысл еще до сдачи информационной системы в эксплуатацию предоставить разработчикам возможность попробовать себя в роли конечных пользователей. Встречались случаи, когда такой подход приводил к отказу от использования на рабочем месте оператора манипулятора типа «мышь», что, в свою очередь, приводило к

многократному повышению производительности оператора.

8. Безопасность

Под безопасностью, прежде всего, подразумевается свойство системы, в силу которого посторонние лица не имеют доступа к информационным ресурсам организации, кроме тех, которые для них предназначены, что достигается с помощью различных методов контроля и разграничения доступа к информационным ресурсам.

Защита информации от постороннего доступа обеспечивается управлением доступом к ресурсам системы, использованием современных программных средств защиты информации. В крупных организациях целесообразно создавать подразделения, основным направлением деятельности которых было бы обеспечение информационной безопасности, в менее крупных организациях назначать сотрудника, ответственного за данный участок работы.

Система, не отвечающая требованиям безопасности, может причинить ущерб интересам заказчика, прежде всего имущественным.

В этой связи следует отметить, что согласно действующему в России законодательству ответственность за вред, причиненный ненадлежащим качеством работ или услуг, несет исполнитель, то есть в нашем случае разработчик информационной системы. Поэтому ненадлежащее обеспечение безопасности информационной системы заказчика в худшем случае обернется для исполнителя судебным преследованием, в

лучшем — потерей клиента и утратой деловой репутации.

Помимо злого умысла, при обеспечении безопасности информационных систем приходится сталкиваться еще с несколькими факторами. В частности, современные информационные системы являются достаточно сложными программными продуктами. При их проектировании с высокой вероятностью возможны ошибки, вызванные большим объемом программного кода, несовершенством компиляторов, человеческим фактором, несовместимостью с используемыми программами сторонних разработчиков в случае модификации этих программ и т. п. Поэтому за фазой разработки информационной системы неизбежно следует фаза ее сопровождения в процессе эксплуатации, в которой происходит выявление скрытых ошибок и их исправление.

Например, при проектировании информационной системы курс доллара США в одной из процедур разработчики обозначили константой. На момент ввода в эксплуатацию этой системы курс доллара был стабилен, поэтому ошибка никак себя не проявляла, а была выявлена только через некоторое время в период роста курса.

Требование безопасности обеспечивается современными средствами разработки информационных систем, современной аппаратурой, методами защиты информации, применением паролей и протоколированием, постоянным мониторингом состояния безопасности операционных систем и средств их защиты.

И наконец, самый важный фактор, влияющий на процесс разработки, — знания и опыт

коллектива разработчиков информационных систем.

6 Выбор аппаратно-программной платформы КИС

Выбор аппаратной платформы и конфигурации системы представляет собой чрезвычайно сложную задачу. Это связано, в частности, с характером прикладных систем, который в значительной степени может определять рабочую нагрузку вычислительного комплекса в целом. Однако часто оказывается просто трудно с достаточной точностью предсказать саму нагрузку, особенно в случае, если система должна обслуживать несколько групп разнородных по своим потребностям пользователей. Например, иногда даже бессмысленно говорить, что для каждых N пользователей необходимо в конфигурации сервера иметь один процессор, поскольку для некоторых прикладных систем, в частности, для систем из области механических и электронных САПР, может потребоваться 2-4 процессора для обеспечения запросов одного пользователя. С другой стороны, даже одного процессора может вполне хватить для поддержки 15-40 пользователей, работающих с прикладным пакетом Oracle*Financial. Другие прикладные системы могут оказаться еще менее требовательными. Но следует помнить, что даже если рабочую нагрузку удается описать с достаточной точностью, обычно скорее можно только выяснить, какая конфигурация не справится с данной нагрузкой, чем с уверенностью сказать, что данная конфигурация системы будет обрабатывать заданную нагрузку, если только отсутствует определенный опыт работы с приложением.

Обычно рабочая нагрузка существенно определяется "типом использования"

системы. Например, можно выделить серверы NFS, серверы управления базами данных и системы, работающие в режиме разделения времени. Эти категории систем перечислены в порядке увеличения их сложности. Как правило серверы СУБД значительно более сложны, чем серверы NFS, а серверы разделения времени, особенно обслуживающие различные категории пользователей, являются наиболее сложными для оценки. К счастью, существует ряд упрощающих факторов. Во-первых, как правило нагрузка на систему в среднем сглаживается особенно при наличии большого коллектива пользователей (хотя почти всегда имеют место предсказуемые пики). Например, известно, что нагрузка на систему достигает пиковых значений через 1-1.5 часа после начала рабочего дня или окончания обеденного перерыва и резко падает во время обеденного перерыва. С большой вероятностью нагрузка будет нарастать к концу месяца, квартала или года.

Во-вторых, универсальный характер большинства наиболее сложных для оценки систем - систем разделения времени, предполагает и большое разнообразие, выполняемых на них приложений, которые в свою очередь как правило стараются загрузить различные части системы. Далеко не все приложения интенсивно используют процессорные ресурсы, и не все из них связаны с интенсивным вводом/выводом. Поэтому смесь таких приложений на одной системе может обеспечить достаточно равномерную загрузку всех ресурсов. Естественно неправильно подобранная смесь может дать совсем

противоположенный эффект.

Все, кто сталкивается с задачей выбора конфигурации системы, должны начинать с определения ответов на два главных вопроса: какой сервис должен обеспечиваться системой и какой уровень сервиса может обеспечить данная конфигурация. Имея набор целевых показателей производительности конечного пользователя и стоимостных ограничений, необходимо спрогнозировать возможности определенного набора компонентов, которые включаются в конфигурацию системы. Любой, кто попробовал это сделать, знает, что подобная оценка сложна и связана с неточностью. Почему оценка конфигурации системы так сложна? Некоторое из причин перечислены ниже:

- Подобная оценка прогнозирует будущее: предполагаемую комбинацию устройств, будущее использование программного обеспечения, будущих пользователей.
- Сами конфигурации аппаратных и программных средств сложны, связаны с определением множества разнородных по своей сути компонентов системы, в результате чего сложность быстро увеличивается. Несколько лет назад существовала только одна вычислительная парадигма: мейнфрейм с терминалами. В настоящее время по выбору пользователя могут использоваться несколько вычислительных парадигм с широким разнообразием возможных конфигураций системы для каждой из них. Каждое новое поколение аппаратных и программных средств обеспечивает настолько больше возможностей, чем их предшественники, что относительно новые

представления об их работе постоянно разрушаются.

- Скорость технологических усовершенствований во всех направлениях разработки компьютерной техники (аппаратных средствах, функциональной организации систем, операционных системах, ПО СУБД, ПО "среднего" слоя (middleware) уже очень высокая и постоянно растет. Ко времени, когда какое-либо изделие широко используется и хорошо изучено, оно часто рассматривается уже как устаревшее.
- Доступная потребителю информация о самих системах, операционных системах, программном обеспечении инфраструктуры (СУБД и мониторы обработки транзакций) как правило носит очень общий характер. Структура аппаратных средств, на базе которых работают программные системы, стала настолько сложной, что эксперты в одной области редко являются таковыми в другой.
- Информация о реальном использовании систем редко является точной. Более того, пользователи всегда находят новые способы использования вычислительных систем как только становятся доступными новые возможности.

При стольких неопределенностях просто удивительно, что многие конфигурации систем работают достаточно хорошо. Оценка конфигурации все еще остается некоторым видом искусства, но к ней можно подойти с научных позиций. Намного проще решить, что определенная конфигурация не сможет обрабатывать определенные виды нагрузки, чем определить с уверенностью, что нагрузка может обрабатываться внутри

определенных ограничений производительности. Более того, реальное использование систем показывает, что имеет место тенденция заполнения всех доступных ресурсов. Как следствие, системы, даже имеющие некоторые избыточные ресурсы, со временем не будут воспринимать дополнительную нагрузку.

Для выполнения анализа конфигурации, система (под которой понимается весь комплекс компьютеров, периферийных устройств, сетей и программного обеспечения) должна рассматриваться как ряд соединенных друг с другом компонентов. Например, сети состоят из клиентов, серверов и сетевой инфраструктуры. Сетевая инфраструктура включает среду (часто нескольких типов) вместе с мостами, маршрутизаторами и системой сетевого управления, поддерживающей ее работу. В состав клиентских систем и серверов входят центральные процессоры, иерархия памяти, шин, периферийных устройств и ПО. Ограничения производительности некоторой конфигурации по любому направлению (например, в части организации дискового ввода/вывода) обычно могут быть предсказаны исходя из анализа наиболее слабых компонентов.

Поскольку современные комплексы почти всегда включают несколько работающих совместно систем, точная оценка полной конфигурации требует ее рассмотрения как на макроскопическом уровне (уровне сети), так и на микроскопическом уровне (уровне компонент или подсистем).

Эта же методология может быть использована для настройки системы после ее инсталляции: настройка системы и сети выполняются как правило после предварительной оценки и анализа узких мест. Более точно, настройка конфигурации представляет собой процесс определения наиболее слабых компонентов в системе и устранения этих узких мест.

Следует отметить, что выбор той или иной аппаратной платформы и конфигурации определяется и рядом общих требований, которые предъявляются к **характеристикам современных вычислительных систем. К ним относятся:**

- отношение стоимость/производительность
- надежность и отказоустойчивость
- масштабируемость
- совместимость и мобильность программного обеспечения.

Отношение стоимость/производительность. Появление любого нового направления в вычислительной технике определяется требованиями компьютерного рынка. Поэтому у разработчиков компьютеров нет одной единственной цели. Большая универсальная вычислительная машина (мейнфрейм) или суперкомпьютер стоят дорого. Для достижения поставленных целей при проектировании высокопроизводительных конструкций приходится игнорировать стоимостные характеристики. Суперкомпьютеры фирмы Cray Research и высокопроизводительные мейнфреймы компании IBM относятся

именно к этой категории компьютеров. Другим крайним примером может служить низкостоимостная конструкция, где производительность принесена в жертву для достижения низкой стоимости. К этому направлению относятся персональные компьютеры различных клонов IBM PC. Между этими двумя крайними направлениями находятся конструкции, основанные на отношении стоимость/производительность, в которых разработчики находят баланс между стоимостными параметрами и производительностью. Типичными примерами такого рода компьютеров являются миникомпьютеры и рабочие станции.

Для сравнения различных компьютеров между собой обычно используются стандартные методики измерения производительности. Эти методики позволяют разработчикам и пользователям использовать полученные в результате испытаний количественные показатели для оценки тех или иных технических решений, и в конце концов именно производительность и стоимость дают пользователю рациональную основу для решения вопроса, какой компьютер выбрать.

Надежность и отказоустойчивость. Важнейшей характеристикой вычислительных систем является надежность. Повышение надежности основано на принципе предотвращения неисправностей путем снижения интенсивности отказов и сбоев за счет применения электронных схем и компонентов с высокой и сверхвысокой степенью интеграции, снижения уровня помех, облегченных режимов работы схем, обеспечение

тепловых режимов их работы, а также за счет совершенствования методов сборки аппаратуры.

Отказоустойчивость - это такое свойство вычислительной системы, которое обеспечивает ей, как логической машине, возможность продолжения действий, заданных программой, после возникновения неисправностей. Введение отказоустойчивости требует избыточного аппаратного и программного обеспечения. Направления, связанные с предотвращением неисправностей и с отказоустойчивостью, - основные в проблеме надежности. Концепции параллельности и отказоустойчивости вычислительных систем естественным образом связаны между собой, поскольку в обоих случаях требуются дополнительные функциональные компоненты. Поэтому, собственно, на параллельных вычислительных системах достигается как наиболее высокая производительность, так и, во многих случаях, очень высокая надежность. Имеющиеся ресурсы избыточности в параллельных системах могут гибко использоваться как для повышения производительности, так и для повышения надежности. Структура многопроцессорных и многомашинных систем приспособлена к автоматической реконфигурации и обеспечивает возможность продолжения работы системы после возникновения неисправностей.

Следует помнить, что понятие надежности включает не только аппаратные средства, но и программное обеспечение. Главной целью повышения надежности систем является

целостность хранимых в них данных.

Масштабируемость представляет собой возможность наращивания числа и мощности процессоров, объемов оперативной и внешней памяти и других ресурсов вычислительной системы. Масштабируемость должна обеспечиваться архитектурой и конструкцией компьютера, а также соответствующими средствами программного обеспечения.

Добавление каждого нового процессора в действительно масштабируемой системе должно давать прогнозируемое увеличение производительности и пропускной способности при приемлемых затратах. Одной из основных задач при построении масштабируемых систем является минимизация стоимости расширения компьютера и упрощение планирования. В идеале добавление процессоров к системе должно приводить к линейному росту ее производительности. Однако это не всегда так. Потери производительности могут возникать, например, при недостаточной пропускной способности шин из-за возрастания трафика между процессорами и основной памятью, а также между памятью и устройствами ввода/вывода. В действительности реальное увеличение производительности трудно оценить заранее, поскольку оно в значительной степени зависит от динамики поведения прикладных задач.

Возможность масштабирования системы определяется не только архитектурой аппаратных средств, но зависит от заложенных свойств программного обеспечения.

Масштабируемость программного обеспечения затрагивает все его уровни от простых механизмов передачи сообщений до работы с такими сложными объектами как мониторы транзакций и вся среда прикладной системы. В частности, программное обеспечение должно минимизировать трафик межпроцессорного обмена, который может препятствовать линейному росту производительности системы. Аппаратные средства (процессоры, шины и устройства ввода/вывода) являются только частью масштабируемой архитектуры, на которой программное обеспечение может обеспечить предсказуемый рост производительности. Важно понимать, что простой переход, например, на более мощный процессор может привести к перегрузке других компонентов системы. Это означает, что действительно масштабируемая система должна быть сбалансирована по всем параметрам.

Совместимость и мобильность программного обеспечения. Концепция программной совместимости впервые в широких масштабах была применена разработчиками системы IBM/360. Основная задача при проектировании всего ряда моделей этой системы заключалась в создании такой архитектуры, которая была бы одинаковой с точки зрения пользователя для всех моделей системы независимо от цены и производительности каждой из них. Огромные преимущества такого подхода, позволяющего сохранять существующий задел программного обеспечения при переходе на новые (как правило, более производительные) модели были быстро оценены как

производителями компьютеров, так и пользователями и начиная с этого времени практически все фирмы-поставщики компьютерного оборудования взяли на вооружение эти принципы, поставляя серии совместимых компьютеров. Следует заметить однако, что со временем даже самая передовая архитектура неизбежно устаревает и возникает потребность внесения радикальных изменений архитектуру и способы организации вычислительных систем.

В настоящее время одним из наиболее важных факторов, определяющих современные тенденции в развитии информационных технологий, является ориентация компаний-поставщиков компьютерного оборудования на рынок прикладных программных средств. Это объясняется прежде всего тем, что для конечного пользователя в конце концов важно программное обеспечение, позволяющее решить его задачи, а не выбор той или иной аппаратной платформы. Переход от однородных сетей программно совместимых компьютеров к построению неоднородных сетей, включающих компьютеры разных фирм-производителей, в корне изменил и точку зрения на саму сеть: из сравнительно простого средства обмена информацией она превратилась в средство интеграции отдельных ресурсов - мощную распределенную вычислительную систему, каждый элемент которой (сервер или рабочая станция) лучше всего соответствует требованиям конкретной прикладной задачи.

Этот переход выдвинул ряд новых требований. Прежде всего такая вычислительная

среда должна позволять гибко менять количество и состав аппаратных средств и программного обеспечения в соответствии с меняющимися требованиями решаемых задач. Во-вторых, она должна обеспечивать возможность запуска одних и тех же программных систем на различных аппаратных платформах, т.е. обеспечивать мобильность программного обеспечения. В третьих, эта среда должна гарантировать возможность применения одних и тех же человеко-машинных интерфейсов на всех компьютерах, входящих в неоднородную сеть. В условиях жесткой конкуренции производителей аппаратных платформ и программного обеспечения сформировалась концепция открытых систем, представляющая собой совокупность стандартов на различные компоненты вычислительной среды, предназначенных для обеспечения мобильности программных средств в рамках неоднородной, распределенной вычислительной системы.

Одним из вариантов моделей открытой среды является модель OSE (Open System Environment), предложенная комитетом IEEE POSIX. На основе этой модели национальный институт стандартов и технологии США выпустил документ "Application Portability Profile (APP). The U.S. Government's Open System Environment Profile OSE/1 Version 2.0", который определяет рекомендуемые для федеральных учреждений США спецификации в области информационных технологий, обеспечивающие мобильность системного и прикладного программного обеспечения. Все ведущие производители

компьютеров и программного обеспечения в США в настоящее время придерживаются требований этого документа.

7 Международные стандарты планирования производственных процессов. MRP/ERP системы

- **MRP** (Material Requirement Planning) – планирование потребностей в материалах и ресурсах
- **MRP II** (Manufacturing Resource Planning) – планирование производственных ресурсов
- **ERP** (Enterprise Resource Planning) – система планирования ресурсов организации
- **CSRP** (Customer Synchronized Resource Planning) – планирование ресурсов организации, синхронизированное на потребителя
- **ERP II** (Enterprise Resource and Relationship Processing) – управление внутренними ресурсами и внешними связями организации

Внедрение

Классические ERP-системы, в отличие от так называемого «коробочного» программного обеспечения, относятся к категории «тяжелых» заказных программных продуктов, их выбор, приобретение и внедрение, как правило, требуют тщательного планирования в рамках длительного проекта с участием партнерской компании — поставщика или консультанта. Поскольку КИС строятся по модульному принципу, заказчик часто (по крайней мере, на ранней стадии таких проектов) приобретает не

полный спектр модулей, а ограниченный их комплект. В ходе внедрения проектная команда, как правило, в течение нескольких месяцев осуществляет настройку поставляемых модулей.

Достоинства

Использование ERP системы позволяет использовать одну интегрированную программу вместо нескольких разрозненных. Единая система может управлять обработкой, логистикой, дистрибуцией, запасами, доставкой, выставлением счёт-фактур и бухгалтерским учётом.

Единая! система безопасности, включенная в ERP, позволяет противостоять как внешним угрозам (например, промышленный шпионаж), так и внутренним (например, хищения). Совместно в связке с CRM-системой и системой контроля качества, ERP позволяют максимально удовлетворять потребности клиентов.

Недостатки

Множество проблем, связанных с ERP, возникают из-за недостаточного инвестирования в обучение персонала, а также в связи с недоработанностью политики занесения и поддержки актуальности данных в ERP...

Ограничения:

- Небольшие компании не могут позволить себе инвестировать достаточно денег в ERP и адекватно обучить всех сотрудников.

- Внедрение может оказаться очень дорогим.
- Иногда ERP сложно или невозможно адаптировать под документооборот компании и ее специфические бизнес-процессы.
- Система может страдать от проблемы "слабого звена" -- эффективность всей системы может быть нарушена одним департаментом или партнером.
- Сопротивление департаментов в предоставлении конфиденциальной информации уменьшает эффективность системы.
- Проблема совместимости с прежними системами.

Зарубежные ERP-системы

В числе самых известных программных продуктов, реализующих концепцию ERP, следует назвать в первую очередь системы mySAP ERP, MySAP All-in-One и SAP BusinessOne компании SAP AG и Oracle E-Business Suite, JD Edwards и PeopleSoft Enterprise компании Oracle. На российском рынке в сегменте среднего и малого бизнеса (SMB) лидирует компания Microsoft с системами Microsoft Dynamics AX (Axapta) и NAV (Navision).

В числе других решений можно отметить системы infor:COM, MAX+, SSA ERP LN (Baan) и SyteLine от фирмы Infor.

Существуют также менее универсальные решения, делающие ставку на расширение функциональности с конкретной отраслевой спецификой. Пример — система IFS

Applications компании IFS с расширенной функциональностью для производства и ремонтов.

Российские ERP-системы

Ряд российских программных систем также реализуют в той или иной мере функциональность вышеперечисленных ERP. Так, систему 1С:Управление производственным предприятием 8.0 некоторые считают полнофункциональной ERP-системой.

Еще примеры российских ERP системы Фрегат – Корпорация, АВА Системы.

7.1 Управление промышленными предприятиями в стандарте MRP II

MRP

Концепция Material Resource Planning (MRP) (конец 60-х) обеспечивала планирование потребностей предприятий в материалах. Преимущество - минимизация издержек, связанных со складскими запасами сырья, комплектующих, полуфабрикатов и прочего, а также с аналогичными запасами, находящимися на различных участках непосредственно в производстве.

В основе MRP лежит понятие Bill Of Material (BOM), то есть спецификации изделия, которая показывает зависимость внутреннего для предприятия спроса на сырье, комплектующие, полуфабрикаты и т.д. от плана выпуска (бюджета реализации) готовой продукции. При этом важную роль играет фактор времени, поскольку несвоевременная

доставка материалов может привести к срыву планов выпуска готовой продукции. Для учета временной зависимости производственных процессов, MRP информационной системе, «необходимо знать» технологию выпуска продукции (технологическую цепочку), то есть последовательность технологических операций и их продолжительность. На основании плана выпуска продукции, BOM и технологической цепочки в MRP – системе осуществляется расчет потребностей в материалах в зависимости от конкретных сроков выполнения тех или иных технологических операций (план потребностей, используется как стержень и в современных системах MRPII). MRP следует двум важнейшим принципам:

- логике зависимого спроса, т.е. если есть потребность в конечном изделии, значит есть потребность во всех его компонентах;
- обеспечивать требуемые компоненты как можно позднее, чтобы уровень запасов был минимальным.

Рисунок 7.1

Серьезный недостаток MRP. При расчете потребности в материалах не учитываются загрузка и амортизация производственных мощностей, стоимость рабочей силы, потребляемой энергии и т.д.

MRP в замкнутом цикле (конец 70-х**)**

Термин “замкнутый цикл” означает интегрированную систему с обратной связью от одной функции к другой, т.е. формировании производственной программы в масштабах всего предприятия и контроля ее выполнения на уровне подразделений. Информация передается обратно через вычислительную систему, но при этом никакие действия не предпринимаются. Принятие решения о корректировке плана остается за человеком.

MRPII - Manufacturing Resource Planning (планирование производственных ресурсов) - это специально разработанный набор методов управления бизнесом, которые поддерживаются вычислительными системами. В рамках MRP II можно уже планировать все производственные ресурсы предприятия: сырье, материалы, оборудование, людские ресурсы, все виды потребляемой энергии и пр. Прогнозирование, планирование и контроль производства осуществляется по всему циклу, начиная от закупки сырья и заканчивая отгрузкой товара потребителю.

Функции КИС стандарта MRP II

1. Планирование продаж и производства
2. Управление спросом
3. Составление плана производства
4. Планирование потребностей в материалах
5. Спецификация продуктов
6. Управление складом
7. Плановые поставки
8. Управление на уровне производственного цеха
9. Планирование производственных мощностей
10. Контроль входа/выхода
11. Материально-техническое снабжение

12. Планирование распределения ресурсов
13. Планирование и контроль производственных операций
14. Финансовое планирование
15. Моделирование
16. Оценка результатов деятельности

Обычно MRPII применяется на производственных предприятиях, в чисто коммерческих предприятиях аналогичную функцию выполняют системы DRP (планирование ресурсов для управления коммерческой деятельностью). В общем виде система управления предприятием, построенная в соответствии со стандартом MRPII, имеет следующий вид (рисунок 7.2):

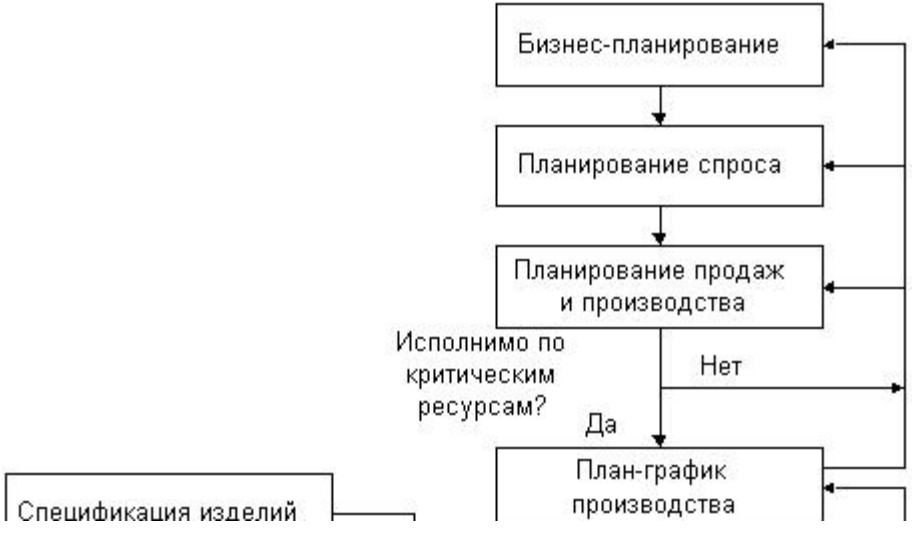


Рисунок 7.2 - Система управления предприятием

Ниже приводится краткая характеристика перечисленных функциональных блоков MRPII.

Бизнес-планирование. Процесс формирования плана предприятия наиболее высокого уровня. Планирование долгосрочное (до нескольких лет), план составляется в

стоимостном выражении. Наименее формализованный процесс выработки решений.

Планирование спроса. Процесс прогнозирования (планирования) спроса на определенный период (как правило, на квартал или на год).

Планирование продаж и производства. Бизнес-план и план спроса преобразуются в планы продаж основных видов продукции (как правило, от 5-ти до 10-ти). При этом производственные мощности могут не учитываться или учитываться укрупнённо. План носит среднесрочный характер.

Далее план продаж по видам продукции преобразуется в объёмный или объёмно-календарный план производства видов продукции. Под видом здесь понимаются семейства однородной продукции. В этом плане впервые в качестве планово-учётных единиц выступают изделия, но представления о них носят усреднённый характер. Например, речь может идти о всех легковых переднеприводных автомобилях, выпускаемых на заводе (без уточнения моделей). Часто этот модуль объединяется с предыдущим (как на приведенной схеме).

План-график выпуска продукции. План производства преобразуется в график выпуска продукции. Как правило, это среднесрочный объёмно-календарный план, задающий количества конкретных изделий (или партий) со сроками их изготовления.

Планирование потребностей в материальных ресурсах. В ходе планирования на этом уровне определяются в количественном выражении и по срокам потребности в

материальных ресурсах, необходимых для обеспечения графика выпуска продукции. Входными данными для планирования потребностей в материалах являются спецификации изделий (состав и количественные характеристики комплектующих конкретного изделия) и размер текущих материальных запасов.

Планирование производственных мощностей. Как правило, в этом модуле выполняются расчёты по определению и сравнению располагаемых и потребных производственных мощностей. С изменениями этот модуль может применяться не только для производственных мощностей, но и для других видов производственных ресурсов, способных повлиять на пропускную способность предприятия. Подобные расчёты, как правило, производятся после формирования планов практически всех предыдущих уровней с целью повышения надёжности системы планирования. Входными данными при планировании производственных мощностей являются также маршрутизация выпускаемых изделий.

Управление заказами клиентов. Здесь реальные потребности клиентов сопоставляются с планами выпуска продукции.

Управление на уровне производственного цеха. Здесь формируются оперативные планы-графики. В качестве планово-учетных единиц могут выступать детали (партии), сборочные единицы глубокого уровня, детале-(партие) операции и т. п. Длительность планирования невелика (от нескольких дней до месяца).

Оценка исполнения. По сути, в данном модуле оценивается реальное исполнение всех вышеперечисленных планов с тем, чтобы внести корректировки во все предыдущие циклы планирования.

Связь между уровнями в MRPII обеспечивается универсальной формулой, на которой строится система. Задача планирования на каждом уровне реализуется как ответ на четыре вопроса:

1. Что необходимо выполнить?
2. Что необходимо для этого?
3. Что есть в наличии?
4. Что необходимо иметь?

В роли ответа на первый вопрос всегда выступает план более высокого уровня. Этим и обеспечивается связь между уровнями. Структура ответов на последующие вопросы зависит от решаемой задачи.

MRPII – центральная часть любой КИС на производственных предприятиях. Объединение процедур обработки заказов на продажу, бухгалтерского учета, закупок и выписки счетов-фактур с производством на основе одной базы данных реального времени позволяет управлять деятельностью предприятия. MRPII включает финансовое планирование и возможность анализа по запросам “что-если”. Но это управление не распространяется на конструкторские

разработки, составление сметы, кадры, сбыт и распределение продукции, обслуживание, т.е. подразделения не объединены в одну систему. Именно эти вопросы решались разработчиками ERP систем в 90-х годах, чтобы обеспечить полностью интегрированные системы для управления производственными предприятиями, в основе которых были заложены принципы MRPII.

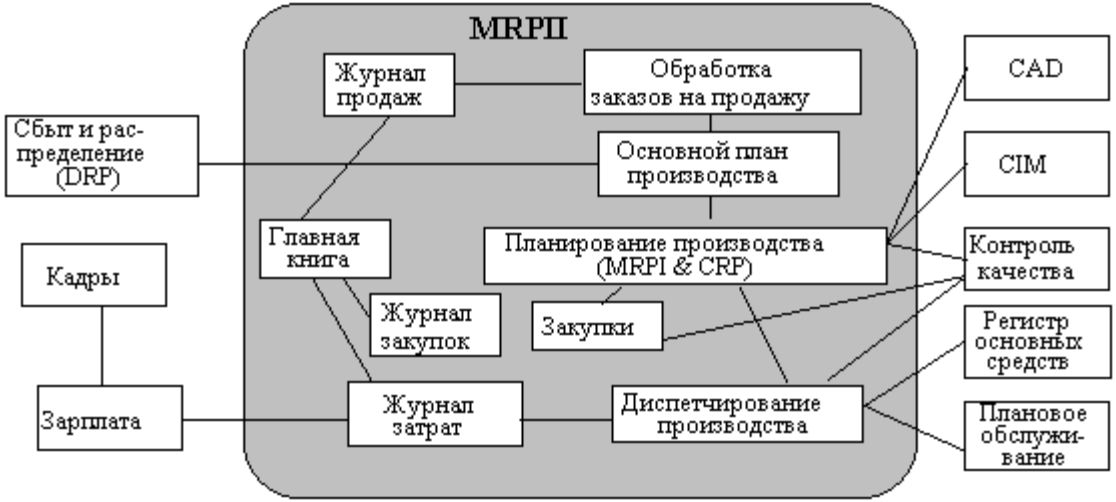


Рисунок 7.3

Преимущества MRP II

- улучшение обслуживания заказчиков за счет своевременного исполнения поставок;
- сокращение цикла производства и цикла выполнения заказа, следовательно,

более гибкая реакция на спрос;

- сокращение незавершенного производства, т.к. работа не будет выдаваться, пока не потребуется “точно ко времени” для удовлетворения конечного спроса;
- значительное сокращение запасов, что позволяет более экономно использовать складские помещения и сокращает расходы на хранение;
- сбалансированность запасов - уменьшение дефицита и устаревших запасов;
- повышение производительности, т.к. людские ресурсы и материалы будут использоваться в соответствии с заказами с меньшими потерями; также возможно использовать анализ “что-если”, чтобы проверить, соответствует ли производство задачам предприятия по получению прибыли;

По существу, эти преимущества позволят одновременно добиться улучшения исполнения поставок, сокращения запасов, длительности циклов, текущих затрат и получить более высокую прибыль.

7.2 Современная структура модели MRP/ERP

Сегодня модель MRP/ERP включает в себя следующие подсистемы, которые часто называют также блоками или сериями:

1. управления запасами;
2. управления снабжением;
3. управления сбытом;

4. управления производством;
5. планирования;
6. управления сервисным обслуживанием;
7. управления цепочками поставок;
8. управления финансами.

7.2.1 Управление запасами

Эта подсистема обеспечивает реализацию следующих функций (рисунок 4.1):

1) Inventory Control – мониторинг запасов;

2) Physical Inventory – регулирование и инвентаризация складских остатков.

При решении задач управления запасами –производятся:

- обработка и корректировка всей информации о приходе, движении и расходе сырья и материалов, промежуточной продукции и готовых изделий;
- учет запасов по складским ячейкам, выбор индивидуальных стратегий контроля, пополнения и списания запасов по каждой позиции номенклатуры сырья и материалов, и т.д.;
- учет нормативной и текущей фактической стоимости запасов;
- отслеживание прохождения отдельных партий запасов и серий изготавливаемой продукции.

Рисунок 7.4 - Управление запасами

7.2.2 Управления снабжением

Подсистема реализует следующие функции (рисунок 7.5):

1) Purchase Orders - заказы на закупку;

2) Supplier Schedules - график поставок;

3) MRP - планирование потребности в материалах, понимаемое как управление заявками на закупку.

Рисунок 7.5 - Управление снабжением

7.2.3 Управление сбытом

Базовыми функциями этой подсистемы являются:

1) Sales Quotations -квотирование продаж;
2) Sales Orders / Invoices -заказы на продажу (счета фактуры);
3) Customer Schedules -график продаж потребителям;
4) Configured Products -конфигурирование продуктов;
5) Sales Analysis -анализ продаж;
6) Distributed Resource Planning **(DRP)** -управления ресурсами распределения.

Потребитель

Рисунок 7.6 - Управление сбытом

7.2.4 Управления производством

В этой подсистеме реализуются следующие функции (рисунок 7.7), соответствующие различными типам производственных процессов:

1) Product Structures -спецификация изделий, определяющая, какие материалы и комплектующие используются в производимом изделии;

2) Routings / Work Centers -операции/центры переработки, включает в себя описание цехов, участков, рабочих мест;

3) Formula / Process -технологические процессы производства продукции с маршрутизацией по рабочим центрам для объемного (процессного) производства.

4) Work Orders – наряд-задание (сменное задание) на производство работ для

позаказного и мелкосерийного производства;

5) Shop Floor Control -управление трудозатратами (диспетчирование);

6) Repetitive -поточное производство (для серийного и массового производства).

7) Quality Management -управление качеством, то есть описание различных проверок изделий во время производственного процесса.

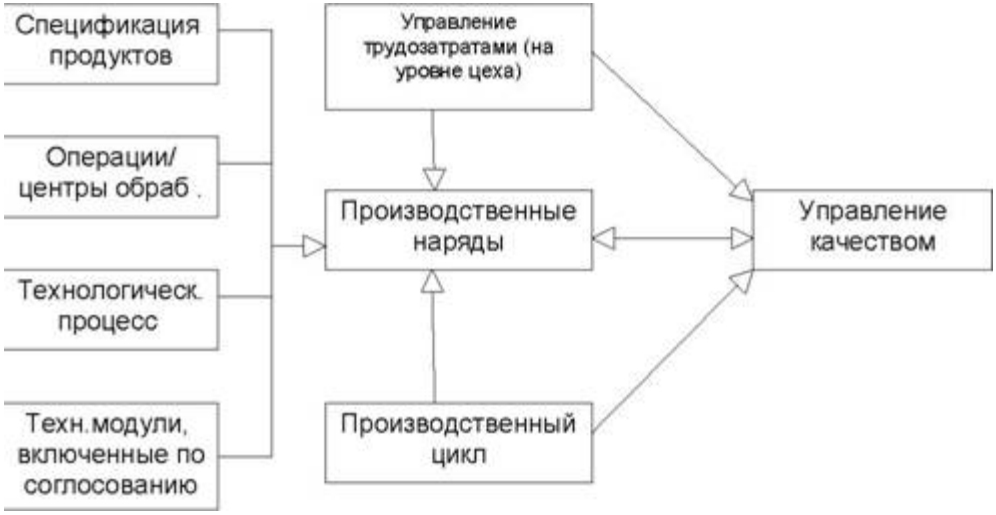


Рисунок 7.7 - Управление производством

7.2.5 Планирование

В модели MRP/ERP предусматривается сквозное планирование, согласование и оперативная корректировка планов и действий снабженческих, производственных и сбытовых звеньев предприятия.

Подсистема планирования реализует следующие функции:

1) Product Line Planning (PLP) – финансовое планирование товарно -номенклатурных групп (ТНГ);
2) Master Scheduling Planning (MSP) – главный календарный график или объемно календарное планирование;
3) Distribution Resource Planning (DRP) – планирование распределения ресурсов (RCP);
4) Materials Requirements Planning (MRP) – планирование потребности материалов;
5) Capacity Requirements Planning (CRP)– планирование потребления мощностей.

Эту функциональность можно условно отнеси к трем уровням планирования, отражающим иерархию планов в ERP-модели (рисунок 7.8).

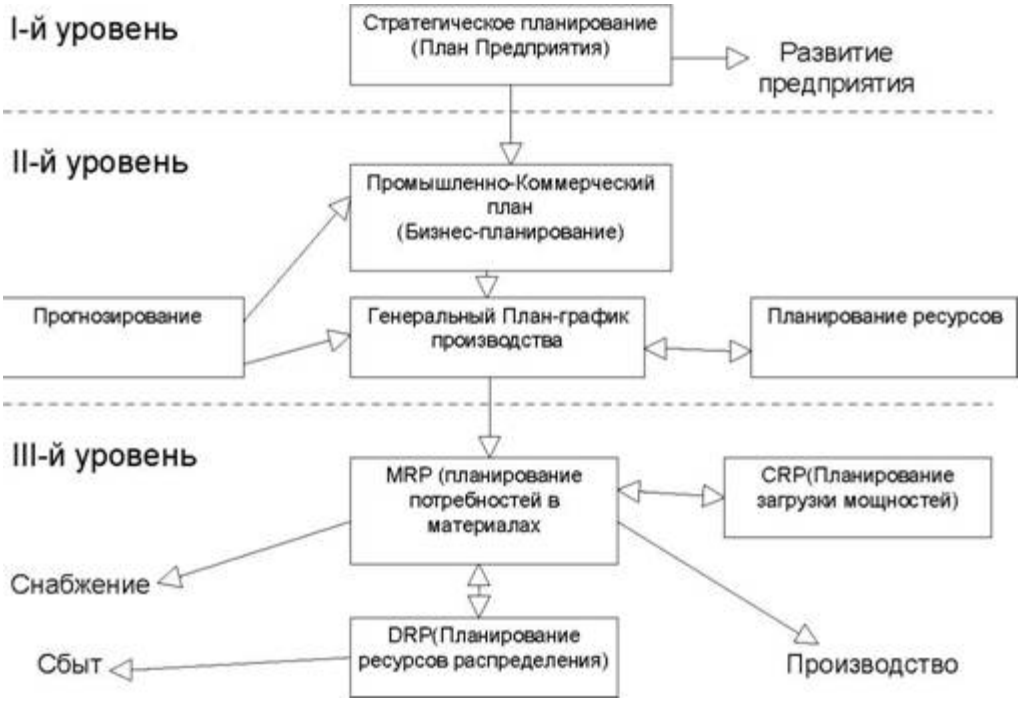


Рисунок 7.8 - Иерархия планов в ERP-модели

7.2.6 Управление сервисным обслуживанием

Эта подсистема активно используется компаниями, которые не только производят и продают свою продукцию, как, например, производители продовольствия, но и обеспечивают послепродажное техническое обслуживание и техническую поддержку своей продукции. Подсистема обеспечивается полный спектр необходимых функций: от создания графика технического обслуживания, заказа комплектующих, учета контрактов

на обслуживание и формирования счетов до учета прибыли, получаемой от послепродажного обслуживания.

7.2.7 Управление цепочками поставок

Эта подсистема предназначена для обеспечения эффективного управления материальными и соответствующими им информационными потоками: от поставщика через производство к потребителю. Реализованная в подсистеме идеология «управления глобальными цепочками поставок» дает промышленным предприятиям возможность представлять свою деятельность в виде так называемых эффективных цепочек логистики: от поставщиков сырья и комплектующих до продажи готовых изделий конечному потребителю. При этом обеспечиваются широкие возможности управления транснациональными компаниями, координации распределенного между многими дочерними компаниями производства.

7.2.8 Управление финансами

В соответствии с идеологией MRP/ERP эта подсистема полностью интегрирована со всеми остальными и позволяет оперативно получать информацию о финансовых потоках, связанных с потоками материальными (рисунок 7.9), о текущем финансовом состоянии компании, и помогает находить оптимальные финансово -экономические решения. Сквозное управление материальными потоками находит свое отражение в управлении финансовыми потоками (движении денежных средств).

В подсистеме реализована функциональность:

1) General Ledger – главная бухгалтерская книга, предназначенная для отражения финансовых транзакций и ведения бухгалтерского учета;
2) Multiple Currency – мультивалютность, для ведения учета в разных валютах;
3) Accounts Receivable -дебиторская задолженность;
4) Accounts Payable -кредиторская задолженность;
5) Payroll -заработная плата;
6) Cost Management -управление себестоимостью;
7) Cash Management -управление платежами;
8) Fixed Assets -учет основных средств.

Рисунок 7.9 - Обращение финансовых и материальных потоков

Модель MRP/ERP реализована в ряде информационных систем (ERP –систем) корпоративного уровня. Согласно статистическим данным, полученным при анализе использования ERP-систем в США, результатом внедрения таких систем на предприятиях является сокращение объемов запасов в среднем на 17 %, уменьшение затрат за закупку сырья и материалов на 7 %, повышение рентабельность производства в среднем на 30% и качества выпускаемой продукции на 60%.

Достоинством и одновременно недостатком классических систем ERP (SAP R/3, BAAN, Oracle Application) является их универсальность. У них есть модели для любого типа производственного процесса, и количество автоматизированных рабочих мест определяется исключительно финансовыми возможностями заказчика. Проект с использованием такой системы не может обойтись дешевле 500 тысяч долларов, а чаще всего стоит несколько миллионов долларов. Эти системы оптимальны для компаний, ведущих масштабный бизнес.

Для компаний среднего масштаба или имеющих не слишком диверсифицированный бизнес больше подходят другие системы ERP. Эти продукты более специализированы и предназначены для самого массового сегмента рынка -среднего и малого бизнеса, то есть для компаний с годовым оборотом от 3 до 10 млн. долларов и количеством работающих от 100 до 1000 человек. Типовая стоимость проекта по внедрению такой системы составляет от 50 до 250 тысяч долларов.

8 Основные аспекты автоматизации деятельности предприятия на примере финансово-управленческих систем

Современные системы автоматизации условно можно разделить на два типа: западные системы управления, реализующие принципы Enterprise Resource Planning (ERP) (основа - планирование производства), и программные комплексы отечественных разработчиков. Последние также часто называют финансово-управленческими (финансово-учетными) системами, потому что главное их назначение – управление и учет материальных и финансовых ценностей.

Основные черты финансово-учетных систем

Как правило, эти системы имеют модульную архитектуру, т. е. состоит из ряда подсистем, отвечающих за отдельные участки автоматизации. Все модули используют единое информационное пространство - ведутся общие списки товаров, клиентов и пр. Благодаря модульной архитектуре можно выбрать конфигурацию комплекса, максимально отражающую структуру компании и предоставляющую только необходимые функции. Типовой состав подсистем позволяет приблизительно оценить возможности финансово-учетных систем:

- центральная бухгалтерия - ядро комплекса, ведет учет на уровне бухгалтерских счетов и проводок, формирует баланс предприятия;
- учет хозяйственных операций - торгово-закупочная деятельность, складской учет

- касса - формирование приходных и расходных ордеров, подготовка авансовых отчетов, ведение кассовых книг;
- учет счетов-фактур - ведение журналов счетов-фактур, книг продаж и покупок в зависимости от учетной политики предприятия (по отгрузке и оплате) в полном соответствии с последними постановлениями по бухучету;
- розничная торговля - учет продаж товаров в розницу;
- основные фонды - учет основных средств, ведение инвентарных карточек основных средств, актов движения, прихода, списания; расчет амортизации;
- кадровая служба - учет сотрудников предприятия, ведение штатного расписания, учет больничных, отпускных, командировочных;
- зарплата - расчет как сдельной, так и повременной зарплаты, подготовка отчетности в Пенсионный фонд и Государственную Налоговую инспекцию;
- учет ТМЦ - ведение картотеки товарно-материальных ценностей, карточек складского учета, формирование актов прихода, движения, расчет амортизации.

Как правило, финансово-учетные системы состоят из модулей, автоматизирующих материальный учет, кадровый учет, составление бухгалтерской отчетности, а также автоматизацию основной деятельности организации. Но если область внутреннего учета более или менее одинакова для всех организаций, то для основной деятельности это не так: программные комплексы для разных отраслей значительно отличаются друг от друга.

Поэтому можно сказать, что финансово-учетная система - это, по существу, внутренний учет плюс отраслевое решение.

Естественно, организация бизнес-процессов может значительно различаться на разных предприятиях даже в рамках одной отрасли. Так что система автоматизации должна уметь изменять свои функциональные возможности в довольно широком диапазоне. Обычно это достигается с помощью встроенного в комплекс языка программирования, позволяющего описывать нетипичные (не осуществленные в тиражной версии продукта) решения. Современная система управления состоит из двух частей: неизменяемого ядра и дополнительного набора функций, создаваемого с использованием встроенных средств настройки. Очень важный момент - как соотносятся между собой эти части: если большинство функций "зашито" в ядре, то программный комплекс получится негибким; в противном случае резко возрастет объем работ по настройке системы на конкретное предприятие.

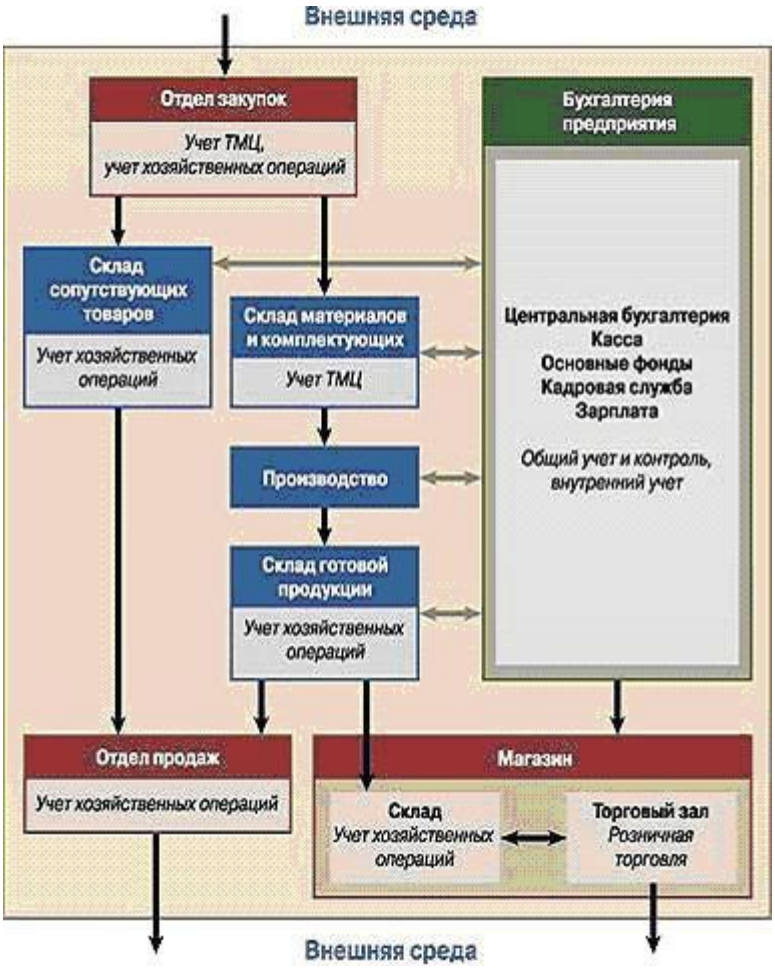


Рисунок 8.1 - Схема применения финансово-учетных систем на предприятии

В общем виде схема применения финансово-учетных систем на предприятии

представлена на рисунке 8.1. Как видим, финансово-учетные системы охватывают практически всю сферу деятельности организации, за исключением производства. Однако многие фирмы-разработчики ПО сейчас ведут работы по устранению этого недостатка, добавляя все новые модули в действующие программные комплексы. Таким образом, можно выделить следующие характерные особенности существующих ныне финансово-учетных систем:

1. модульная архитектура: программный комплекс состоит из ряда подсистем, автоматизирующих отдельные участки учета;
2. принцип автоматизированного рабочего места (АРМ): при настройке системы принимаются во внимание обязанности конкретных сотрудников;
3. традиционно слабые возможности учета производственной деятельности;
4. адаптация к специфике российского законодательства и бухгалтерского учета;
5. система нацелена именно на учет и лишь предоставляет необходимую для принятия управленческого решения отчетную информацию о состоянии организации.

9 Области применения и примеры реализации информационных технологий управления корпорацией

В последние несколько лет компьютер стал неотъемлемой частью управленческой системы предприятий. Современный подход к управлению предполагает вложение денег в информационные технологии. Причем чем крупнее предприятие, тем больше должны быть подобные вложения.

Благодаря стремительному развитию информационных технологий наблюдается расширение области их применения. Если раньше чуть ли неединственной областью, в которой применялись информационные системы, была автоматизация бухгалтерского учета, то сейчас наблюдается внедрение информационных технологий во множество других областей. Эффективное использование корпоративных информационных систем позволяет делать более точные прогнозы и избегать возможных ошибок в управлении.

Из любых данных и отчетов о работе предприятия можно извлечь массу полезных сведений. И информационные системы как раз и позволяют извлекать максимум пользы из всей имеющейся в компании информации.

Именно этим фактом и объясняются жизнеспособность и бурное развитие информационных технологий — современный бизнес крайне чувствителен к ошибкам в управлении, и для принятия грамотного управленческого решения в условиях неопределенности и риска необходимо постоянно держать под контролем различные

аспекты финансово-хозяйственной деятельности предприятия (независимо от профиля его деятельности).

Поэтому можно вполне обоснованно утверждать, что в жесткой конкурентной борьбе большие шансы на победу имеет предприятие, использующее в управлении современные информационные технологии.

Рассмотрим наиболее важные задачи, решаемые с помощью специальных программных средств.

9.1 Бухгалтерский учет

Бухгалтерский учет — классическая и наиболее часто реализуемая на сегодняшний день область применения информационных технологий. Такое положение вполне объяснимо. Во-первых, ошибка бухгалтера может стоить очень дорого, поэтому очевидна выгода автоматизации бухгалтерии. Во-вторых, задача бухгалтерского учета довольно легко формализуется, так что разработка систем автоматизации бухгалтерского учета не представляет технически сложной проблемы.

Тем не менее разработка систем автоматизации бухгалтерского учета является весьма трудоемкой. Это связано с тем, что к системам бухгалтерского учета предъявляются повышенные требования в отношении надежности, максимальной простоты и удобства в эксплуатации. Следует отметить также постоянные изменения в бухгалтерском и налоговом учете.

9.2 Управление финансовыми потоками

Внедрение информационных технологий в управление финансовыми потоками также обусловлено критичностью этой области управления предприятия к ошибкам. Неправильно построив систему расчетов с поставщиками и потребителями, можно спровоцировать кризис наличности даже при налаженной сети закупки, сбыта и хорошем маркетинге. И наоборот, точно просчитанные и жестко контролируемые условия финансовых расчетов могут существенно увеличить оборотные средства фирмы.

9.3 Управление складом, ассортиментом, закупками

Далее, можно автоматизировать процесс анализа движения товара, тем самым отследив и зафиксировав те двадцать процентов ассортимента, которые приносят восемьдесят процентов прибыли. Это же позволит ответить на главный вопрос — как получать максимальную прибыль при постоянной нехватке средств?

«Заморозить» оборотные средства в чрезмерном складском запасе — самый простой способ сделать любое предприятие, производственное или торговое, потенциальным инвалидом. Можно просмотреть перспективный товар, вовремя не вложив в него деньги.

9.4 Управление производственным процессом

Оптимальное управление производственным процессом представляет собой очень трудоемкую задачу. Основным механизмом здесь является планирование.

Автоматизированное решение подобной задачи дает возможность грамотно

планировать, учитывать затраты, проводить техническую подготовку производства, оперативно управлять процессом выпуска продукции в соответствии с производственной программой и технологией.

Очевидно, что чем крупнее производство, тем большее число бизнес-процессов участвует в создании прибыли, а значит, использование информационных систем жизненно необходимо.

9.5 Управление маркетингом

Управление маркетингом подразумевает сбор и анализ данных о фирмах-конкурентах, их продукции и ценовой политике, а также моделирование параметров внешнего окружения для определения оптимального уровня цен, прогнозирования прибыли и планирования рекламных кампаний. Решения большинства этих задач могут быть формализованы и представлены в виде информационной системы, позволяющей существенно повысить эффективность маркетинга.

9.6 Документооборот

Документооборот является очень важным процессом деятельности любого предприятия. Хорошо отлаженная система учетного документооборота отражает реально происходящую на предприятии текущую производственную деятельность и дает управленцам возможность воздействовать на нее. Поэтому автоматизация документооборота позволяет повысить эффективность управления.

9.7 Системы поддержки принятия решений, системы интеллектуального анализа данных

Следующим немаловажным моментом в функционировании КИС является необходимость обеспечить помимо средств генерации данных также и средства их анализа. Имеющиеся во всех современных СУБД средства построения запросов и различные механизмы поиска хотя и облегчают извлечение нужной информации, но все же не способны дать достаточно интеллектуальную ее оценку, т. е. сделать обобщение, группирование, удаление избыточных данных и повысить достоверность за счет исключения ошибок и обработки нескольких независимых источников информации (как правило, не только корпоративных баз данных, но и внешних, расположенных, например, в Internet). Проблема эта становится чрезвычайно важной в связи с лавинообразным возрастанием объема информации и увеличением требований к инфосистемам по производительности — сегодня успех в управлении предприятием во многом определяется оперативностью принятия решений, данные для которых и предоставляет КИС. В этом случае на помощь старым методам приходит оперативная обработка данных (On-Line Analitical Processing, OLAP). Сила OLAP заключается в том, что в отличие от классических методов поиска запросы здесь формируются не на основе жестко заданных (или требующих для модификации вмешательства программиста и, следовательно, времени, т. е. об оперативности речь идти не может) форм, а с помощью гибких

нерегламентированных подходов. OLAP обеспечивает выявление ассоциаций, закономерностей, трендов, проведение классификации, обобщения или детализации, составление прогнозов, т. е. предоставляет инструмент для управления предприятием в реальном времени.

Не останавливаясь на тонкостях организации различных моделей OLAP (например, таких, как радиальная схема, “звезда”, “снежинка” или многомерные таблицы), суть работы OLAP можно описать как формирование и последующее использование для анализа массивов предварительно обработанных данных, которые еще называют предвычисленными индексами. Их построение становится возможным исходя из одного основополагающего предположения, — будучи средством принятия решений, OLAP работает не с оперативными базами данных, а со стратегическими архивами, отличающимися низкой частотой обновления, интегрированностью, хронологичностью и предметной ориентированностью. Именно неизменность данных и позволяет вычислять их промежуточное представление, ускоряющее анализ гигантских объемов информации.
Сегодня доступен целый ряд различных систем OLAP, ROLAP (реляционный OLAP), MOLAP (многомерный OLAP) — Oracle Express, Essbase (Arbor Software), MetaCube (Informix) и другие. Все они представляют собой дополнительные серверные модули для различных СУБД, способные обрабатывать практически любые данные. Интеграция КИС с системой оперативного анализа информации позволит во много раз увеличить

эффективность первой, поскольку данные в ней будут не просто храниться, а работать.

9.8 Предоставление информации о предприятии

Активное развитие Интернета привело к необходимости создания корпоративных серверов для предоставления различного рода информации о предприятии. Практически каждое уважающее себя предприятие сейчас имеет свой веб-сервер. Веб-сервер предприятия решает ряд задач, из которых можно выделить две основные:

1. Создание имиджа предприятия;
2. Максимальная разгрузка справочной службы компании путем предоставления потенциальным и уже существующим абонентам возможности получения необходимой информации о фирме, предлагаемых товарах, услугах и ценах.

Кроме того, использование веб-технологий открывает широкие перспективы для электронной коммерции и обслуживания покупателей через Интернет.

10 Распределенные системы

В последние 2-3 года резко возрос интерес к так называемым распределенным системам. Под **распределенными системами** обычно понимают программные комплексы, составные части которых функционируют на разных компьютерах в сети. Эти части взаимодействуют друг с другом, используя ту или иную технологию различного уровня - от непосредственного использования сокетов TCP/IP до технологий с высоким уровнем абстракции, таких, как RMI или CORBA.

Рост популярности распределенных систем вызван существенным ужесточением требований, предъявляемых заказчиком к современным программным продуктам. Пожалуй, важнейшими из этих требований являются следующие:

- Обеспечение масштабируемости систем, т.е. способности эффективно обслуживать как малое, так и очень большое количество клиентов одновременно.
- Надежность создаваемых приложений. Программный комплекс должен быть устойчив не только к ошибкам пользователей (это определяется главным образом качеством клиентских приложений), но и к сбоям в системе коммуникаций. Надежность подразумевает использование транзакций, т.е. гарантированного перехода системы в процессе функционирования из одного устойчивого и достоверного состояния в другое.

- Возможность непрерывной работы в течение длительного времени (так называемый режим 24x7, т.е. режим круглосуточной работы в течение недель и месяцев).
- Высокий уровень безопасности системы, под которой понимается не только контроль доступности тех или иных ресурсов системы и защищенность информации на всех этапах функционирования, но и отслеживание выполняемых действий с высокой степенью достоверности.
- Высокая скорость разработки приложений и простота их сопровождения и модификации с использованием программистов средней квалификации.

Оказалось, что обеспечить соответствие этим требованиям, используя традиционные технологии - а именно, двухзвенные системы “клиент-сервер”, в которых в качестве серверов выступают системы управления базами данных, почти невозможно.

Заметим, что далеко не для всех приложений вышеперечисленные требования являются существенными. Легко представить распределенную систему, которая функционирует на небольшом (до 100) количестве компьютеров, работающих в локальной сети, где нет проблем с перезапуском одного или двух-трех серверов в случае необходимости, а главными задачами, для решения которых используются распределенные технологии, являются задачи использования функциональности нескольких стандартных серверов приложений (например, текстовых процессоров, броузеров и электронных таблиц) в интересах клиентских задач. Необходимо иметь в

виду, что термин “распределенные системы” относится к огромному числу задач самого разного класса.

Причины построении распределенной системы. Приведем некоторые из них:

- Размещение часто используемых данных ближе к клиенту, что позволяет минимизировать сетевой трафик.
- Расположение часто меняющихся данных в одном месте, обеспечивающее минимизацию затрат по синхронизации копий данных.
- Увеличение надежности комплекса к единичным отказам серверов (горячее резервирование).
- Понижение стоимости системы за счет использования группы небольших серверов вместо одного мощного центрального сервера.

Если в информационной системе наличествуют все перечисленные факторы, то распределенная база данных может оказаться хорошим решением. Часто распределение базы данных изначально определяется требованиями бизнеса, например когда объединяется информация нескольких крупных подразделений одной компании и требуется постоянный обмен данными между филиалами.

Решение о распределенной базе данных оправданно и для систем, где есть четко выраженные группа меняющихся данных и группа устойчивых данных, по которым выполняются отчеты. Тогда в самом простом варианте работают два сервера: один

обслуживает часто меняющиеся данные — это, как правило, OLTP (On-Line Transaction Processing. — Прим. ред.), второй — отчеты, то есть DSS (Decision Support System. — Прим. ред.). Ряд СУБД не очень хорошо совмещает обработку OLTP- и DSS-потоков запросов, поскольку для этих двух типов потоков запросов оптимальные параметры конфигурации серверов будут различаться. Решение такой базы данных как распределенной может оказаться более выгодным.

Преимущества создания распределенной системы имеют свою цену — должны быть обеспечены:

- Непротиворечивость данных, независимо от того, какой клиент к какому серверу обратился.
- Производительность распределенной базы данных должна удовлетворять требованиям заказчика, а это может оказаться более сложной задачей, чем в случае централизованной базы данных.
- Надежность распределенной базы данных не должна уступать надежности централизованной базы данных.

Современные СУБД позволяют осуществлять запрос данных, находящихся на разных узлах распределенной сети в одном SQL-запросе. Прозрачность доступа СУБД также обеспечивают — в каких-то реализациях хуже, в каких-то лучше. Одна из самых распространенных проблем — более сложная обработка транзакций. Практически все

промышленные реализации поддерживают только двухфазную фиксацию транзакций, а в этом режиме не все типы отказов разрешимы. Распределенный deadlock (взаимоблокировка. — Прим. ред.) также может представлять значительную проблему, и большинство реализаций используют только таймауты для его детектирования.

Как бы то ни было, стратегия распределения данных должна определяться не политикой предприятия (что обычно пытается навязать руководство), а требованиями надежности и производительности системы. При построении распределенной базы данных следует уделить особое внимание проектированию топологии сети. Узлы распределенной базы данных должны быть соединены скоростными надежными линиями связи. Это же касается линий соединения узлов базы данных и серверов приложений. Наличие низкоскоростного и ненадежного канала связи между узлами распределенной базы данных резко повышает количество детектируемых отказов сети.

В распределенной базе данных могут быть использованы следующие типы вызовов:

- Удаленные DDL- и DML- операции, а также выборка данных.
- Синхронные удаленные вызовы процедур.
- Асинхронные удаленные вызовы процедур.
- Непротиворечивые снимки.
- Асинхронная симметричная репликация.

- Синхронная симметричная репликация.
- Вызов распределенного запроса (запрашивает данные на чтение и модификацию с нескольких узлов).

Распределенная база данных может обеспечить горизонтальную фрагментацию; например, в филиале чаще всего используют данные о клиентах, находящихся в городе N (CLIENT_PLACE = ‘N’). В этом случае на узле распределенной базы данных этого филиала может быть расположен фрагмент таблицы данных, выделенный согласно условию CLIENT_PLACE = ‘N’.

Стратегия распределения данных для каждой СУБД определяется по-своему и достойна отдельной книги. Определение стратегии преследует две цели: сократить нагрузку на сеть и сервер и/или повысить уровень готовности данных.

Следует отметить, что проектировщики должны четко представлять себе особенности реализации распределенной базы данных используемой СУБД. Не владеющий этой информацией проектировщик рискует создать нерабочую или плохо работающую схему, причем проявится это даже не на моделях, а в реальной эксплуатации. Если вы решили строить распределенную базу данных, позаботьтесь о наличии в штате квалифицированного администратора. Есть одно простое правило: проектировщик не может принять окончательного решения об определении стратегии распределения данных без администратора баз данных. Редко встречаются

проектировщики, которые имеют практический опыт администрирования 2-3 серверов баз данных и которые реально работали на них с распределенными базами данных. Для каждой СУБД принципы, влияющие на детали распределения базы данных, индивидуальны.

Особый интерес представляет неоднородная распределенная база данных. Это то, что постоянно, но безуспешно пытаются изживать и проектировщики, и администраторы баз данных. Никто не любит иметь дело со шлюзами данных. Но если неоднородной распределенной базы избежать не удалось, уделите особое внимание выбору шлюзов данных, а также ПО, обеспечивающему синхронизацию информации базах данных под управлением разных СУБД. Если синхронизация данных на узлах под управлением одной СУБД может быть обеспечена самой СУБД (что реализуется большинством производителей), то синхронизация данных на узлах под управлением разных СУБД обеспечивается, как правило, специальным ПО. Тщательно оцените, что будет дешевле: использовать это ПО или импортировать данные и схему и сделать распределенную базу данных однородной.

Следует также отметить, что в подавляющем большинстве реализаций каждый из узлов распределенной базы данных может обслуживаться параллельным сервером баз данных, и это является важным критерием повышения производительности системы. Промышленные СУБД также используют параллельный доступ к хранилищам данных;

RAID, например, могут размещать данные, используя не файловую систему операционной системы, а свою файловую систему. Реализация всех этих возможностей существенно различается у различных СУБД. Это означает, что выбор параллельного сервера баз данных должен осуществляться только после серьезных консультаций с администраторами тех СУБД, которые рассматриваются как возможные претенденты на роль сервера баз данных в проекте.

Метод распределения данных по дискам для поддержки параллельных вычислений во многом зависит и от особенностей реализации СУБД. Администратор базы данных не может выбрать такой метод в отрыве от особенностей конкретной информационной системы. Еще одна возможность параллельной обработки данных, предоставляемая СУБД: обработка одного запроса несколькими менеджерами ресурсов. В реализациях также имеется возможность использования одного хранилища данных несколькими серверами баз данных (Parallel Server). Такая архитектура может быть эффективно использована на кластерах.

10.1 Распределенные БД в Oracle и Oracle в распределенных БД

СУБД Oracle позволяет поддерживать связь не только между клиентами и сервером, но и между серверами. Построение распределенных БД открывает возможности для решения целого комплекса задач: собрать в единое целое данные, хранящиеся в разных местах, увеличить серверную мощность системы, добавив в нее новые серверы

(воспользоваться так называемой горизонтальной масштабируемостью), сосредоточить данные в непосредственной близости от их потребителей, сохраняя при этом целостность системы, и многие другие.

Концепция построения распределенных БД в Oracle основана на децентрализованной их организации. Серверы взаимодействуют друг с другом с помощью уже знакомого нам протокола SQL*Net. Ссылки друг на друга - так называемые каналы связи БД (database links) - серверы хранят в качестве объектов БД. В свою очередь полное имя объекта может включать в себя канал связи (т. е. вместо самого объекта в локальной базе данных может храниться как бы ссылка на него): это, безусловно, требует обеспечения уникальности имен серверов ("сервисов" в терминологии Oracle) в сети, что достигается с помощью иерархической организации доменов, подобной существующей в Internet.

Поскольку вместо "настоящих" имен объектов можно использовать их синонимы (также в свою очередь являющиеся объектами БД), то приложение клиента может попросту не знать, является ли данный объект локальным для сервера, с которым установлена связь, или нет. Конечно, механизм каналов связи и синонимов - лишь внешняя сторона той системы, которая позволяет сделать структуру распределенной БД Oracle абсолютно прозрачной для приложений независимо от размещения данных и режима взаимодействия серверов. А таких вариантов организации Oracle предлагает

множество - как говорится, на любой вкус, хотя точнее было бы сказать, для любой ситуации.

Синхронная связь без тиражирования данных

Рассмотрим сначала самый простой, с логической точки зрения, вариант, при котором любой объект распределенной БД хранится только в одном экземпляре (не тиражируется). Это неизбежно означает, что если какая-либо транзакция (или запрос) пользовательского приложения включает в себя обращения к удаленным (расположенным не на том сервере, с которым установлена связь) объектам, эта транзакция (запрос) не может быть завершена, пока все задействованные серверы не обменяются необходимой информацией (т. е. они взаимодействуют в синхронном режиме).

Наиболее сложные проблемы при этом возникают при выполнении транзакций, одновременно изменяющих данные, хранящиеся на нескольких серверах (такие транзакции называются распределенными в отличие от удаленных, которые хотя и изменяют удаленные данные, но только на одном сервере). Суть проблемы в том, что при любых обстоятельствах нужно обеспечить, чтобы транзакция либо завершилась, либо "откатилась" на всех затронутых ею серверах (в противном случае возникнет рассогласование данных в распределенной БД). Вот почему при работе распределенной БД требуется использовать двухфазную фиксацию транзакций.

Схема алгоритма двухфазной фиксации упрощенно сводится к тому, что на первой

фазе (подготовке к фиксации) сервер-инициатор транзакции рассылает соответствующий запрос другим серверам (находящимся для него "в непосредственной видимости") и ждет ответа. Каждый из этих серверов может в случае необходимости повторить то же действие рекурсивно. Если все затронутые транзакцией серверы информируют о готовности к ее завершению в своих ответах, инициируется вторая фаза - собственно завершение транзакции.

Описанная схема действительно сильно упрощена, поскольку основные сложности при двухфазной фиксации транзакций начинаются тогда, когда встает вопрос об обеспечении корректного и предсказуемого поведения системы в случае выхода из строя отдельных серверов или временных разрывов линий связи. Самое простое - переложить решение данной проблемы на прикладного программиста (что фактически и делается в ряде СУБД), но, даже если это требуется в ограниченном наборе ситуаций, говорить о прозрачности структуры распределенной БД для приложений в таком случае уже нельзя. Реализация двухфазного завершения в Oracle такова, что все проблемы разрешаются автоматически, хотя возможность вмешательства администратора предусмотрена.

Как нетрудно догадаться, организация распределенной БД в данном варианте требует наличия надежных, постоянно доступных и достаточно быстрых линий связи между серверами. Как же быть, если все эти условия не выполнены?

Тиражирование данных

Предположим, что банк помимо центрального офиса имеет ряд филиалов, связанных с ним выделенными линиями связи, не обладающими, однако, достаточной пропускной способностью для подключения рабочих мест в филиалах к центральной БД в режиме клиентов. Напрашивается решение с использованием распределенной БД, организованной так, чтобы данные, чаще всего требующиеся в филиале, физически располагались на его локальном сервере. При этом возникает вопрос: как обеспечить возможность доступа из филиалов к центральной БД и из центрального офиса к данным в филиалах?

Если организовать распределенное хранение данных, как в предыдущем варианте, любой такой запрос (или транзакция) может выполняться с большой задержкой. В качестве выхода можно предложить хранить объекты данных в распределенной БД не в единственном экземпляре, а тиражировать их на всех серверах, где к ним может потребоваться быстрый доступ. Естественно при этом возникает необходимость каким-либо образом обеспечить синхронизацию различных копий одних и тех же данных, иначе распределенная БД потеряет свою целостность и превратится в хаос.

Отличие промышленных систем от игрушечных

Методика обеспечения целостности распределенной БД - лишь один из аспектов, определяющих упомянутое различие. Если попробовать сформулировать основной его критерий в общем виде, то, пожалуй, можно предложить следующую формулировку: в

промышленной системе всегда можно дать однозначный ответ на вопрос "А что будет, если..?". Другими словами, промышленная система должна обеспечивать своими внутренними средствами предсказуемость своего состояния независимо от внешних условий. Безусловно, любое общее определение страдает неполнотой (в самом деле, формально наиболее предсказуемым является состояние системы, которая вообще никогда не работает), однако, по мнению автора, оно все-таки дает идею, необходимую для понимания сути утверждения о том, что Oracle предоставляет технологию для создания именно промышленных систем.

В качестве поясняющего примера вернемся к проблеме синхронизации тиражируемых данных. Рассмотрим вопрос об организации связи между серверами в распределенной БД. Очень соблазнительной является идея: если уж мы не требуем, чтобы целостность данных (в частности соответствие тиражированных копий друг другу) обеспечивалась в любой момент времени, нельзя ли организовать указанную связь на основе электронной почты? На первый взгляд, ничего опасного в этом нет, но вспомним, что электронная почта не гарантирует, что "письма" будут получены адресатом в той же последовательности, в которой они были посланы и даже что все они вообще будут получены. Даже если с помощью специального контроля на приеме последствия данной неоднозначности сводятся к минимуму, все равно оказывается невозможным предсказать, скажем, момент времени, когда информация об изменении одной из копий объекта дойдет

до других его копий, и в каком состоянии к этому моменту эта изменившаяся копия будет находиться. Означает ли это, что электронной почтой в распределенной БД пользоваться вообще нельзя? И да, и нет. Все зависит от требований к системе. Если "свежесть" тиражированных данных требуется, допустим, в пределах суток, а последствия непредсказуемости системы и потери ее целостности не слишком разрушительны, то почему бы и нет? Но если все-таки требуется действительно промышленная система, то необходимо очень тщательно оценить все возможные последствия применения выбранного механизма взаимодействия и ввести в использование того же тиражирования такие ограничения, которые обеспечили бы требуемый уровень целостности системы.

Если вернуться к теме целостности распределенной БД с тиражируемыми данными, то при ее проектировании необходимо, как минимум, задать себе следующие вопросы.

Допустимо ли временное рассогласование тиражированных данных?

Если да, то в каких временных пределах должна осуществляться процедура их согласования, и каким образом она должна инициироваться?

Не могут ли привести сбои каких-либо элементов системы к безвозвратной потере целостности распределенной БД, всегда ли система будет корректно (и предсказуемо) вести себя в случае тех или иных сбоев?

Какова должна быть дисциплина работы с тиражированными данными, чтобы исключить возможность конфликтов между модификацией различных копий одних и тех

же данных, или - если такие конфликты допускать - какова должна быть процедура их разрешения (в какой степени система будет способна делать это автоматически, и будет ли она извещать о возникновении конфликтов администратора)?

Варианты тиражирования данных в Oracle

Самым простым (и исторически реализованным первым) вариантом тиражирования в Oracle является механизм так называемых неизменяемых снимков (read-only snapshots). Он подразумевает создание удаленной копии таблицы (или ее подмножества), которая доступна только на чтение и обновляется по заданному сценарию и расписанию. Точнее, снимок определяется так же, как представление - view, т. е. он может быть основан и на нескольких таблицах.

Следующим по своей логической сложности вариантом является организация изменяемых снимков, предоставляющая возможность модификации удаленных копий. Однако, как и в предыдущем случае, отношения серверов при этом асимметричны (один из них является владельцем "оригинала" данных, хотя и подверженного удаленным изменениям).

Последним "атомарным" вариантом тиражирования в Oracle (ибо возможны также любые комбинации) является тиражирование с множественными хозяевами (multi-master site replication). При данном варианте полностью тиражируются целые наборы объектов БД (в них помимо таблиц могут входить индексы, представления, триггеры, пакеты

хранимых процедур, синонимы, генераторы последовательностей). При этом тиражируются все определения и атрибуты объектов, так что в результате все хозяева их копий становятся равноправными. Любые изменения тиражированных данных непосредственно передаются ("распространяются") всем хозяевам (в отличие от варианта изменяемых снимков, где несколько снимков одного объекта могут обменяться изменениями только через посредство хозяина этого объекта). Такое решение, в частности, приводит к тому, что в системе не будет ни одного сервера, единичный выход из строя которого означал бы невозможность продолжения работы с набором тиражированных объектов.

Механизм распространения изменений в Oracle встроен в ядро системы и не требует использования никаких дополнительных программных продуктов. Любое изменение тиражированного объекта приводит в действие специальный триггер, который формирует вызов удаленной процедуры, необходимый для воспроизведения изменения во всех оставшихся копиях объекта. Далее в случае использования синхронного тиражирования сформированный вызов немедленно начинает выполняться, в случае же асинхронного тиражирования он помещается в специальную очередь отложенных вызовов, ожидая наступления момента своей передачи.

Очередь отложенных вызовов является объектом БД, а стало быть, ею можно управлять наряду с прочими объектами, после сбоев сервера она восстанавливается также

вместе с другими.

Без дисциплины работать трудно

Теперь, когда общий механизм тиражирования ясен, попробуем разобраться в некоторых аспектах его применения.

Как уже упоминалось, возможны два режима тиражирования: синхронный, когда все изменения данных распространяются немедленно, и асинхронный, когда по отношению к этим изменениям применяется алгоритм "запомнить и передать" (store and forward), а момент передачи выбирается по заданному правилу (или явно инициируется, например, после восстановления связи).

Синхронный вариант оправдан, по-видимому, в примере с банком и его филиалами, приведенном в самом начале раздела (в случае асинхронного тиражирования появляется опасность, что нечистый на руку клиент банка может, к примеру, несколько раз закрыть свой счет, быстро перемещаясь из филиала в филиал, - впрочем, как показано ниже, данная проблема может быть разрешена и иначе). По сравнению с вариантом использования синхронной связи без тиражирования, преимущества состоят в том, что, во-первых, запросы выполняются на локальном сервере и не требуют передачи данных между серверами, во-вторых, транзакции, хотя и требуют распространения изменений, все же выполняются для клиентов быстрее, поскольку это распространение происходит асинхронно по отношению к транзакциям (клиент не ждет окончания обмена данными

между серверами).

Другой важный пример использования синхронного тиражирования - поддержка "зеркальной" БД на резервном сервере, причем оба сервера (основной и резервный) в этом случае могут работать параллельно.

Что касается асинхронного тиражирования, то оно применимо тогда, когда нет возможности поддерживать постоянную связь между серверами, а временным рассогласованием данных можно поступиться (опять-таки в предположении, что состояние БД во времени предсказуемо). Если на Западе в качестве примера чаще всего приводят торгового агента, разъезжающего со своим ноутбуком и имеющего возможность лишь время от времени подключаться к сети, то в российских условиях, увы, аналогичная ситуация весьма типична. В тех регионах нашей могучей страны, где до сих пор самое надежное средство связи - это трактор, реализация даже предсказуемо работающего тиражирования данных представляется нелегкой задачей.

Впрочем, задача эта непроста и в случае, когда со связью все в порядке. Серьезной проблемой при проектировании системы является обеспечение ее корректного функционирования в условиях возможных конфликтов по обновлению различных копий одних и тех же данных на разных серверах. Здесь мы сталкиваемся с ситуацией, когда универсального решения попросту не существует, поэтому все, что может предоставить СУБД, - это средства для реализации различных вариантов решений и рекомендации по

их использованию.

Прежде всего, необходимо выбрать общую архитектуру системы, точнее, тот ее аспект, который относится к дисциплине доступа к данным. Можно, конечно, никакой дисциплины не устанавливать, но в этом случае задача проектировщика становится наиболее сложной, да и стопроцентная гарантия корректности работы не всегда достигается. Впрочем, давайте по порядку.

Самый простой вариант архитектуры, заведомо гарантирующий отсутствие конфликтов, предполагает, что для любого тиражированного элемента данных среди всех равноправных хранителей его копий выбирается один, так сказать, самый равноправный. Ему одному разрешается этот элемент данных изменять, всем остальным дозволяется только наблюдать за этим. Вариантов реализации такой схемы может быть множество: от самого простого - звездообразного (филиалам банка разрешается видеть данные центрального отделения, но не изменять их) до гораздо более изощренных со сложной сетью неизменяемых снимков.

Более развитый вариант архитектуры, также дающий гарантию отсутствия конфликтов, разрешает динамическую передачу права модификации (в англоязычных материалах обычно употребляется термин, буквально переводимый как "документооборот") от сервера к серверу. Каждый элемент данных снабжается специальным атрибутом "разрешена запись", и вводится процедура передачи этого

атрибута. Варианты реализации опять-таки могут быть различными. Характерным примером может служить информационная система торговой фирмы, где для каждого заказа эстафета последовательно передается от отдела продаж к управлению складом и далее к отделу доставки. Аналогичная схема в принципе решает упомянутую выше проблему "многократного закрытия счета" в банке.

Любые другие организации архитектуры тиражирования допускают возникновение конфликтов, следовательно, не остается ничего другого, как пытаться их разрешать. От СУБД здесь в первую очередь требуется обеспечить автоматическое обнаружение конфликтов и возможность извещения о них (наверное, излишне упоминать, что Oracle делает это), ибо момент возникновения конфликта зависит от механизма распространения изменений. Но просто извещать о проблеме и ждать ее "ручного" разрешения (по всей видимости, крайним, как всегда, окажется администратор БД) - вероятно, не лучший выход. По возможности нужно стремиться разрешать конфликты автоматически.

Oracle предлагает для этой цели набор процедур разрешения конфликтов, которые можно ассоциировать с группами столбцов таблицы. Руководствоваться при этом лучше всего здравым смыслом: скажем, если речь идет об описательных данных клиента, в качестве разрешающей функции разумно выбрать последнюю по времени модификацию, изменения количества товара на складе имеет смысл просуммировать и т. д.

Помимо внушительного набора стандартных функций разрешения конфликтов

можно использовать и свои собственные. Однако не все так просто. Далеко не все функции способны обеспечить стопроцентную конвергенцию (непременную установку в одно и то же значение) данных, особенно в случае, когда в конфликте участвует более двух серверов. Если применяется нестандартная функция, для определения ее свойств может потребоваться весьма тонкий анализ (для стандартных он уже проведен). Как бы то ни было, в системе необходимо предусматривать либо выбор только тех функций, которые обеспечивают конвергенцию данных при принятой дисциплине доступа к ним, либо использовать извещение администратора, когда функция не способна справиться с ситуацией самостоятельно.

В качестве резюме отметим еще раз, что тиражирование дает чрезвычайно широкие возможности, но относиться к нему необходимо с большой осторожностью и тщательно анализировать все аспекты возможного поведения системы при ее проектировании.

Поддержка резервной копии БД

Oracle предлагает еще один механизм, напоминающий тиражирование. Это предназначенная для повышения устойчивости системы к сбоям поддержка резервной копии БД (standby database). Смешивать данный механизм с тиражированием, пожалуй, не стоит, ибо резервная БД недоступна для пользователей одновременно с основной. Зато отсутствует дополнительная нагрузка на ядро основного сервера, связанная с распространением изменений. Дело в том, что для поддержания соответствия резервной и

основной баз данных используются журнальные файлы изменений, вообще говоря, предназначенные для восстановления БД после сбоев. Собственно, резервная БД как раз и функционирует в режиме перманентного восстановления, считывая журнальные файлы основной БД, переданные тем или иным способом на резервный сервер.

Свой среди чужих

Чтобы завершить тему, осталось сказать несколько слов о возможностях использования Oracle в гетерогенных распределенных БД. Выбор варианта решения задачи во многом зависит от составляющих гетерогенную систему СУБД.

Если это реляционные СУБД (MS SQL Server, Informix, Sybase, DB2, CA Ingres), можно использовать так называемые прозрачные шлюзы для объединения их с Oracle. Для пользователя такого шлюза полностью имитируется функциональная среда сервера Oracle при доступе к данным, хранящимся в "чужих" СУБД.

Для реализации шлюза используется промежуточный сервер Oracle (чаще всего он функционирует на том же компьютере, что и "чужой" сервер), за счет которого и достигается эффект "ораклизации" данных. Например, если пользователь вызывает хранимую процедуру на PL/SQL, то она фактически выполняется севером-шлюзом (СУБД других производителей с PL/SQL не работают), а "чужому" серверу передаются только SQL-предложения, содержащиеся или сформированные в процедуре.

Сложнее обстоит дело, если необходимо получить доступ к данным, хранящимся в

нереляционных СУБД (ADABAS, VSAM и пр.). В таком случае, как правило, невозможно формально однозначно отобразить эти данные в реляционные структуры Oracle, поэтому подход прозрачных шлюзов не применим. Тем не менее Oracle предлагает решение для таких ситуаций в виде процедурных шлюзов. В них вместо стандартного SQL для взаимодействия с "чужими" данными предоставляется библиотека процедур, с помощью которых разработчик реализует необходимое отображение данных.

Другой вариант решения проблемы в случае обращения к экзотическим системам хранения данных - использование мониторов транзакций.

Надо отметить, что при работе со шлюзами данные других СУБД органически включаются в среду распределенных БД Oracle: реализуется полнофункциональная поддержка синхронной связи между серверами без тиражирования и даже некоторые варианты тиражирования данных.

10.2 Администрирование распределенных систем на примере Oracle

Немного поговорим о тех средствах, которые Oracle предлагает в помощь администратору распределенной информационной системы.

В принципе, такая система может иметь (и обычно имеет) более одного администратора. Однако подобная организация имеет много недостатков. Не говоря уж о необходимости найти нужное количество высококвалифицированных специалистов, их деятельность нужно тесно координировать, например, для обеспечения единой политики

защиты данных. Понятны неудобства пользователя, который должен помнить множество паролей для доступа к различным серверам. Если же пароли везде одинаковы, то увеличивается вероятность потери их секретности.

Полезность централизации хотя бы части административных функций очевидна. На рынке программных продуктов существует достаточно много средств, направленных на решение именно этой задачи. К примеру, есть системы, реализующие централизованную идентификацию пользователей. Для некоторых из них (Kerberos, Sesame) Oracle предоставляет интерфейсы, что позволяет ввести их функциональность в распределенную БД на базе Oracle.

Что касается средств централизованного управления, то их на рынке тоже достаточно много, и значительная часть из них в той или иной степени способны работать с СУБД Oracle. Однако в данной области находиться в полной зависимости от технологии третьих фирм чересчур рискованно для крупных поставщиков передовых программных технологий, таких как Oracle: слишком большое значение приобретает данный аспект системы, чтобы игнорировать его в собственных разработках.

В значительной степени задачу централизованного администрирования распределенной БД позволяет решить Oracle Enterprise Manager, поставляемый сейчас в комплекте с сервером БД. Этот программный продукт позволяет с помощью графического интерфейса управлять сколь угодно сложной конфигурацией распределенной БД, включая

возможность определения удаленных заданий, выполняемых по заданному расписанию, извещения о различных событиях в системе и т. п.

Кроме этого, в состав программного продукта включен ряд утилит, выполняющих детальный мониторинг серверов БД, оптимизацию их параметров. В их состав входит также Oracle Expert - экспертная система, позволяющая провести оптимизирующую настройку любого сервера.

Важно еще и то, что Oracle Enterprise Manager предоставляет открытый интерфейс, позволяющий дополнять управляющую консоль администратора новыми средствами как третьим фирмам, так и самим пользователям. Можно рассчитывать поэтому на то, что в самое ближайшее время большая часть существующих на рынке средств централизованного администрирования систем на основе Oracle объединится в интегрированную управляющую среду.

11 OMG и её стандарт CORBA

CORBA (Common Object Request Broker Architecture) - это стандарт, набор спецификаций для промежуточного программного обеспечения (ППО, middleware) объектного типа.

Задача ППО, как известно, и заключается в связывании программных приложений для обмена данными.

Эволюция ППО - это путь от программ передачи информации между конкретными приложениями, через средства импорта- экспорта данных и организацию мостов между некоторыми приложениями, через SQL, RPC (Remote Procedure Call), TP мониторы (Transaction Proceesing) обработки транзакций, Groupware - управление различными неструктурированными данными (тексты, факсы, письма электронной почты, календари и т.д.) и, наконец, MOM - Message-Oriented Middleware (асинхронный обмен сообщениями между сервером и клиентом), к созданию распределенных компьютерных систем. Элементы этих систем могут взаимодействовать друг с другом как на одной локальной машине, так и по сети. Уникальная полифоничность CORBA позволяет организовать единую информационную среду, элементы которой могут общаться друг с другом, вне зависимости от их конкретной реализации, "прописки" в распределенной системе, платформы и языка их реализации.

В стандарте CORBA звучат две основные темы:

1. *Точка отсчета (развития).* В отличие от стандартов MS COM/DCOM (Component Object Model / Distributed COM), которые были созданы для объединения мелких офисных программ, CORBA возник в ответ на необходимость интеграции промышленных приложений.

2. *Объектность идеологии.* CORBA - стандарт для объединения объектов.

Чтобы разобраться в объектно - ориентированной партитуре CORBA, отвлечемся на краткий курс объектной теории.

Классический объект - самостоятельная часть кода для компилятора. В общем случае неизвестен и недоступен вне данной программы.

Распределенный объект или компонент - объект, который может существовать независимо от данной программы, в том числе на сети. Это самостоятельная, отдельно скомпилированная часть кода.

Бизнес-объект - распределенный объект, выполняющий ту или иную бизнес-функцию, иначе говоря, законченную производственную, финансовую, административную, информационную операцию.

Основа объектной технологии - 3 главных свойства объекта: наследование, т. е. возможность строить дерево классов с наследованием методов и структур данных; полиморфизм, когда один и тот же метод (операция или функция) может приводить к разным результатам на различных классах; инкапсуляция - реализация метода происходит

внутри объекта, для других объектов доступен только интерфейс, по которому они связываются с данным объектом.

OMG сформулировала единый семантический стандарт для своих членов - **Объектную Модель**, в которую входит 4 основных понятия:

1. объекты, моделирующие окружающий мир (человек, лодка, документ);
2. операции, относящиеся к объекту и описывающие его особенности и специфику поведения (дата рождения для объекта "человек", материал, из которого сделана лодка);
3. типы, описывающие конкретные значения операций на объекте (деревянная лодка, длиной 2 метра, с двумя сиденьями, без мотора и т.д.);
4. подтипы, уточнения типов (лодка, максимальная скорость которой 10км/час).

На основе этих понятий OMG определила набор собственных объектных понятий - Объектную Модель, спецификацию для развития стандарта CORBA. Как и все части CORBA, Объектная Модель постоянно изменяется, отражая диалектику окружающей реальности.

Как показывает практика, постоянное совершенствование заложено в самом стиле, выработанном OMG для развития CORBA.

11.1 История создания OMG и стандарта CORBA

В 1989 г. несколько компаний, поставщиков и потребителей компьютерных

технологий, устав от неразберихи в промышленных приложениях, образовали OMG. Цель новой некоммерческой организации звучала в лучших традициях философских утопий: "Объединить мир". Переходя от философии к реальной жизни, это означает: объединить, путем создания средств интеграции приложений, мир прикладных программ, с их комплексами и системами, прежде всего в области автоматизации различных отраслей промышленности.

В названии группы уже был скрыт ключ к решению поставленной задачи: OMG определяет Object Management (Объектное Управление) как создание программного обеспечения, которое через понятие объекта моделирует реальный мир. Объектная технология - великолепное средство разработки промежуточного ПО. Ее главное достоинство, способность расширять функциональность и добавлять новые компоненты в систему без изменения существующей структуры, позволяет легко строить гибкие, самоуправляемые, масштабируемые распределенные системы. С другой стороны, именно с развитием объектных методов возникла необходимость конструирования промежуточного ПО нового типа - не раз навсегда установленного моста между компонентами, а универсальной среды их взаимодействия.

OMG с самого начала объявила себя демократической организацией, а выработанные стандарты - бесплатными и открытыми для дополнений и изменений. Члены OMG разработали необыкновенно интересную процедуру создания новых

стандартов, основанную на понятии Request For Proposal (RFP - запрос на разработку). RFP выпускается специальным комитетом OMG - Task Force и представляет собой адресованный членам OMG подробный запрос на развитие какого - либо конкретного стандарта. Task Force формирует запросы на основании информации, поступающей как от членов OMG, так и от независимых компаний и частных лиц. Запрос на RFP должен быть обоснован реальными потребностями существующих или разрабатываемых продуктов. Через 3 недели после публикации проекта нового RFP (это время дается членам OMG на обдумывание задачи) происходит обсуждение запроса и определяется график выпуска нового стандарта. После создания новых спецификаций члены OMG голосуют за принятие нового стандарта и включение его в структуру CORBA. Обычно процесс разработки нового стандарта занимает около года. Сейчас актуальны, например:

1. RFP о компонентной модели CORBA - спецификации для интерфейсов и механизмов распределенной компонентной модели, основанной на CORBA, которые способны взаимодействовать с другими компонентными технологиями;
2. RFP о специальном языке сценариев для облегчения работы с компонентами CORBA;
3. RFP для управления документами медицинского страхования и бухгалтерских расчетов в медицине, передаваемыми по сети.

Очевидно, что при таком подходе темпы развития CORBA стремительно растут, ведь чем больше компаний используют CORBA совместимые продукты, тем больше выпускается RFP и тем быстрее развивается стандарт.

Сегодня в OMG входят более 800 компаний, среди которых: Acer, Cisco, HP, American Airlines, Hitachi, IBM, Siemens, Microsoft Sun, Sybase, Boeing, EDS, Ericsson, Netscape, Nokia, Ford Motor, Oracle и ряд других. Большинство крупных компаний, имеющих отношение к информационным технологиям, входят в OMG. Корпорация Microsoft долго не присоединялась к OMG - развивала собственный стандарт, COM/DCOM. Сегодня битва OMG - Microsoft на поле промежуточного ПО завершилась, наконец, мирными переговорами. Разработаны специальные средства, которые позволяют приложениям, поддерживающим один стандарт, взаимодействовать с приложениями из другого лагеря. DCOM присущи все недостатки стандарта, разрабатываемого одной компанией: он сконцентрирован на Windows и Microsoft не портируется на другие платформы; кроме того, DCOM проигрывает и по некоторым другим позициям.

OMG работает в тесном контакте с другими центрами стандартизации: ISO, Open Group (X/Open), WWW консорциум, ANSI, IEEE и многими другими. Как утверждает президент OMG Вильям Хоффман, в 1997 г. CORBA стал неотъемлемой частью жизни распределенных объектных компьютерных систем. Окончательная ли это победа? Будем осторожны. Ведь в данном случае говорить о полной интеграции приложений можно

только если их "общение" столь же естественно, как телефонный разговор. До этого еще далеко.

Первый итоговый документ OMG был опубликован в 1991 г., это OMA (Object Management Architecture) Guide - путеводитель по архитектуре объектного управления, описывающий ядро CORBA. В 1992 г. вышел его переработанный вариант, а в 1994 г. появился CORBA 2.0. Именно с этого момента стало очевидно, что стандарт скорее жив, чем мертв и сейчас он в превосходной форме. Стандарт CORBA состоит из 4 основных частей:

1. Object Request Broker - брокер объектных запросов;
2. Object Services - объектные сервисы;
3. Common Facilities - общие средства;
4. Application и Domain Interfaces - прикладные и отраслевые интерфейсы.

11.2 Брокер (посредник) объектных запросов ORB (Object Request Broker)

Обобщенная Архитектура построения Брокеров Объектных Запросов разработана для поддержки интеграции самых разнообразных объектных систем. Спецификация CORBA устанавливает п**ринципы создания Брокеров Объектных Запросов**, которые и допускают такую интеграцию.

Запрос посылается от клиента к серверу. *Клиент* это приложение, или нечто другое, выполняющее операцию над объектом, а *реализация объекта* - это код и данные, которые

на самом деле выполняют эту операцию. ORB способен выполнить все действия, необходимые для нахождения реализации указанного объекта, подготовке этой реализации к обработке запроса и передаче данных, относящихся к запросу. Интерфейс, предоставляемый клиенту абсолютно не зависит от местоположения реализации объекта, языка программирования, на котором он написан или каких-либо других аспектов, не влияющих на определение интерфейса для данного объекта.

При определении конкретной архитектуры Брокер Объектных Запросов вовсе необязательно должен быть реализован как один компонент, но каждая реализация должна реализовывать три категории операций:

- Операции, которые одинаковы для всех реализаций ORB-а.
- Операции, специфичные для конкретного объектного типа.
- Операции, специфичные для отдельных видов реализаций объектов.

Различные реализации ORB-а могут поддерживать различные виды реализаций, а различные адаптеры объектов могут обеспечивать различные наборы сервисов для клиента и реализаций.

Ядро Брокера Объектных Запросов обеспечивает основные механизмы для манипуляций объектами и выполнения запросов. Спецификация CORBA предназначается для поддержки различных механизмов реализации объектов, поэтому структура ядра не определяется. Вместо этого задается набор интерфейсных функций, которые должны

присутствовать в каждой реализации ORB-а и тем самым маскируют отличия между различными реализациями Брокеров Объектных Запросов.

Объекты

Система объектов обеспечивает клиента набором сервисов. Клиент способен запросить некоторый сервис. Объект - это нечто, что обеспечивает один или более сервисов, которые клиент может запросить.

Пример Брокеров Объектных Запросов

Доступно широкое множество способов реализации конкретных ORB-ов. Далее будут приведены примеры таких реализаций. Следует иметь ввиду, что конкретный ORB может быть реализован сразу несколькими способами.

ORB, включаемый в клиентское и серверное приложение

Если имеется подходящий механизм коммуникаций, то возможна реализация ORB-а в виде набора подпрограмм как со стороны клиента, так и со стороны реализации объекта. Вызовы методов могут транслироваться в работу со средствами взаимодействия процессов (Inter Process Communication - IPC).

ORB, выполненный в виде сервера

С целью обеспечения централизованного сбора и управления всевозможной информацией, ORB может быть реализован в виде отдельного приложения. Взаимодействующие приложения устанавливают контакт с ORB-ом посредством

нормальных механизмов IPC.

ORB как часть системы

Для повышения надежности, защиты данных и достижения лучшей производительности ORB может быть реализован как часть операционной системы. При этом ссылки на объект могут быть сделаны постоянными, таким образом, уменьшая время, необходимое для обработки каждого запроса. При реализации ORB-а как части операционной системы возможны всевозможные виды оптимизации, такие как избежание кодирования и декодирования данных, если клиент и сервер находятся на одной и той же машине.

ORB, основанный на библиотеках

Если код объекта занимает небольшой объем и не требует никаких дополнительных средств, то он может быть выполнен в виде библиотеки. При этом все заглушки на самом деле будут являться настоящими методами. При этом предполагается, что имея доступ к данным реализации, клиент не разрушит эти данные.

Реализации объектов

Реализация объекта обеспечивает само понятие объекта, обычно задавая данные для конкретного экземпляра объекта и код для выполнения методов объекта. Часто реализация будет использовать другие объекты или вспомогательные программы для обеспечения функционирования объектов. В некоторых случаях выполнение операции

над объектом влечет некие побочные действия не над объектами.

Конкретный ORB может поддерживать широкий набор объектных реализаций: отдельные серверы, библиотеки, объектно-ориентированные системы управления базами данных и др. С помощью использования дополнительных Адаптеров Объектов теоретически можно поддерживать любую реализацию объекта.

Адаптеры объектов

Адаптер объектов - это первичный путь для обеспечения сервиса конкретной реализацией объекта. Предполагается, что имеется несколько адаптеров объектов, каждый из которых обеспечивает доступ к объектам определенного вида.

Сервисы, которые обеспечиваются ORB-ом посредством адаптеров объектов часто включают: генерацию и интерпретацию ссылок на объекты, вызов методов, активацию и деактивацию реализаций объектов, а также регистрацию конкретных реализаций и отображение объектных ссылок и реализаций.

Информация, которая находится в каждом из хранилищ может быть произвольно изменена в любой момент времени с помощью методов, обеспечиваемых реализацией ORB-а. Однако, неосторожное изменение, сделанное во время работы может привести нарушить целостность информации, находящейся в каждом из хранилищ и сделать невозможным дальнейшее функционирование ORB-а.

Скелет реализации

Для конкретного отображения и, возможно, используемого адаптера объектов определяется свой порядок вызова методов каждого объекта. Этот интерфейс в общем случае является интерфейсов обратных вызовов. При необходимости ORB вызывает требуемые процедуры.

Динамическая обработка запросов

Также доступен интерфейс для динамической обработки поступающих запросов. В этом случае реализация объекта взаимодействует с заданным интерфейсом по аналогии с интерфейсом динамических вызовов.

Подпрограммы динамической отработки запросов могут вызываться как с помощью интерфейса динамических вызовов, так и с помощью процедур-заглушек, каждый метод дает одинаковый результат.

Запросы

Клиент запрашивает сервисы инициированием запросов. Запрос - это событие, то есть действие, происходящее в конкретный момент. С запросом связана информация, состоящая из операции, объекта, у которого запрашивается сервис, нуля или более действительных параметров вызова и необязательный контекст запроса. Форма запроса - это описание или шаблон, который может быть выполнен произвольное количество раз. Форма запроса определяется отображением для конкретного языка программирования. Альтернативной формой запроса является использования Интерфейса Динамических

Вызовов, который позволяет создать запрос, добавить аргументы и выполнить запрос. Под значением понимается допустимый параметр запроса. Значение которое определяет объект, называется ссылкой на объект, связанной с конкретным экземпляром объекта. Выполнение запроса вызывает выполнение соответствующего сервиса. После завершения запроса клиенту возвращается результат запроса (если он есть). В случае ненормального завершения запроса клиенту возвращается исключение. Исключение может содержать специфические параметры, специфические для данного типа исключений.

Параметры

Параметр характеризуется режимом передачи и своим типом. Режим определяет, должно ли передаваться значение параметра от клиента к серверу (in), от сервера к клиенту (out) или в обоих направлениях (inout).

- Возвращаемое значение.
- Если есть возвращаемое значение, то оно рассматривается как параметр типа out.
- Исключения.
- Исключение свидетельствует о том, что операция не была успешно выполнена. Исключение может содержать дополнительную информацию, специфичную для конкретного исключения.
- Контекст.
- Контекст запроса обеспечивает передачу дополнительной, специфичной для

операции информации, которая может повлиять на выполнение запроса.

Параметры запросов определяются их позицией. Параметры могут быть входные, выходные и входные и выходные одновременно. Как результат запрос может возвратить одно значение, как, впрочем, и любые выходные параметры. В случае возникновения исключения значение всех выходных параметров не определено.

Интерфейсы

Интерфейс - это описание множества возможных операций, которые клиент может выполнять над объектом. Объект удовлетворяет интерфейсу, если он может быть указан как конечным объектом для каждого потенциального запроса, описанного в интерфейсе. Типу интерфейса удовлетворяют только объектные типы.

Интерфейс ORB-а

Интерфейс ORB-а является функциям, вызываемым непосредственно у Брокера Объектных Запросов и идентичным для всех ORB-ов, не зависящим от конкретного объекта либо адаптера объектов. Но так как большинство действий с объектами выполняется посредством адаптеров объектов, существует всего несколько общих операций, которые могут быть выполнены над каждым объектом. Эти операции могут вызываться как клиентом, так и реализацией объекта.

11.3 IDL (Interface Definition Language - язык определения интерфейсов)

Язык описания интерфейсов (IDL), используемый OMG определяет типы объектов

посредством спецификации их интерфейсов. Интерфейс состоит из множества именованных операций и их параметров.

Язык описания интерфейсов рассматривается как средство, с помощью которого реализация объекта сообщает своим потенциальным клиентам о том, какие операции доступны и каким образом их следует вызывать. Можно оттранслировать описание на языке IDL в исходный код на конкретном языке программирования.

Отображение IDL в языки программирования

Различные объектно-ориентированные или объектно-неориентированные языки программирования могут получать доступ к объектам различным образом. Для объектно-ориентированных языков допускается отображение объектов, доступных ORB-у в объекты в смысле этих языков программирования. Даже для объектно-неориентированных языков декорирование настоящего представления ссылок на объекты будет полезным. Конкретное отображение IDL для языка программирования должно быть идентичным для всех реализаций ORB-ов. Отображение для языка программирования включает в себя определение специфичных для языка программирования типов данных и описания процедур доступа к объектам посредством ORB-а. Оно также включает в себя интерфейс для доступа клиента к заглушке, что может не требоваться для объектно-ориентированных языков, интерфейс динамических вызовов, скелет реализации, описание адаптеров объектов и прямой интерфейс к ORB-у.

Отображение для языка также определяет порядок взаимодействия между вызовом метода и потоками (тредами - threads) как со стороны клиента, так и со стороны реализации. Обычно обеспечивается синхронный вызов, в котором подпрограмма вызова возвращает управление при завершении выполнения запроса. Допускается определение дополнительных средств, для определения порядка передачи управления и синхронизации клиентского кода с вызовом метода объекта.

Типы данных

Типом называется некий предикат (математическая функция с одним аргументом возвращающее значение логического типа истина/ложь), который определен на множестве всевозможных значений. Значения удовлетворяют этому типу, если результат предиката - истина. Такие значения называются членами типа.

Объектным типом называется тип, членами которого являются объекты, удовлетворяющие данному типу.

Определены следующие основные (базовые) типы данных:

1. 16 и 32 разрядные знаковые и беззнаковые целые типы;
2. 32 и 64 разрядные типы с плавающей точкой в соответствии с IEEE;
3. Символьный тип в соответствии с ISO Latin-1 (8859.1);
4. Логический тип с множеством значений истина и ложь;
5. 8 разрядный тип, который гарантированно не подвергается никаким изменениям

при передаче между различными системами;

6. Перечислимые типы, состоящие из последовательности идентификаторов;
7. Строковый тип, состоящий из последовательности символов переменной длины, длина строки доступна во время выполнения программы;
8. Тип "any", который может принимать значения всех базовых и составных типов.

Также могут быть определены составные типы:

1. структура, состоящая из упорядоченных пар (имя, значение);
2. объединение, состоящее из дискриминатора и значения типа, связанного с дискриминатором;
3. последовательность, которая является массивом переменной длины значений одного типа, длина последовательности доступна во время выполнения;
4. массив фиксированной длины, элементами которого являются значения одного типа;
5. тип интерфейс, который определяет множество операций, которое должен поддерживать экземпляр этого типа.

Параметры, представленные в запросе должны удовлетворять одному из перечисленных типов, за исключением типа интерфейс, как показано на рисунке 2-1.

Синтаксис Общего Представления Данных - CDR

CDR - это способ представления всех типов данных, определенных в OMG IDL в

виде последовательности восьмиразрядных величин, далее называемых байтами.

Поток байт представляет из себя некоторую абстракцию обычно соответствующую буферу данных, который передается между процессами или машинами с помощью средств IPC или сетевого транспорта. Далее считается, что поток байт или просто поток - это последовательность переменной (но конечной) длины величин, состоящих из 8 бит (байт) с четко определенным заголовком. Байты в потоке нумеруются от *0* до *n-1*, где *n* - это длина потока. Индекс каждого байта используется для вычисления границ выравнивания, как это описано далее.

Протокол GIOP определяет два вида потоков - сообщение и инкапсуляция. Сообщение - это основная единица обмена информацией в протоколе GIOP. Инкапсуляция - это поток, внутри которого любая структура данных, имеющаяся в OMG IDL может быть декодирована независимо от остального контекста сообщения. Инкапсуляция позволяет осуществлять предварительное кодирование сложных типов данных (таких как TypeCode) или обрабатывать части сообщений без требования полного его декодирования.

Кодирование базовых типов

Все базовые типы могут быть закодированы как набор байт. При этом допускается использование как представления, в котором первым в поток помещается наиболее значимый или наименее значимый байт. Заголовок сообщения включает в себя флаг,

который определяет порядок при кодировании базовых типов в сообщении. Порядок байт внутри любой инкапсуляции может отличаться от порядка байт в сообщении или другой инкапсуляции, внутри которой эта инкапсуляция находится. Изменение порядка байт в случае необходимости возлагается на получателя сообщения.

Для того, чтобы сделать возможным помещение и извлечение значений базовых типов в поток и из потока с помощью подпрограмм, предназначенных для работы именно с этими типами данных, все базовые типы данных при помещении в поток должны быть выровнены на свою естественную границу, то есть на число байт, которое нацело делится на число байт, необходимых для представления этого типа. Таким образом значение, имеющее размер *n* должно кодироваться с позиции *m*n*, где *m* - это целое число. В CDR *n* может принимать значения 1, 2, 4 или 8. Если необходимо, то выровненному значению предшествует область минимально возможного размера, необходимого для выравнивания. Значение байтов внутри этой области не определено.

Выравнивание определяется относительно начала потока. Первый байт в сообщении или инкапсуляции имеет индекс 0.

Кодирование составных типов

Выравнивание составных типов не налагает никаких дополнительных требований, кроме тех, которые применяются при кодировании их элементов.

Элементы структуры кодируются в том порядке, в котором они определены в

описании на IDL. Каждый элемент кодируется образом, соответствующим его типу.
Объединение кодируется значением дискриминатора и членом объединения, соответствующим данному значению.

Массив кодируется как последовательность его элементов. Так как длина массива фиксирована, она не кодируется. Если массив имеет несколько измерений, то первый индекс изменяется наиболее медленно, а последний - наиболее быстро.
Последовательность элементов кодируется как величина типа unsigned long, за которым следуют элементы последовательности. Это значение определяет количество элементов. Каждый элемент кодируется в соответствии со своим типом.

Строка кодируется как величина типа unsigned long, содержащее длину строки, и отдельными символами - элементами строки. Длина строки и ее представление в виде списка символов включают завершающий нулевой символ, что дает возможность использования стандартных функций библиотеки языка C (например, strcpy) для декодирования сообщения.
Значение перечислимого типа кодируется в виде величины типа unsigned long, соответствующей данному значению. Первому в порядке перечисления в определении на IDL значению соответствует 0, второму - 1 и так далее.

Кодирование инкапсуляции

Первый байт инкапсуляции кодирует порядок байт внутри нее - значение типа 0

означает кодирования по принципу первым - старший байт, 1 - младший. Далее идут данные. Флаг порядка байт не включается в данные, но он включается в инкапсуляцию. Все значения внутри инкапсуляции выравниваются относительно ее начала, первый байт (с индексом 0) соответственно занимает флаг порядка байт. Если инкапсуляция кодируется как последовательность величин типа octet (байтов), то ей предшествует значение типа unsigned long, содержащее общий размер инкапсуляции. Никакого выравнивания для инкапсуляции не предполагается, но такой способ кодирования всегда гарантирует 4-байтное выравнивание для первого байта инкапсуляции.

Кодирование псевдообъектов

Спецификация CORBA определяет несколько псевдообъектов, которые не являются ни базовыми ни составными типами и кодируются специальным образом. Ввиду особой специфичности данного кодирования и оно здесь не рассматривается.

Операции

Операция представляет сервис, выполнение которого может быть запрошено. Операция определяется идентификатором операции. Операция описывается некоторой сигнатурой, которая задает параметры запроса и возвращаемое значение. В частности сигнатура состоит из:

1. спецификации параметров, требуемых для выполнения операции
2. спецификации возвращаемого значения

3. спецификации исключения, которые могут возникнуть во время выполнения операции и типов данных, которые соответствуют этим исключениям
4. спецификации дополнительной контекстной информации, которая может повлиять на выполнение запроса
5. индикации семантики, которую клиент должен учитывать при выполнении операции.

Хранилище описаний

Хранилище описаний представляет из себя сервис, который обеспечивается постоянным объектом, доступном из программы. Во время выполнения программы он дает доступ к информации, аналогичной той, что сохраняется в IDL описании объекта. Эта информация может быть использована для выполнения запроса - таким образом программа, которая не предусматривала использование объекта какого-либо типа, определить доступные у этого типа методы, типы его параметров и осуществить вызов.

11.4 Object Services - объектные сервисы

Само по себе ORB обеспечивает чистое взаимодействие объектов друг с другом, телефонные линии распределенной информационной среды. Сервисы, написанные на IDL, поставляют объектам дополнительные интерфейсы. Благодаря сервисам разработка прикладных программ упрощается едва ли не до уровня все той же игры с конструктором. В реальной системе не обязательно должны присутствовать все сервисы, их набор зависит

от требуемой функциональности. На сегодня разработано всего *14 объектных сервисов*:

1. Naming, сервис наименований, позволяет компонентам системы по ссылкам легко находить друг друга в распределенной среде.
2. Events, сервис событий, определяет методику работы с событиями. Объект может динамически подписаться на интересующее его событие другого объекта и получить соответствующее извещение, если это событие произошло. Сервис создает специальный объект (канал событий), который собирает и распределяет уведомления о происшедших событиях среди заинтересованных объектов.
3. Security, сервис безопасности, ограничивает права доступа в распределенной среде.
4. Transactions, сервис транзакций, при работе с базами данных обеспечивает подтверждение или "roll back" всех изменений, сделанных по ORB.
5. Trading, сервис коммерции, альтернатива сервису наименований. Вместо ссылки на нужный объект по имени дает возможность объекту-отправителю выбрать группу объектов заказанного типа.
6. Life Cycle, сервис функциональности разрешает создавать, копировать, перемещать и уничтожать компоненты по ORB.
7. Externalization, сервис импорта / экспорта - с его помощью внутренние (internal) параметры объекта сохраняются в текстовом файле, из которого затем можно скопировать или восстановить объект с теми же параметрами.

8. Licensing, сервис лицензирования, защищает от несанкционированного использования программных компонентов в распределенной среде.

9. Time, сервис времени - набор интерфейсов для установки текущего времени и операций с временными интервалами. Дает возможность оперировать с событиями во времени.

10. Property, сервис свойств, используется для приписывания свойств объектам. Это нужно в тех случаях, когда само приложение не может приписать объекту распределенной среды необходимые свойства через механизм атрибутов и операций, не затрагивая IDL определения.

11. Relationships, сервис отношений, с его помощью можно связать два типа объектов через IDL определения.

12. Concurrency Control, сервис согласования, предохраняет систему от разрушения в случаях совместного использования одних и тех же данных. В такой ситуации сервис определяет способы блокировки данных.

13. Persistent Objects, сервис надежного хранения - для особо важных данных, которые надо хранить в защищенных структурах, таких как БД или специальные файловые системы. Интерфейсы сервиса могут выполняться, например, над реляционной базой данных, включенной в распределенную систему.

14. Query, сервис запросов, определяет тип простого набора данных и способы

выполнения запросов (query) к объектам такого набора. Поддерживает большинство query языков и прежде всего SQL. Соответствует SQL3 спецификациям.

Компоненты ORB, развиваясь вместе с объектными сервисами, постепенно превращаются в суперкомпоненты. Суперкомпонент - это компонент с высшим образованием. Пройденные дисциплины: безопасность, лицензирование, поддержка версий, самоуправление функциональностью, т. е. создание, клонирование, передвижение с места на место в распределенной среде, разрушение и архивирование, обработка событий, управление транзакциями и блокировками, выживание (сохранение состояния и восстановление из сохраненного), взаимодействие с другими компонентами, открытость (компоненты должны по запросу обеспечивать информацию о самих себе), самотестирование, самоинсталляция.

11.5 Common Facilities - общие средства

Между объектными сервисами и общими средствами CORBA нет четкой границы. Так, лицензирование вполне могло бы относиться к общим средствам. Чтобы пояснить, что такое общие средства, проще всего перечислить те из них, которые уже включены в стандарт.

USER Interface - представление объектов и сложных документов. Сюда входят средства работы с подсказками, проверка правописания и грамматики, управление рабочим полем (desktop). Система OpenDoc (совместная разработка группы компаний,

среди которых и IBM) может служить хорошим примером использования USER Interface.

Information management - моделирование информации, ее сохранение и восстановление, кодирование и перевод, поддержка времени и календаря. System management - управление ORB и CORBA приложениями. Для этой спецификации OMG использовала стандарт X/Open. Task management - контроль выполнения, отслеживание агентов.

Все перечисленные интерфейсы представляют собой горизонтальные средства, общие для всех доменов. Домен в лексике CORBA - отрасль промышленности. Это одно из ключевых понятий, ведь основная задача OMG - объединение именно промышленных приложений. Нетрудно заметить, что промышленные приложения сильно зависят от предмета, который призваны автоматизировать. Поэтому кроме общих горизонтальных средств выделились вертикальные общие средства по доменам. Сейчас они определены по следующим направлениям: телекоммуникация, финансы, производство, медицина, транспорт, электронная коммерция, бизнес-объекты. Этот список постоянно пополняется: выпускаются новые RFP, по ним разрабатываются новые стандарты, которые относятся к Object Services или Common Facilities, к Application или Domain Interfaces.

11.6 Достоинства CORBA

1. Язык IDL поддерживает разнообразные программные языки, операционные системы, сети и объектные системы. IDL позволяет отделить описание интерфейса

от его реализации. Таким образом, можно менять объекты, не затрагивая интерфейсы. Приложение, даже если оно написано не на объектно-ориентированным языке, с помощью IDL можно инкапсулировать в объектную структуру.

2. CORBA - сетевая архитектура по определению, эта идея лежит в основе его развития. Объектно-ориентированные интерфейсы CORBA легко определять, создавать и использовать.
3. Каждый сервер может содержать много объектов. Связь между отправителем и адресатом осуществляется напрямую. Объекты могут быть разных размеров.
4. CORBA хорошо сочетается с разнообразным промежуточным ПО, включая OLE языки (например, для реализации интерфейса можно использовать VisualBatch).
5. В рамках CORBA можно обеспечить необходимый уровень безопасности системы.
6. Интеграция с другими распространенными технологиями: базами данных, системами обработки сообщений, системами обработки пользовательского интерфейса и другими. Специализация по отраслям промышленности открывает дополнительные возможности для приближения объектов к реальным структурам.
7. Существует протокол IIOP, который позволяет взаимодействовать различным ORB по TCP/IP. CORBA сервисы обеспечивают ряд дополнительных возможностей: транзакции, события, query и т. д. Одновременная поддержка статических и

динамических интерфейсов. Возможность включения в распределенную среду Web-клиентов и серверов, в частности, через Java-реализации CORBA.

8. CORBA - широко используемый стандарт, со множеством реализаций, но создается и поддерживается он централизованно, OMG.
9. Так как CORBA - только стандарт, между его реализациями естественное возникает соревнование. Тем самым повышается качество продуктов, а их совместимость заложена в стандарте.
10. По мере развития CORBA процесс создания программных приложений все больше напоминает конструирование из готовых деталей.

11.7 Обзор протоколов GIOP и IIOP

Спецификация протокола GIOP состоит из следующих элементов:

1. Определение Общего Представления Данных (*Common Data Representation - CDR*). CDR - это способ кодирования типов данных, определенных в IDL в низкоуровневое представление, пригодное для передачи их по имеющимся каналам связи между ORB-ами.
2. Формат сообщения протокола GIOP. Сообщения протокола GIOP обеспечивают нахождение объекта, отработку запросов, а также простейшее управление каналом коммуникации.
3. Предположения о транспорте. Спецификация GIOP описывает общие

предположения, которые делаются при рассмотрении любого сетевого транспортного слоя, который может быть использован для обмена сообщениями протокола GIOP. Также описываются общие принципы управления соединением.

Спецификация IIOP добавляет к спецификации протокола GIOP следующий пункт:

4. Транспорт для сообщений протокола IIOP.

Спецификация IIOP описывает, каким образом агенты могут установить соединение по протоколу TCP/IP и использовать его для передачи сообщений протокола GIOP.

Протокол IIOP не является самостоятельной спецификацией - это специализированное отображение протокола GIOP поверх транспортного слоя TCP/IP. Спецификация GIOP (без элементов, специфичных для IIOP) может рассматриваться как самостоятельный документ, являющийся базовым для обеспечения в будущем отображения на новые транспортные протоколы.

Протокол обмена GIOP

За исключением редкого случая прямых вызовов методов между классами одного и того же языка программирования необходим механизм кодирования вызова метода в некоторую последовательность байт (*byte stream*) у клиента и декодирования этой последовательности у сервера. Для этой цели спецификация CORBA определяет Общий Протокол обмена между Брокерами Объектных Запросов (*General Inter-Orb Protocol - GIOP*). Кроме того, определен протокол передачи сообщений протокола GIOP поверх

транспортного протокола TCP/IP, являющегося основным видом взаимодействия в Internet, ввиду чего этот протокол получил название Протокола обмена между Брокерами Объектных в Internet (*Internet Inter-Orb Protocol - IIOP*). Протокол IIOP должен поддерживаться всеми Брокерами Объектных Запросов независимо от особенностей их реализации, что является главным требованием для обеспечения взаимодействия между произвольными ORB-ами двух разных и совершенно независимых производителей.

Особенности и цели протокола

Протоколы GIOP и IIOP допускают взаимодействие между различными ORB-ами независимо от платформ, на которых они выполняются, операционных систем, под управлением которых происходит взаимодействие и прочих аппаратно- и программно-зависимых аспектов. **При разработке этих протоколов преследовались следующие цели:**

1. Распространенность

Протоколы GIOP и IIOP разрабатывались с учетом доступного широко распространенного и гибкого транспортного механизма (TCP/IP) и задает минимум дополнительных протоколов, необходимых для передачи запросов между отдельными ORB-ами.

2. Простота

Помимо прочих требований, протокол GIOP сделан максимально простым. Его

простота допускает возможность реализации взаимодействия по этому протоколу практически в любой системе.

3. Масштабируемость

Протокол GIOP/IIOP должен поддерживаться как отдельными ORB-ами, так и ORB-ами, объединенными в сеть на уровне Internet и, возможно, шире.

4. Небольшие затраты на реализацию

Реализация поддержки протоколов GIOP/IIOP должна потребовать минимальных затрат как в плане инженерного проектирования, так в плане распространения готовых ORB-ов.

5. Общность

В то время как IIOP изначально определен поверх протокола TCP/IP, сообщения, которыми происходит обмен в рамках протокола GIOP специально разработаны для реализации поверх любого протокола, который базируется на установленном между сервером и клиентом соединении.

6. Архитектурная независимость

Спецификация GIOP делает минимальные предположения об архитектуре агентов, которые поддерживают обмен данными по этому протоколу. Спецификация GIOP считает ORB некой системой с неизвестной архитектурой.

Подход конкретного ORB-а к обеспечению поддержки протокола GIOP/IIOP не

определен. Например, ORB может принять IIOP в качестве внутреннего протокола, использовать его только для внешнего обмена, используя для обмена в рамках самого ORB-а какие-то дополнительные средства коммуникации или выбрать нечто среднее между этими двумя крайностями. Все что требуется от ORB-а - это чтобы существовало нечто способное принимать и отправлять сообщения по протоколу IIOP.

Формат сообщений протокола GIOP

Перед тем, как описывать сообщения протокола GIOP, необходимо определить понятие клиента и сервера. Под клиентом далее понимается агент, который открыл соединение и инициировал запрос. Сервер - это агент, который принял соединение и этот запрос получил. Протокол GIOP определяет семь сообщений, список которых приведен далее в таблице вместе с указанием того, какая сторона какие сообщения может посылать.

Значение, соответствующее типу сообщения	Тип сообщения	Кто может посылать сообщение	
		Клиент	Сервер
0	Request	Да	-
1	Reply	-	Да
2	CancelRequest	Да	-
3	LocateRequest	Да	-
4	LocateReply	-	Да
5	CloseConnection	-	Да
6	MessageError	Да	Да

Заголовок сообщения однозначно определяет его тип. Заголовок определен таким

образом, чтобы не зависеть от порядка байт в представлении базовых типов данных. Элементами заголовка являются:

1. Поле magic, которое состоит из четырех символов "GIOP", идентифицирующих все сообщения протокола GIOP.
2. Поле GIOP_version, которое состоит из двух полей major и minor, идентифицирующих старший и младший номера версии используемого протокола. Текущая спецификация определяет версию 1.0. Приложение должно поддерживать взаимодействие в рамках протокола только если номер, содержащийся в поле major равен, а в поле minor - больше или равен номерам версии, используемой при разработке приложения.
3. Поле byte_order. Значение 0 в этом поле определяет, что в сообщении принято кодирование данных с лидирующим наиболее значащим байтом, 1 - наименее значащим. В настоящее время подавляющее большинство процессоров, в том числе и серия Intel x86 используется представление с лидирующим наименее значащим байтом.
4. Поле message_type содержит значение от 0 до 6, определяющее тип сообщения.
5. Поле message_size содержит длину оставшейся части сообщения (0 если больше ничего нет).

За общим заголовком каждого сообщения в зависимости от его типа может идти

заголовок и тело конкретного сообщения. Структура каждого заголовка специфична для каждого типа сообщения и представляет особенного интереса для рассмотрения.

Транспорт для протокола GIOP

Протокол GIOP предназначается для реализации поверх большого количества транспортных протоколов. При этом делаются следующие предположения об особенности протокола:

1. Транспорт ориентируется на установление соединения с последующим обменом информации в рамках соединения. Соединение используется для определения правил нумерации запросов.
2. Транспорт протокол должен гарантировать прохождение переданных байт в том порядке, в котором они были посланы.
3. Транспорт может рассматриваться как поток байт без дополнительных ограничений на размеры, фрагментацию или выравнивание размеров посылок.
4. Транспорт должен обеспечивать сигнализацию об разрыве соединения. Если один из участников обмена неожиданно прервал свою работу, произошел сбой в операционной системе или сети, то другой должен быть уведомлен об этом.

Сервер не инициирует установление соединения, но он выполняет некоторые действия по подготовке к принятию запросов от клиентов. Клиент знает местоположение (адрес) сервера и устанавливает соединение по указанному адресу. Сервер может принять

соединение, уникальное для каждого клиента (при этом продолжив ожидание новых запросов), а может и не принять, например вследствие недостатка ресурсов. Если соединение установлено, то по данному каналу может происходить двусторонний обмен информацией, причем каждая стороны имеет возможность в произвольный момент времени разорвать соединение.

Вовсе не обязательно конкретный транспорт должен прямо реализовывать все перечисленные требования, но должна иметься возможность для эмулирования описанной транспортной модели.

Управление соединением

Соединение двунаправленное в смысле потока данных, но оно не является симметричным в плане обмена сообщениями GIOP. Сообщения *Request*, *LocateRequest* и *CancelRequest* могут посылаться только клиентом. Сообщения *Reply*, *LocateReply* и *CloseConnection* - только сервером. Сообщение *ErrorMessage* может быть послано обеими сторонами. Через соединение для обмена в соответствии с протоколом GIOP могут посылаться только сообщения GIOP.

Каждое сообщение типа *Request* должно иметь уникальный номер, который идентифицирует запрос в рамках установленного соединения. Этот номер никоим образом не интерпретируется сервером, но он позволяет клиенту установить соответствие между запросом и пришедшим ответным сообщением в случае инициации сразу

нескольких запросов. Генерация этих уникальных номеров возлагается на клиента.

Соединение может быть либо закрыто в рамках протокола, либо разорваться. Закрытие соединения может инициироваться со стороны сервера посредством посылки сообщения *CloseConnection* или клиентом посредством обычного закрытия соединения в произвольный момент времени. Если на момент закрытия соединения имеются неотработанные запросы, то сервер должен рассматривать такие запросы как отмененные. Сервер не может послать сообщение *CloseConnection*, если он начал обработку запроса, для которого не поступило сообщения *CancelRequest*, или он (сервер) не послал ответного сообщения.

При получении сообщения *CloseConnection* клиент должен рассматривать все сообщения, для которых не было получено ответа как необработанные. Такие сообщения клиент может без дополнительных мер предосторожности послать еще раз при установлении нового соединения.

В случае если сервер предоставляет такую возможность, то клиент может посылать в рамках одного соединения запросы к разным объектам, оптимизируя таким образом использование ресурсов. С другой стороны, клиент может предпочесть установление нового соединения для каждого нового объекта, обслуживаемого сервером, хотя рекомендуется избегать таких случаев.

11.8 Безопасность в CORBA

Обеспечение безопасности в самом широком смысле этого слова было и остается одной из важнейших задач, которую должны решать разработчики распределенных информационных систем. Хотя основные элементы любой универсальной программной подсистемы безопасности хорошо известны и достаточно стандартны (аутентификация, авторизация, шифрование данных, обеспечение их целостности, отслеживание выполненных в системе действий), на практике при их реализации возникает множество вопросов, а именно:

- На кого – разработчиков системных средств или прикладных программ – возлагать главную часть обязанностей с точки зрения обеспечения безопасности?
- Как обеспечить оптимальное сочетание универсальности решения с его гибкостью, эффективностью и надежностью?
- Как обеспечить возможность (и простоту) использования в универсальной системе обеспечения безопасности уже хорошо зарекомендовавших себя различных технологических решений и подходов?
- Как обеспечить взаимодействие приложений, использующих различные реализации систем обеспечения безопасности?
- Как организовать относительно простое и удобное управление безопасностью для менеджеров информационных систем?
- Как обеспечить взаимодействие подсистем с различными уровнями обеспечения

безопасности?

- Как удовлетворить требованиям обеспечения безопасности с учетом того, что требуемый уровень обеспечения безопасности очень сильно отличается для различных систем;
- Как предложить разработчикам универсальное и эффективное решение по доступной цене?

Система обеспечения безопасности в CORBA (CORBA Security Service) должна была дать ответы на эти и многие другие вопросы. Она разрабатывалась долго, примерно в течение двух лет, и первая версия была принята в самом конце 1995 года. В ее разработке принимали участие специалисты из AT&T, DEC, Expersoft, Hewlett-Packard, IBM, Novell, Siemens, Sunsoft, Tivoli Systems и других компаний. В настоящий момент последней утвержденной является версия 1.8 (принята в марте 2002 года). Номер версии говорит о том, что с момента появления в сервис не вносилось кардинальных изменений, хотя появление новых спецификаций сервисов CORBA требовало учета новых реалий и разработки улучшенных версий Security Service.

Как и следовало ожидать, Security Service построен по «драйверному» принципу – спецификация содержит большое количество IDL-интерфейсов, не задавая существенных ограничений с точки зрения их реализаций. Особенностью Security Service является специфическая «область применения» интерфейсов. Б***о***льшая их часть не интересует

прикладных программистов – она предназначена, в первую очередь, для создателей самой реализации Сервиса Безопасности. Другая группа интерфейсов может использоваться прикладными разработчиками, если это необходимо в том или ином конкретном случае. Третья группа предназначена для обеспечения управления настройками системы на этапе ее функционирования и, следовательно, интересует, в первую очередь, администраторов систем, а также разработчиков утилит и приложений для администраторов.

Полная реализация всех – обязательных и необязательных – интерфейсов CORBA Security Service – является сложной, ресурсоемкой и дорогостоящей задачей. В настоящий момент на рынке присутствуют несколько достаточно полно соответствующих стандарту реализаций Security Service, такие, как ORBAsec компании Adiron, IONA OrbixSecurity, MICOSec от ObjectSecurity (соответствуют Security Level 2 версии 1.7 спецификации Security Service). Компания Borland предлагает расширенную службу обеспечения безопасности – Borland Security Service, сочетающую базовые возможности Security Service CORBA с реализацией безопасности в стандартах Java. Ограниченные реализации последних версий Security Service поставляются для TAO и других некоммерческих реализаций CORBA.

Использование Сервиса Безопасности CORBA само по себе не гарантирует требуемого уровня безопасности в каждом конкретном случае – для решения этой задачи требуется совместные усилия системных администраторов, архитекторов и менеджеров

системы на всех этапах ее создания и функционирования, от дизайна до управления в процессе эксплуатации. Например, Сервис Безопасности CORBA предусматривает возможность использования самых разных алгоритмов и технологий шифрования данных, которые обеспечивают различные уровни защиты данных. То же можно сказать и о механизмах выполнения аутентификации пользователей, и о выполнении авторизации при доступе к защищаемым ресурсам.

11.8.1 Основные понятия CORBA Security Service

Спецификация Security Service вводит несколько основных понятий, на которых основана модель обеспечения безопасности. Имеет смысл разобрать их достаточно подробно.

1. **Принципал (principal)**

Принципалом называется пользователь или объект приложения, который должен быть идентифицирован в процессе взаимодействия и с которым должны быть сопоставлены его собственные права.

К сожалению, с использованием термина «принципал» часто происходит путаница. Связано это и со стилем самой спецификации, и с тем, что похожая терминология используется в Java-стандартах распределенных систем, а многие Java- технологии очень сильно связаны с CORBA.

Спецификация Security Service часто вместо «принципала» использует (практически

как синоним) термин «субъект» (subject). Можно встретить и такую трактовку понятий «принципал» и «субъект»: субъект – это пользователь или объект, который необходимо идентифицировать перед началом собственно взаимодействия. Если субъект прошел процедуру идентификации, то с ним сопоставляются права, и этот субъект уже называется принципалом.

В Java Authentication and Authorization Service (JAAS) тоже используются термины «принципал» и «субъект», но несколько в ином значении. Кроме того, формализация этих понятий в JAAS более строгая. В JAAS субъект является совокупностью принципалов, причем под принципалом понимается некоторое имя, сопоставленное с объектом, участвующим во взаимодействии. Другими словами, один субъект (subject) JAAS может при обращении к разным сервисам выступать под разными именами, и каждое такое имя является принципалом.

Подобные терминологические отличия всегда следует иметь в виду при чтении литературы, создании дизайна проекта и общении с другими участниками разработки.

2. Аутентификация (authentication)

Аутентификацией называется процедура идентификации принципала – тот ли он, за кого себя выдает. Для выполнения аутентификации принципал должен предоставить не только имя, но и дополнительные атрибуты, например, пароль или сертификат. Если процедура аутентификации прошла успешно, с принципалом сопоставляются его права.

Что это за права – зависит от системы администрирования, политики безопасности и т.д.

3. **Удостоверения (credentials)**

Удостоверения – это набор атрибутов принципала, используемых при его аутентификации, определении прав доступа, передаче информации и в других случаях. Хорошими примерам удостоверений являются пароль или сертификат принципала. Удостоверениями часто являются привилегии (роли, группы и т.д.), сопоставленные с принципалом.

Удостоверения, являющиеся привилегиями, могут быть сопоставлены с принципалом различным образом. Обычно принципал получает свои привилегии в процессе собственной аутентификации.

В системах с использованием CORBA Security Service удостоверения часто являются частью информации, неявно передаваемой по ORB (наряду с аргументами вызываемого метода, контекстом вызова и контекстом транзакции) – например, для принятия решения, имеет ли данный принципал доступ к защищенному ресурсу и/или при передаче (делегировании) полномочий при наличии цепочки вызовов.

4. **Авторизация (authorization)**

Авторизация – это процесс принятия решения, может ли аутентифицированный принципал получить доступ к конкретному защищенному ресурсу.

Спецификация CORBA Security Service разрешает самые различные подходы выполнения

авторизации. Проверка может выполняться как на стороне клиента, так и сервера. Эта проверка может быть выполнена в коде, написанном разработчиком серверного приложения, а может – на основе данных, заданных администратором безопасности системы таким образом, что приложений даже не знает о выполнении такой проверки.
Обычно при выполнении авторизации принимаются во внимание идентификатор и/или другие привилегии принципала, а также свойства защищенного ресурса.

5. **Делегирование (delegation)**

Делегирование – это передача полномочий (прав и привилегий) при выполнении цепочки вызовов. В этом случае нужно различать «первоначального» клиента (initiator), промежуточный (intermediate) и конечный серверы (final target). Каждый промежуточный объект в последовательности вызовов выступает в роли как сервера (по отношению к объекту, который непосредственно обращается к нему), так и клиента (по отношению к следующему промежуточному или конечному серверу в цепочке вызовов). Поддерживаемые спецификацией Сервиса Безопасности варианты делегирования будут рассмотрены ниже.

6. **Доверительные отношения (trust)**

В широком понимании смысла этого термина, доверительные отношения – это отношения двух участников обмена информацией, для которых можно отказаться от выполнения действий и проверок, обязательных в других случаях. Следствия установки

доверительных отношений могут быть самые различные. Например, отказ от выполнения аутентификации «доверенного» клиента при его обращении к «доверенному» серверу, выполнение облегченной процедуры шифрования или даже отказ от шифрования, отказ или облегченная процедура авторизации и т.п. Часто говорят, что установка доверительных отношений между конкретными сервисами приводит к заданию «защищенного взаимодействия» (security association).

Установка доверительных отношений ставит задачу повышения производительности системы за счет отказа от ресурсоемких процедур защиты информации и упрощение администрирования информационной системы. Желательность установки доверительных отношений и их конкретный вид следует иметь в виду уже на ранних стадиях разработки архитектуры проекта.

11.8.2 Структура CORBA Security Service

Обеспечение безопасности требует решения многих подзадач различной важности. Кроме того, существует объективное противоречие между универсальностью решения и его гибкостью и эффективностью. Не следует забывать о необходимости обеспечения определенной совместимости различных реализаций, использующих разные технологические подходы.

Все это приводит к тому, что спецификация делит функциональность Security Service на группы: есть интерфейсы, реализация которых обязательна для совместимости

со стандартом, есть интерфейсы, реализация которых является необязательной. Это не единственный критерий разделения интерфейсов на группы. Еще одним критерием является обеспечиваемый уровень совместимости различных реализаций. Очевидно, что совместимость связана с использованием определенных ограничений, отказ от которых приведет к отказу и от совместимости.

Спецификация вводит группа интерфейсов, называемые «пакетами» (package). Определено 7 групп таких пакетов:

1. Пакеты, определяющие основную, наиболее затребованную и часто используемую в реальных системах, функциональность Сервиса Безопасности. В эту группу входят два основных пакета (если разработчик реализации объявляет, что ORB поддерживает CORBA Security Service, то он должен реализовать хотя бы один из этих пакетов):

- Пакет уровня 1 (Level 1). Он содержит функциональность, которая обеспечивает защиту ресурсов средствами только ORB и Security Service, без явного использования Security Service API в коде самого приложения. Такие приложения в спецификации называются security unawared-приложениями, т.е. приложениями, при изучении кода которых нельзя узнать, будут ли при выполнении этого приложения задействованы механизмы обеспечения безопасности CORBA. Разумеется, в этом случае нет возможности управлять доступом к ресурсам на основе информации, которая станет

известна только на этапе работы программы, например, текущего значения аргументов вызова или времени выполнения запроса.

- Пакет уровня 2 (Level 2). Функциональность сервиса на этом уровне позволяет явно управлять режимом доступа с помощью кода самого приложения. Кроме того, на уровне 2 объявлены средства администрирования, базирующиеся на использовании политик (policies) обеспечения безопасности.

2. Пакеты, определяющие вспомогательную (optional) функциональность Сервиса Безопасности, которая используется только в некоторых реальных системах. В настоящий момент в эту группу входит единственный пакет, связанный с обеспечением «доказательности» (“non-repudiation”) выполняемых действий. Эту функциональность можно трактовать как ведение журналов, в которые заносится информация о том, кто отправил данные и кто их получил. Предполагается, что надежность методов хранения такой контрольной информации такова, что на нее можно ссылаться при разрешении конфликтов даже на уровне юридических разбирательств.

3. Пакеты обеспечения взаимозаменяемости реализаций (Security Replaceability packages). Это очень важные пакеты, которые влияют на то, насколько гибко ORB может взаимодействовать с Сервисом Безопасности (разумеется, эти пакеты интересны в первую очередь разработчикам реализаций ORB и CORBA Security Service, а не прикладным программистам). Тем не менее, знание таких особенностей реализации ORB может

пригодиться на этапе выбора нужной реализации ORB.

В эту группу входят два пакета, оба обеспечивают возможность взаимодействия ORB с различными реализациями Security Service, но разными путями:

- Обеспечение взаимозаменяемости на уровне ORB (ORB services replaceability package). При таком подходе сам ORB почти либо совсем не «общается» с сервисом безопасности – вместо этого на нем регистрируются специальные интерсепторы, которые и взаимодействуют с Сервисом Безопасности. Порядок вызова методов этих интерсепторов и их функциональности определена в этом пакете спецификации.
- Обеспечение взаимозаменяемости на уровне взаимодействия с Security Service (Security Service replaceability package). Этот пакет определяет, если можно так выразиться, «интерфейс» Сервиса Безопасности при его взаимодействии с ORB. Использование этого интерфейса приводит к тому, что ORB и Сервис Безопасности становятся максимально независимыми друг от друга, что позволяет использовать различные реализации ORB и CORBA Security Service при совместной работе. В таком режиме ORB может и не использовать интерсепторы – обращение к переносимому API из кода самого ORB'а обеспечивает определенный уровень взаимозаменяемости.

Реализация ORB'а может не использовать ни один из этих подходов – вся необходимая функциональность может быть встроена в код реализации ORB'а. В этом

случае не приходится говорить о гибкости настроек и способности ORB'а взаимодействовать с различными реализациями CORBA Security Service.

Спецификация называет реализации ORB, которые поддерживают функциональность одного из этих пакетов, как «ORB, взаимодействующий с сервисом безопасности» (Security Ready ORB). Обратите внимание, что ORB может обеспечивать защищенное взаимодействие, но не относиться к классу Security Ready.

4. Общие средства обеспечения переносимости (Common Secure Interoperability (CSI) Feature packages). Определены три уровня переносимости:

- CSI уровня 0. Этот уровень характеризуется тем, что нет поддержки делегирования привилегий. При обращении первоначального клиента к серверному объекту передается только идентификатор этого клиента (и никакие другие атрибуты). Если этот серверный объект является промежуточным серверным объектом, т.е. обращается к другому серверному объекту, то при обработке этого последующего вызова будет использован идентификатор промежуточного объекта, а не первоначального клиента.
- CSI уровня 1. На этом уровне по-прежнему может передаваться только идентификатор первоначального клиента, но не его другие атрибуты, зато этот идентификатор может передаваться дальше по цепочке вызовов. Такое делегирование называется «неограниченным» (unrestricted). Промежуточные серверы

с точки зрения прав доступа рассматриваются как первоначальные клиенты. В спецификации и литературе часто используется термин «подражание» (impersonation).

- CSI уровня 2. Этот уровень обеспечивает совместимость ORB со всей функциональностью Сервиса Безопасности. Поддерживается передача не только идентификатора принципала, но и всех его атрибутов (включая роли и группы). Возможна передача и использование привилегий сразу нескольких принципалов – так называемое «комплексное делегирование» (composite delegation).

Стоит обратить внимание на то обстоятельство, что все три уровня совместимости обеспечивают базовые возможности по защите информации.

5. Пакет обеспечения переносимости на уровне протокола (SECIOP Interoperability package). ORB, реализующий функциональность этого пакета, способен генерировать и передавать информацию, связанную с защитой ресурсов, на уровне объектных ссылок и протокола GIOP. Это означает, что два SECIOP-совместимых различных ORB'а способны взаимодействовать друг с другом в защищенной среде – в случае, если они используют одни и те же (или совместимые) технологии защиты данных.

6. Пакеты используемых механизмов обеспечения защиты (Security Mechanism packages). Как уже говорилось, CORBA Security Service позволяет использовать качестве реализации самые различные технологии и подходы. Тем не менее, спецификация

формально оговаривает интерфейсы, связанные с применением следующих четырех стандартных протоколов:

- Протокол SPKM. Протокол позволяет использовать открытые ключи и с точки зрения поддержки и делегирования атрибутов принципала соответствует CSI уровня 0.
- Протокол GSS Kerberos. С точки зрения поддержки и делегирования атрибутов принципала соответствует CSI уровня 1.
- Протокол CSI-ECMA. С точки зрения поддержки и делегирования атрибутов принципала соответствует CSI уровня 1, хотя может при желании использоваться в соответствие с правилами CSI 1 и 0.
- Протокол SSL. С точки зрения поддержки и делегирования атрибутов принципала соответствует CSI уровня 0.

Использование трех первых протоколов требует, чтобы ORB поддерживал расширения протокола IIOP, соответствующие требованиям SECIOP. Использование SSL не предъявляет к реализации ORB каких-либо специфических требований – обеспечение защиты данных ложится на уровень транспортного протокола, т.е. более низкий логический уровень, чем уровень CORBA.

7. Последний пакет связан с обеспечением безопасности при взаимодействии CORBA и DCE – технологии создания распределенных систем, с которой у CORBA

давние дружеские отношения.

11.8.3 Делегирование в CORBA Security Service

Графически возможные (согласно спецификации) схемы делегирования можно выразить следующим образом:

- Запрет делегирования.

Рисунок 11.1

Каждый промежуточный объект, участвующий в цепочке вызовов, выступает от «собственного имени» и использует свои атрибуты и привилегии.

- Простое (simple) делегирование.

Рисунок 11.2

Клиент делегирует свои полномочия промежуточному серверу, который может использовать их как для разрешения (запрета) доступа к своим ресурсам, так и для передачи дальше, по цепочке вызовов. Каждый промежуточный (и конечный) сервер

считает, что к нему обращается начальный клиент.

- Комплексное (composite) делегирование.

Рисунок 11.3

Клиент позволяет делегировать свои полномочия дальше по цепочке вызовов, при этом промежуточный сервер может добавить к ним собственные привилегии. Если это необходимо, последующие сервера могут анализировать полученные по ORB привилегии независимо друг от друга.

- Комбинированное (combined) делегирование.

Рисунок 11.4

Режим похож на предыдущий, за одним исключением: промежуточный объект создает обобщенные наборы атрибутов и привилегий, и последующий сервер (конечный сервер на приведенном рисунке) уже не может отличить, к какому из участвующих в цепочке вызовов объекту относятся отдельные привилегии.

Сервис Безопасности CORBA позволяет при делегировании вводить и другие ограничения. Например, клиенты и промежуточные сервера могут делегировать только часть своих полномочий; администратор или программист может задать период времени, в течение которого можно использовать делегирование и т.д.

Для приложений, которые непосредственно не обращаются к API Security Service (Level 1), режим делегирования задается по умолчанию с использованием политик безопасности, и промежуточные объекты не могут в процессе работы приложения менять указанный способ делегирования.

11.8.4 Домены безопасности

Одним из важных понятий спецификации Сервиса Безопасности является понятие «домена безопасности» (security domain). Доменная структура системы обеспечения безопасности оказывает непосредственное влияние и на совместимость, и на переносимость используемых программных средств, и на уровень сложности администрирования, и на эффективность работы приложений. Спецификация различает три вида доменов безопасности:

- Технологический домен (security technology domain).

В один технологический домен входят приложения, использующие, например, единую систему аутентификации принципалов, одну технологию распространения ключей, одну систему шифрования и/или обеспечения целостности данных, одну систему

принятия решения о предоставлении доступа, одну систему аудита и т.д. Решение задачи защищенного взаимодействия при наличии нескольких технологических доменов, как правило, связано с серьезными сложностями. Спецификация Сервиса Безопасности не формализует никаких правил обеспечения взаимодействия в этом случае.

- Домен политик безопасности (security policy domain).

CORBA Security Service обладает очень развитой системой QoS (quality of service), т.е. средствами настройки системы с точки зрения ее оптимизации по различным критериям. Спецификация вводит большое количество политик безопасности. Поскольку в CORBA «единицей защиты» является объект, то на практике защищать каждый объект отдельно совершенно нереально. Вместо этого объекты объединяются в домены политик безопасности (этот подход используется в CORBA не только для политик безопасности, но и вообще для любых политик), и в каждом таком домене единый набор политик относится ко всем объектам данного домена. Как обычно, для управления политиками на уровне домена необходимо определить менеджера этих политик (Policy Manager). Применительно к доменам политик безопасности, такой менеджер часто называется «SecurityAuthority»

Поскольку домены политик безопасности ничем принципиально не отличаются от других доменов политик CORBA, поддерживаются иерархии доменов и объединения (федерации) доменов. Домены могут перекрываться –один и тот же объект может входить

в различные домены политик, например, в один домен политик с точки зрения политик управления правами доступа и в другой – с точки зрения политик аудита.

Поскольку CORBA Security Service поддерживает два вида приложений с точки зрения их явного использования API Сервиса, то можно делить домены политик безопасности еще на две группы – домены системных политик безопасности и домены прикладных политик безопасности. Задание и использование системных политик безопасности выполняется силами ORB, Сервиса Безопасности и операционной системы, и приложения об этом может даже не знать,

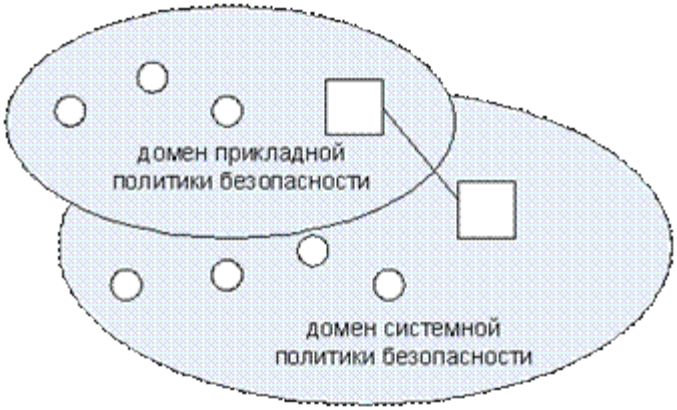


Рисунок 11.5

- Домен среды безопасности (security environment domain)

Понятие «домена среды безопасности» тесно связано с «доменом политик безопасности». Домен политик – это логическая область с заданным набором политик и

их значений. Домен среды – это область, в которой используется некий единый подход, который обеспечивает защиту ресурсов в соответствии с заданными политиками.

Домен среды безопасности – понятие логическое, явно границы таких доменов не определены ни на уровне приложений, ни на уровне самого Сервиса Безопасности. Обычно домены среды безопасности появляются в связи с отказом от использования тех или иных методов защиты данных на уровне Сервиса Безопасности. Например, если шифрование данных используется на уровне транспортного протокола, то нет разумных причин возлагать решение этой задачи на приложение или ORB. Другой пример – отказ от избыточных операций аутентификации принципалов в случае, если между объектами в одном домене среды безопасности установлены доверительные отношения.

С точки зрения организации взаимодействия в защищенной гетерогенной среде, можно также принять во внимание отличия в реализации ORB'ов, введя тем самым, понятие «домена ORB».

Общая схема организации защищенного взаимодействия объектов с учетом наличия различных технологических доменов и доменов ORB может выглядеть так:

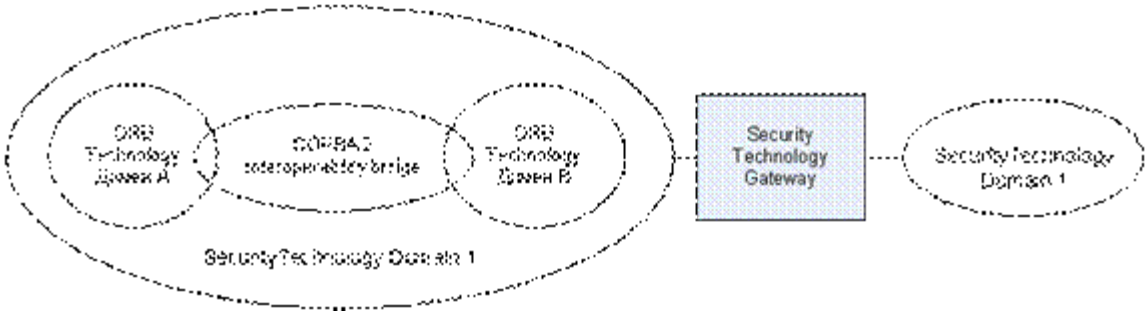


Рисунок 11.6

11.8.5 Объектная модель обеспечения безопасности

Объектную модель CORBA Securtiy Service удобнее рассматривать по частям, а именно, с точки зрения различных участников процесса разработки и эксплуатации системы. Здесь мы будем рассматривать эту модель с двух точек зрения – с точки зрения разработчика приложений и администратора системы.

11.8.5.1 Модель с точки зрения разработчика

Разумеется, здесь разговор пойдет только о таких приложениях, которые явно используют те или возможности Сервиса Безопасности. Средства, предоставляемые этим сервисом, позволяют решать следующие задачи:

- Определить, какие возможности поддерживаются данной реализацией сервиса (и ORB’а).
- Задать необходимые удостоверения и привилегии принципала для последующей его аутентификации, если это необходимо.

- Выполнить защищенный удаленный вызов.
- Управлять режимом обеспечения безопасности на уровне промежуточных и конечных серверных объектов, управлять делегированием полномочий и принимать решение о предоставлении доступа к защищенным ресурсам.
- Отслеживать происходящие в процессе работы системы действия.
- Вести журнал в режиме обеспечения доказательности выполненных действий.
- Управлять политиками безопасности для объектов.

Начнем с задания удостоверений и аутентификации.

Рисунок 11.7

Основой для выполнения аутентификации (если она необходима) является объект

Credentials. Пути создания и настройки этого объекта могут быть различными. Например, если принципал уже аутентифицирован, то для получения объекта Credentials можно использовать интерфейс Current. Напоминаем, что интерфейсы Current, определенные в различных сервисах CORBA, в принципе предназначены для решения одной задачи, а именно: они позволяют получить доступ к информации, передаваемой по ORB, в контексте (и потоке) текущего вызова. Интерфейс Current Сервиса Безопасности обеспечивает доступ к параметрам, связанным с безопасностью (подобно тому, как интерфейс Current Сервиса Объектных Транзакций обеспечивает доступ к параметрам текущей транзакции).

В общем случае, т.е. тогда, когда принципал (User на приведенном выше рисунке) должен быть аутентифицирован, но еще не аутентифицирован.

User Sponsor (который нарисован с использованием пунктирной линии потому, что это не CORBA-интерфейс, а некоторый фрагмент кода приложения) получает от пользователя необходимые данные (например, учетное имя и пароль), а затем обращается к объекту Principal Authenticator, который проводит аутентификацию и в качестве ее результата получает объект Credentials, который содержит идентификатор принципала и его привилегии. Как правило (но не обязательно!) эти действия выполняются в клиентском приложении.

После того, как все необходимые удостоверения заданы для данного принципала,

они используются при выполнении защищенных вызовов. Программист может при необходимости менять эти удостоверения. Сделать это можно двумя способами: либо изменить состояние соответствующего объекта Credentials (для чего предусмотрен метол set_attributes()), либо изменив политики на уровне самой объектной ссылки обычными средствами управления политиками CORBA (за режим привилегий при вызове отвечает политика InvocationCredentialsPolicy, рисунок 9.8).

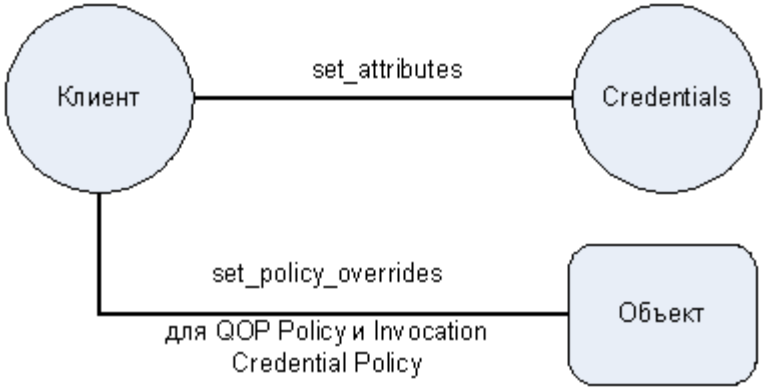


Рисунок 11.8

В обоих случаях эффект изменений распространяется на все последующие вызовы с использованием данного объекта Credentials и/или данной объектной ссылки.

Что касается собственно выполнения защищенного удаленного вызова, то здесь все внешне выглядит, как обычно – все необходимое с точки зрения параметров безопасности выполняется ORB'ом и Сервисом Безопасности без участия самих приложений – как

клиента, так и сервера.

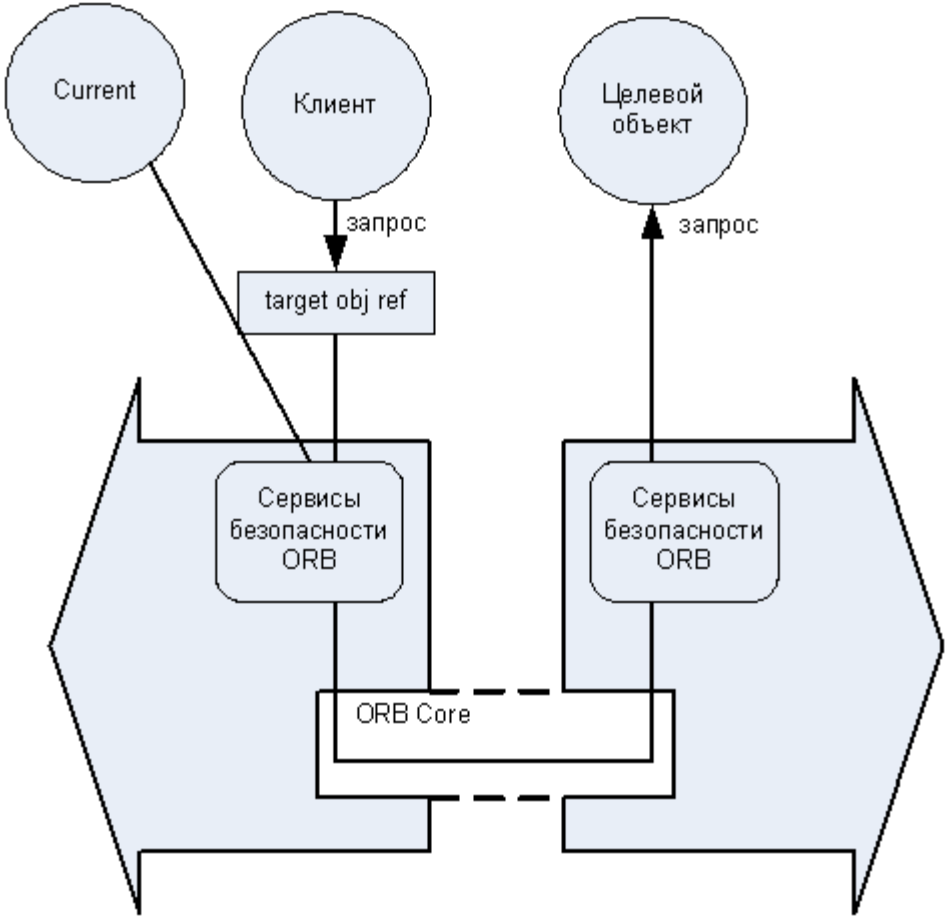


Рисунок 11.9

Security Service перехватывает вызов и с помощью интерфейса Current получает

доступ ко всем атрибутам объекта Credentials.

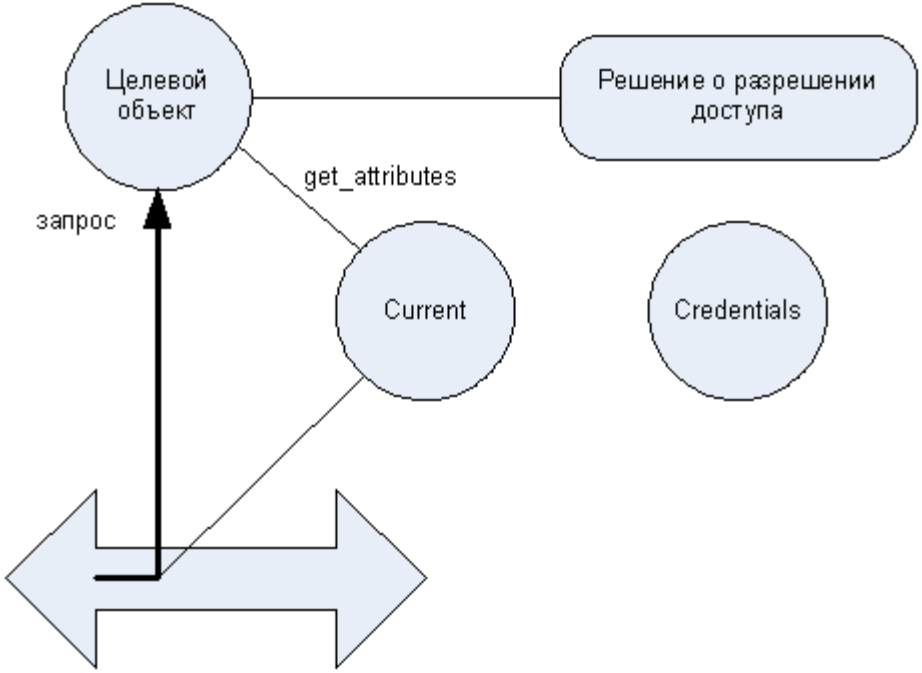


Рисунок 11.10

Переходим на сторону сервера. Там выполняются похожие действия –Security Service перехватывают пришедший по ORB'у запрос. Получить доступ к параметрам безопасности – идентификатору принципала и его привилениям – можно с помощью вызова метода get_attributes() интерфейса Current. Далее эти привилегии используются при принятии решения, можно ли разрешить доступ данному принципалу в контексте

этого вызова к данному серверному объекту (т.е. выполняется авторизация), см. рисунок 11.10.

Похожие механизмы используются и в случае, когда имеет место цепочка вызовов, т.е. некоторый промежуточный объект выступает сначала в роли сервера, а затем, при последующем вызове – уже в роли клиента. Если приложение использует API Security Service, то промежуточный объект может анализировать параметры безопасности пришедшего запроса и, в случае необходимости, менять из перед выполнением следующего вызова в цепочке. Для этой цели опять используется интерфейс Current:

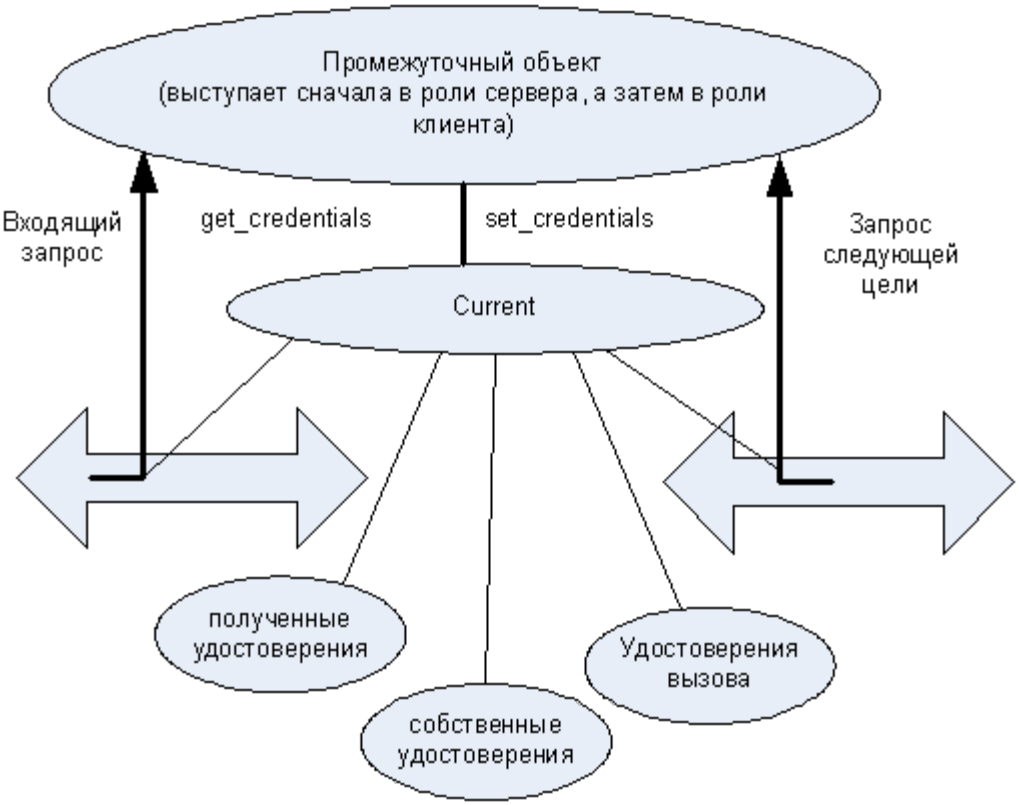


Рисунок 11.11

Для получения привилегий, содержащихся в полученном запросе, удобно использовать атрибут received_credentials интерфейса Current.

Промежуточный объект может быть самостоятельным принципалом и выполнять последующий вызов «от собственного имени». Это означает, что он должен получить

свой объект Credential вместо того, чтобы извлекать существующий в контексте пришедшего запроса объект с помощью Current. В этом случае промежуточный объект обращается к методу authenticate() объекта PrincipalAuthenticator, а затем настраивает его с помощью метода set_attributes() объекта Credentials.

Что касается других аспектов управления приложением – принятия решений о предоставлении доступа или выполнении аудита – то спецификация не предусматривает объявления определенных политик и не занимается строгой формализацией этих процессов. Тем не менее, CORBA Security Service предоставляет некоторые стандартные вспомогательные объекты, например, AuditChannel – логический канал вывода информации, или AuditDecision (для принятия решения, нужно ли отслеживать определенное событие), которые удобно использовать в системе аудита,

11.8.5.2 Модель с точки зрения администратора

Модель администрирования (формализованная в спецификации на уровне объектов) ограничивается политиками безопасности и средствами управления этими политиками. Администрирование на уровне технологических доменов и доменов среды безопасности не рассматривается вообще, так как эта функциональность в очень сильной степени является реализационно-зависимой. Остается решение следующих задач:

- Обеспечение создания защищенных объектов, сопоставленных с политиками безопасности.

- Обеспечение работы с менеджерами доменов для таких объектов.
- Управление политиками с использованием этих менеджеров доменов.
- Сопоставление операций и прав доступа.

Среди всех возможных видов компонентов и сервисов, которые могут использовать политики безопасности (сами приложения, ORB, реализации Сервиса Безопасности CORBA, другие сервисы), спецификация подробно описывает только то, что происходит на уровне ORB.

```
local interface SecurityManager
{
  readonly attribute Security::MechandOptionsList supported_mechanisms;
  readonly attribute CredentialsList own_credentials;
  readonly attribute RequiredRights required_rights_object;
  readonly attribute PrincipalAuthenticator principal_authenticator;
  readonly attribute AccessDecision access_decision;
  readonly attribute AuditDecision audit_decision;
  ...
  CORBA::Policy get_security_policy (in CORBA::PolicyType policy_type);
};
```

Прежде чем рассматривать подробнее модель администрирования, необходимо ознакомиться с основными политиками безопасности.

11.8.6 Основные политики безопасности

CORBA Security Service определяет следующие основные стандартные политики безопасности:

- Политика предоставления доступа при вызове (Invocation access policy).

```
const CORBA::PolicyType SecClientInvocationAccess = 1;
const CORBA::PolicyType SecTargetInvocationAccess = 2;

module SecurityAdmin
{
  interface AccessPolicy : CORBA::Policy { ... }
  interface DomainAccessPolicy : AccessPolicy { ... }
  ...
};
```

- Политика аудита при выполнении вызова (Invocation audit policy). Эта политика позволяет указать, какие события в процессе выполнения вызовов нужно отслеживать в системе аудита.

```
const CORBA::PolicyType SecClientInvocationAudit = 4;
const CORBA::PolicyType SecTargetInvocationAudit = 5;

module SecurityAdmin
{
  interface AuditPolicy : CORBA::Policy { … }
  ...
};
```

- Политика выполнения удаленного вызова (Secure Invocation policy), которая, в частности, позволяет определить необходимость установки доверительных отношений и уровень защиты информации (Quality of Protection).

```
const CORBA::PolicyType SecClientSecureInvocation = 8;
const CORBA::PolicyType SecTargetSecureInvocation = 9;

module SecurityAdmin
{
  interface SecureInvocationPolicy : CORBA::Policy { … }
  ./..
```

```
};
```

- Политика делегирования при выполнении вызова (Invocation delegation policy); позволяет задать поведение промежуточного объекта, участвующего в цепочке вызовов, с точки зрения делегирования удостоверений.

```
const CORBA::PolicyType SecDelegation = 7;

module SecurityAdmin
{
  interface DelegationPolicy : CORBA::Policy { … }
    ...
};
```

- Политика предоставления доступа для приложения (Application access policy). Эта политика отличается от Invocation access policy тем, что может использоваться непосредственно приложением и не обязана находиться под управлением менеджера домена политик безопасности. Invocation-политики управляются и используются не приложением, а ORB'ом.

```
const CORBA::PolicyType SecApplicationAccess = 3;
```

- Политика аудита для приложения. Как и предыдущая политика, она предназначена для использования (управления режимом аудита) непосредственно из приложения.

```
const CORBA::PolicyType SecApplicationAudit = 6;
```

- Политика доказательности (Non-repudiation policy). Определяет режимы генерации и контроля свидетельств о событиях. За ее использование отвечает приложение, а не ORB.

```
const CORBA::PolicyType SecNonRepudiation = 10;
```

- Политика конструирования (Construction policy), которая позволяет определить, нужно ли создавать новый домен политик безопасности при создании нового объекта с определенными политиками (CORBA::SecConstruction). Политика применяется в момент создания объектной ссылки объектным адаптором POA.

11.8.6.1 Управление политиками безопасности на уровне приложения

Взаимодействие с Сервисом Безопасности CORBA – явное или неявное – начинается с момента создания объектной ссылки на CORBA-объект при работе в защищенной среде. Как всегда, фабрикой CORBA-объектов являются объектные адаптеры (POA). В этот момент ORB сопоставляет объектную ссылку с разными видами доменов безопасности – политик безопасности, технологическими доменами, доменами среды. После получения объектной ссылки приложение может использовать стандартные методы типов CORBA::Object, DomainManager, CORBA::Policy для управления политиками. Графически это можно изобразить следующим образом:

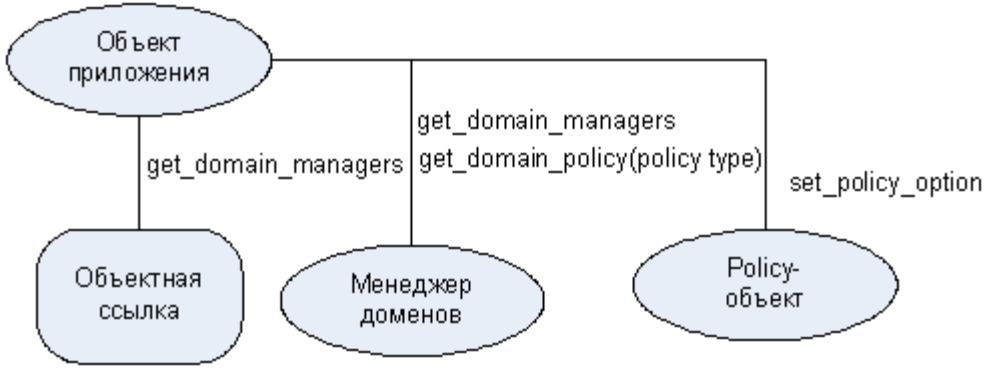


Рисунок 11.12

По объектной ссылке можно получить список Менеджеров доменов, с которым сопоставлен данный объект, и найти менеджер (объект DomainManager), который отвечает за нужную политику. Метод DomainManager::get_domain_policy() обеспечивает получение объекта Policy с параметрами для указанной политики. Дальнейшая установка нужного состояния Policy-объекта зависит от того, с какой политикой необходимо работать – в CORBA Security Service предусмотрены различные интерфейсы для работы с различными политиками:

1. **Доказательность (non-repudiation)**

Интерфейсы, связанные с поддержкой доказательности, относятся к Уровню 2, т.е. отслеживание происходящих в системе событий происходит не автоматически, а под управлением приложения. Спецификация оговаривает типы событий, политики и

интерфейсы, которые должен использовать разработчик, чтобы создавать и сохранять информацию о происходящем.

По сути, поддержка доказательности сводится к ведению защищенной базы данных, в которую заносятся записи, связанные с фиксацией создания, отправки, получения и использования удаленных сообщений. Спецификация Сервиса Безопасности определяет следующие виды деятельности, информация о выполнении которых сохраняется в БД:

```
enum EvidenceType
{
  SecProofofCreation,
  SecProofofReceipt,
  SecProofofApproval,
  SecProofofRetrieval,
  SecProofofOrigin,
  SecProofofDelivery,
  SecNoEvidence
};
```

Важнейшими типами событий являются Creation (создание сообщения) и Receipt (его получение адресатом).

Заносимая в БД информация о событии обычно содержит тип события и сопоставленное с ним описание. Описание включает множество параметров, в том числе информацию о принципале, о дате и времени работы с событием и т.д. Интерфейсы обеспечения доказательности содержат методы, которые позволяют создать описание на основе его параметров.

Несколько слов о соотношении средств обеспечения доказательности в CORBA

Security Service и более общих моделях обеспечения доказательности.

Существуют стандарты таких моделей. Графическое представление модели ISO-стандарта приведено на рисунке ниже:

Рисунок 11.13

Серым цветом выделен компонент, включенный в систему обеспечения доказательности CORBA в данной версии спецификации. Основная функциональность, обеспечиваемая этим компонентом:

- Генерация информации о происходящих событиях.
- Контроль (verification) этой информации.
- Генерация запроса для информации, связанной с отправленным событием.

- Получение запроса для информации, связанной с полученным событием.
- Анализа информации о событии.

Другие аспекты обеспечения доказательности – собственно обеспечение хранения информации в БД и извлечение ее оттуда, обеспечение доставки этой информации компонентам или пользователем, непосредственно принимающим решения в случае конфликтов, само разрешение конфликтов на основе полученной информации – в спецификации CORBA Security Service не рассматриваются.

2. Интерфейс Current

Обычно разработчики прибегают к использованию API Security Service в тех случаях, когда нужно отслеживать действия в системе и сохранять информацию об этом, предоставлять права доступа с учетом большого количества информации (в том числе той, которая известна только на момент выполнения вызова) или при организации взаимодействия с унаследованными системами.

Одним из наиболее важных интерфейсов Сервиса Безопасности является интерфейс Current - точнее, таких интерфейсов даже два – они определены в разных модулях. Это интерфейсы SecurityLevel1::Current и SecurityLevel2::Current. Получение доступа к интерфейсу Current выполняется стандартным образом, с помощью вызова для ORB метода resolve_initial_references() с аргументом «SecurityCurrent».Как обычно, после вызова метода resolve_initial_references() нужно выполнить преобразование типа

результата к нужному интерфейсу Current. При преобразовании к SecurityLevel2::Current необходимо предварительно убедиться, что реализация поддерживает этот уровень. Получить информацию о реализации можно с помощью вызова метода CORBA::ORB::get_service_information(), задав в качестве параметра типа CORBA::ServiceType значение константы CORBA::Security. Метод возвращает одно из следующих значений:

```
module Security
{
  const CORBA::ServiceOption CommonInteroperabilityLevel0 = 10;
  const CORBA::ServiceOption CommonInteroperabilityLevel1 = 11;
  const CORBA::ServiceOption CommonInteroperabilityLevel2 = 12;
};
```

На уровне Security Level 1 интерфейс Current содержит единственный метод, который позволяет получить список атрибутов безопасности в контексте выполняемого защищенного вызова:

```
module SecurityLevel1
{
  local interface Current : CORBA::Current
  {
    Security::AttributeList get_attributes (in
      Security::AttributeTypeList attributes);
  };
};
```

Единственный метод возвращает связанный с контекстом вызова список атрибутов безопасности указанных типов.

IDL-объявления основных типов выглядят так:

```
module Security
{

  typedef string SecurityName;

  typedef sequence <octet> Opaque;

  // Объявления констант для Security Service Options
  const CORBA::ServiceOption SecurityLevel1 = 1;
  const CORBA::ServiceOption SecurityLevel2 = 2;
  const CORBA::ServiceOption NonRepudiation = 3;
  const CORBA::ServiceOption SecurityORBServiceReady = 4;
  const CORBA::ServiceOption SecurityServiceReady = 5;
  const CORBA::ServiceOption ReplaceORBServices = 6;
  const CORBA::ServiceOption ReplaceSecurityServices = 7;
  const CORBA::ServiceOption StandardSecureInteroperability = 8;
  const CORBA::ServiceOption DCESecureInteroperability = 9;

  // Service options for Common Secure Interoperability
  const CORBA::ServiceOption CommonInteroperabilityLevel0 = 10;
  const CORBA::ServiceOption CommonInteroperabilityLevel1 = 11;
  const CORBA::ServiceOption CommonInteroperabilityLevel2 = 12;


...
  // Расширяемые семейства (families) для стандартных типов данных

  struct ExtensibleFamily
  {
    unsigned short family_definer;
    unsigned short family;
  };
  typedef sequence<octet> OID;
  typedef sequence<OID> OIDList;
```

```
  typedef unsigned long SecurityAttributeType;
  // Общие атрибуты (family = 0)
  const SecurityAttributeType AuditId = 1;
  const SecurityAttributeType AccountingId = 2;
  const SecurityAttributeType NonRepudiationId = 3;

  // Атрибуты привилегий (family = 0)
  const SecurityAttributeType _Public = 1;
  const SecurityAttributeType AccessId = 2;
  const SecurityAttributeType PrimaryGroupId = 3;
  const SecurityAttributeType GroupId = 4;
  const SecurityAttributeType Role = 5;
  const SecurityAttributeType AttributeSet = 6;
  const SecurityAttributeType Clearance = 7;
  const SecurityAttributeType Capability = 8;

  struct AttributeType
  {
    ExtensibleFamily attribute_family;
    SecurityAttributeType attribute_type;
  };
  typedef sequence<AttributeType> AttributeTypeList;

  struct SecAttribute
  {
    AttributeType attribute_type;
    OID defining_authority;
    Opaque value; // значение зависит от defining_authority
  };
  typedef sequence <SecAttribute> AttributeList;
}
```

Таким образом, метод SecurityLevel1::Current::get_attributes() возвращает, по сути, состояние объекта Credentials, сопоставленного с выполняемым вызовом. На уровне

Level1 вызов этого метода обычно выполняется самим ORB'ом.

На уровне Level 2 появляется дополнительная функциональность, связанная с явным управлением процессом обеспечения безопасности:

```
module SecurityLevel2
{
  ...

  local interface Credentials
  {
    Credentials copy ();
    void destroy();

    readonly attribute Security::InvocationCredentialsType
      credentials_type;
    readonly attribute Security::AuthenticationStatus
      authentication_state;
    readonly attribute Security::MechanismType mechanism;
    attribute Security::AssociationOptions
      accepting_options_supported;
    attribute Security::AssociationOptions accepting_options_required;
    attribute Security::AssociationOptions
      invocation_options_supported;
    attribute Security::AssociationOptions
      invocation_options_required;

    ...

    boolean set_attributes (
      in Security::AttributeList requested_attributes,
      out Security::AttributeList actual_attributes
      );

    Security::AttributeList get_attributes (
```

```
      in Security::AttributeTypeList attributes
      );

    ...
  };

  typedef sequence <Credentials> CredentialsList;

  local interface ReceivedCredentials : Credentials { ... };
...
  local interface Current : SecurityLevel1::Current
  {
    readonly attribute ReceivedCredentials received_credentials;
  };
};
```

Заключение

Совместное использование возможностей ORB, стандартных компонентов Сервиса Безопасности CORBA и его API позволяет создавать распределенные системы с требуемым уровнем защиты. При этом разработчик можно легко задействовать стандартные решения в области обеспечения безопасности, которые уже широко используются в компьютерной индустрии.

12 Стандарт ODBC

ODBC предназначен для предоставления прикладным разработчикам функциональных возможностей по обработке баз данных независимо от типа данных, к которым выполняется доступ, - базам данных ISAM, текстовым данным (Excel) или базам данных SQL.

Эта цель достигается путем закрепления каждого драйвера ODBC за одним из предопределенных уровней соответствия. Чтобы считаться драйвером ODBC, драйвер должен соответствовать спецификациям ядра ODBC.

Эти требования гарантируют, что разработчик приложения всегда может рассчитывать на одни и те же функциональные возможности независимо от того, к каким данным происходит обращение. Если формат используемых данных непосредственно не поддерживает основные функциональные возможности, драйвер ODBC должен эмулировать эти функции. С помощью ODBC можно манипулировать данными любой СУБД (и даже данными, не имеющими прямого отношения к базам данных, например данными в файлах электронных таблиц или в текстовых файлах), если для них имеется ODBC-драйвер. Для каждой используемой СУБД нужен собственный ODBC-драйвер. Говоря об ODBC, нельзя не отметить, что спецификация ODBC подразумевает несколько стандартов на ODBC-драйверы. Эти стандарты отличаются различной функциональностью, которая должна быть реализована в таком драйвере. Метод

соединения с источником данных, получения сообщений об ошибках, а также стандартные интерфейсы регистрации являются общими для всех драйверов. Для обеспечения унифицированности драйверы придерживаются основных требований. Открытый интерфейс доступа к базам данных представляет собой библиотеку функций, которая позволяет прикладной программе обращаться к различным СУБД. Используя структурированный язык запросов SQL. Таким образом, разработчик прикладной программы может создавать программу для виртуальной базы данных и позволить загружаемому драйверу преобразовать логические данные в данные конкретной СУБД или систем, используемых данной прикладной программой.

Привлекательность ODBC обусловлена ее портативностью и взаимодействием с кодом прикладной программы. ODBC функционирует как стандартный интерфейс для разработчиков прикладных программ, а также для разработчиков библиотек драйверов.

Архитектура ODBC имеет четыре основных компонента: - прикладная программа, - диспетчер драйверов, - драйвер, - источник или источники данных.

Приложение, использующее интерфейс ODBC, выполняет следующие задачи: Запрашивает соединение с источником данных. Посылает SQL- запросы к источнику данных. Описывает область хранения и формат для результатов SQL- запросов. Запрашивает данные. Обрабатывает ошибки. Оповещает об ошибках. Осуществляет фиксацию или откат действий в режиме транзакций. Закрывает соединение с источником

данных.

Диспетчер драйверов, совместно с набором средств ODBC, является динамически связанной библиотекой (DLL), которая загружает драйверы, обеспечивая единственную точку входа в функции ODBC для различных драйверов.

Функции интерфейса ODBC подразделяются на семь групп.

1. назначение и отмена назначения: идентификатор окружения, идентификатор соединения, идентификатор оператора
2. соединение
3. выполнение SQL-операторов
4. получение результатов
5. управление транзакциями
6. идентификация ошибок
7. смешанные функции

Теперь подробнее:

1. Назначение и отмена назначения

Идентификатор окружения определяет базу данных, идентификатор соединения определяет соединение с базой данных и идентификатор оператора определяет отдельный SQL-оператор.

Идентификатор окружения. Этот идентификатор указывает на область памяти для

глобальной информации. Переменная типа HENV также включает в себя сведения обо всех соединениях с базами данных и информацию о том, какое соединение является текущим.

Идентификатор соединения. Этот идентификатор указывает на область памяти для информации о конкретном соединении. В то время как каждый идентификатор соединения ассоциируется с единственным идентификатором окружения, этот единственный идентификатор окружения может иметь один или более связанных с ним идентификаторов соединения.

Идентификатор оператора. Этот идентификатор, который относится к типу HSTMT, указывает на область памяти для информации о SQL-операторе. Прикладная программа должна запрашивать идентификатор оператора прежде, чем она выдаст SQL-запрос. В то время как каждый идентификатор оператора связывается с единственным идентификатором соединения, каждый идентификатор соединения может иметь один или более связанных с ним идентификаторов операторов.

2. Соединение

При управлении выполнением прикладных программах, как только было назначено окружение, включающее свой идентификатор, могут быть назначены идентификаторы соединения. Аналогично, после назначения идентификатора соединения могут быть назначены идентификаторы операторов. С помощью этих функций вы можете установить

свое соединение с сервером базы данных.

3. Выполнение операторов SQL

Существует два способа определения и выполнения SQL-операторов: с предварительной подготовкой и непосредственный.

4. Получение результатов

Этот набор функций управляет восстановлением данных из результирующего множества SQL-оператора и восстанавливает такую информацию о результирующем множестве как: описание какого-нибудь столбца результирующего множества и его атрибутов, получение следующей строки результирующего множества, подсчет числа строк, на которые воздействует оператор SQL, и т.д.

5. Управление транзакцией

Эта функция позволяет завершить транзакцию или осуществить откат к началу транзакции.

6. Идентификация ошибок

Функция идентификации ошибок возвращает информацию об ошибке, которая связана с указанным идентификатором.

7. Смешанные функции

Смешанные функции в этой группе позволяют попытаться завершить выполнение SQL-оператора.

13 Технология COM

***COM* (Component Object Model) - это объектная модель компонентов**. Данная технология является базовой для технологий ActiveX и OLE. Технологии OLE и ActiveX - всего лишь надстройки над данной технологией. В качестве примера можно привести объект TObject, как базовый объект VCL Delphi. Точно так же технология COM является базовой по отношению к OLE и ActiveX.

Технология COM применяется при описании API и двоичного стандарта для связи объектов различных языков и сред программирования. COM предоставляет модель взаимодействия между компонентами и приложениями.

Технология COM работает с так называемыми COM-объектами. COM-объекты похожи на обычные объекты визуальной библиотеки компонентов Delphi. В отличие от объектов VCL Delphi, COM-объекты содержат свойства, методы и интерфейсы.

Обычный COM-объект включает в себя один или несколько интерфейсов. Каждый из этих интерфейсов имеет собственный указатель.

Технология COM имеет два явных преимущества:

1. создание COM-объектов не зависит от языка программирования. Таким образом, COM-объекты могут быть написаны на различных языках;
2. COM-объекты могут быть использованы в любой среде программирования под Windows. В число этих сред входят Delphi, Visual C++, C++Builder, Visual Basic, и

многие другие.

Хотя технология COM обладает явными плюсами, она имеет также и минусы, среди которых зависимость от платформы. То есть, данная технология применима только в операционной системе Windows и на платформе Intel.

Все COM-объекты обычно содержатся в файлах с расширением DLL или OCX. Один такой файл может содержать как одиночный COM-объект, так и несколько COM-объектов.

Ключевым аспектом технологии COM является возможность предоставления связи и взаимодействия между компонентами и приложениями, а также реализация клиент-серверных взаимодействий при помощи интерфейсов.

Технология COM реализуется с помощью *COM-библиотек* (в число которых входят такие файлы операционной системы, как OLE32.DLL и OLE-Aut32.DLL). COM-библиотеки содержат набор стандартных интерфейсов, которые обеспечивают функциональность COM-объекта, а также небольшой набор функций API, отвечающих за создание и управление COM-объектов.

В Delphi реализация и поддержка технологии COM называется *каркасом Delphi ActiveX* (Delphi ActiveX framework, DAX). Реализация DAX описана в модуле Axctris.

13.1 Развитие COM-технологий

Одной из важнейших задач, которые ставила перед собой фирма Microsoft, когда

продвигала операционную систему Windows, была задача по обеспечению эффективного взаимодействия между различными программами, работающими в Windows.
Самыми первыми попытками решить эту непростую задачу были буфер обмена, разделяемые файлы и технология динамического обмена данными (Dynamic Data Exchange, DDE).

После чего была разработана технология связывания и внедрения объектов (Object Linking and Embedding, OLE). Первоначальная версия OLE 1 предназначалась для создания составных документов. Эта версия была признана несовершенной и на смену ей пришла версия OLE 2. Новая версия позволяла решить вопросы предоставления друг другу различными программами собственных функций. Данная технология активно внедрялась до 1996 года, после чего ей на смену пришла технология ActiveX, которая включает в себя автоматизацию (OLE-автоматизацию), контейнеры, управляющие элементы, Web-технологию и т. д.

13.2 Терминология COM

Всякая новая технология приносит с собой новые термины для ее описания.

COM-объект

COM-объект представляет собой двоичный код, который выполняет какую-либо функцию и имеет один или более интерфейс.

COM-объект содержит методы, которые позволяют приложению пользоваться

COM-объектом. Эти методы доступны благодаря COM-интерфейсам. Клиенту достаточно знать несколько базовых интерфейсов COM-объекта, чтобы получить полную информацию о составе свойств и методов объекта. COM-объект может содержать один или несколько интерфейсов. Для программиста COM-объект работает так же, как и класс в Object Pascal.

COM-интерфейсы

COM-интерфейс применяется для объединения методов COM-объекта. Интерфейс позволяет клиенту правильно обратиться к COM-объекту, а объекту - правильно ответить клиенту. Названия COM-интерфейсов начинаются с буквы I. Клиент может не знать, какие интерфейсы имеются у COM-объекта. Для того чтобы получить их список, клиент использует базовый интерфейс lunknown, который есть у каждого COM-объекта.

Пользователь COM-объекта

Пользователем COM-объекта называется приложение или часть приложения, которое использует COM-объект и его интерфейсы в своих собственных целях. Как правило, COM-объект находится в другом приложении.

COM-классы

COM со-классы (coclass) - это классы, которые содержат один или более COM-интерфейс. Вы можете не обращаться к COM-интерфейсу непосредственно, а получать доступ к COM-интерфейсу через со-класс. Со-классы идентифицируются при

помощи идентификаторов класса (CLSID).

Библиотеки типов

COM-объекты часто используют библиотеки типов. *Библиотека типов* - это специальный файл, который содержит информацию о COM-объекте. Данная информация содержит список свойств, методов, интерфейсов, структур и других элементов, которые содержатся в COM-объекте. Библиотека типов содержит также информацию о типах данных каждого свойства и Типах данных, возвращаемых методами COM-объекта.
Файлы библиотеки типов имеют расширение TLB.

Технология DCOM

Технология DCOM (Distributed COM) - это распределенная COM-технология. Она применяется для предоставления средств доступа к COM-объектам, расположенным на других компьютерах в сети (в том числе и сети Internet).

Операционные системы Windows NT 4 и Windows 98 имеют встроенную поддержку DCOM.

Счетчики ссылок

Каждый COM-объект имеет счетчик ссылок. Данный счетчик содержит число процессов, которые в текущий момент времени используют COM-объект. Под процессом здесь подразумевается любое приложение или DLL, которые используют COM-объект, т. е. пользователи COM-объекта. Счетчик ссылок на COM-объект нужен для того, чтобы

высвобождать процессорное время и оперативную *память, занимаемую COM-объектом, в том* случае, когда он не используется.

После создания и обращения к COM-объекту счетчик ссылок увеличивается на единицу. Всякий раз, когда новое приложение подключается к COM-объекту - счетчик увеличивается. Когда процесс отключается от COM-объекта - счетчик уменьшается. При достижении счетчиком нуля память, занимаемая COM-объектом, высвобождается.

OLE-объекты

Часть данных, использующаяся совместно несколькими приложениями, называется *OLE-объектом.* Те приложения, которые могут содержать в себе OLE-объекты, называются *OLE-контейнерами* (OLE container). Приложения, имеющие возможность содержать свои данные в OLE-контейнерах, называются *OLE-серверами* (OLE server).

Составные документы

Документ, включающий в себя один или несколько OLE-объектов, называется *составным документом.* Приложение, которое может содержаться внутри документа, называется *ActiveX-документом* (ActiveX document).

Остальные термины, присущие технологии COM, мы рассмотрим в следующих разделах данной книги.

Состав COM-приложения

При создании COM-приложения необходимо обеспечить следующее:

- COM-интерфейс;
- COM-сервер;
- COM-клиент.

Рассмотрим эти три составляющие COM-приложения более подробно.

COM-интерфейс

Клиенты COM связываются с объектами при помощи COM-интерфейсов. *Интерфейсы* -- это группы логически или семантически связанных процедур, которые обеспечивают связь между поставщиком услуги (сервером) и его клиентом. На рис. 3.1 схематично изображен стандартный COM-интерфейс.

Рис. 3.1. COM-интерфейс

По правилам обозначения COM-объектов, интерфейсы COM-объекта обозначаются кружками справа или слева от COM-объекта. Базовый интерфейс lUnknown рисуется кружком сверху от COM-объекта.

Для примера, каждый COM-объект всегда поддерживает основной COM-интерфейс lUnknown, который применяется для передачи клиенту сведений о поддерживаемых интерфейсах.

Как уже говорилось выше, COM-объект может иметь несколько интерфейсов, каждый из которых обеспечивает какую-либо свою функцию.

Ключевыми аспектами COM-интерфейсов являются следующие:

1. Однажды определенные, интерфейсы не могут быть изменены. Таким образом, вы можете возложить на один интерфейс определенный набор функций. Дополнительную функциональность можно реализовать с помощью дополнительных интерфейсов.
2. По взаимному соглашению, все имена интерфейсов начинаются с буквы *I,* например IPersist, IMalloc.
3. Каждый интерфейс гарантированно имеет свой уникальный идентификатор, который называется *глобальный уникальный идентификатор* (Globally Unique Identifier, GUID). Уникальные идентификаторы интерфейсов называют *идентификаторами интерфейсов* (Interface Identifiers, IIDs). Данные идентификаторы обеспечивают устранение конфликтов имен различных версий приложения или разных приложений.
4. Интерфейсы не зависят от языка программирования. Вы можете воспользоваться любым языком программирования для реализации COM-интерфейса. Язык программирования должен поддерживать структуру указателей, а также иметь возможность вызова функции при помощи указателя явно или неявно.
5. Интерфейсы не являются самостоятельными объектами, они лишь обеспечивают доступ к объектам. Таким образом, клиенты не могут напрямую обращаться к

данным, доступ осуществляется при помощи указателей интерфейсов.

6. Все интерфейсы всегда являются потомками базового интерфейса Iunknown.

Основной COM-интерфейс *IUnknown*

Базовый интерфейс lunknown достаточно подробно был рассмотрен во второй главе книги. В дополнение ко всему вышесказанному, добавим, что интерфейс lunknown обеспечивает механизм учета ссылок (счетчик ссылок на COM-объект). При передаче указателя на интерфейс выполняется метод интерфейса lunknown AddRef. По завершении работы с интерфейсом приложение-клиент вызывает метод Release, который уменьшает счетчик ссылок.

При вызове метода Querylnterface интерфейса Iunknown в метод передается параметр IID, имеющий тип TGUID, т. е. идентификатор интерфейса. Параметр метода out возвращает либо ссылку на запрашиваемый интерфейс, либо значение NH. Результатом вызова метода может быть одно из значений, перечисленных в табл. 3.1.

Таблица 3.1. Значения, возвращаемые методом Queryinterface

Значение	Описание
S_OK	Интерфейс поддерживается
E_NOINTERFACE	Интерфейс не поддерживается
E_UNEXPECTED	Неизвестная ошибка

Указатели COM-интерфейса

Указатель интерфейса - это 32-битный указатель на экземпляр объекта, который является, в свою очередь, указателем на реализацию каждого метода интерфейса. Реализация методов доступна через массив указателей на эти методы, который называется vtable. Использование массива vtable похоже на механизм поддержки виртуальных функций в Object Pascal.

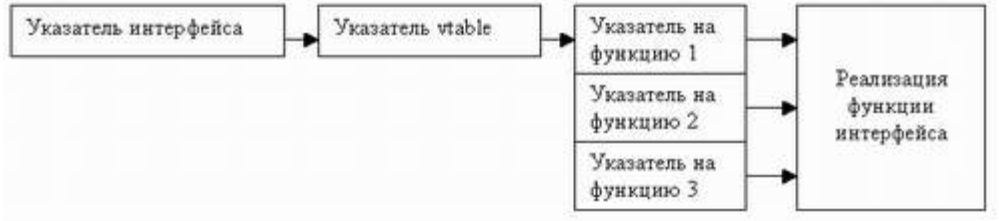


Рис. 13.1. Схема работы указателя COM-интерфейса

Наглядное представление работы указателей COM-интерфейса представлено на рис. 13.1.

COM-серверы

COM-сервер представляет собой приложение или библиотеку, которая предоставляет услуги приложению-клиенту или библиотеке. COM-сервер содержит один или более COM-объектов, где COM-объекты выступают в качестве наборов свойств, методов и интерфейсов.

Клиенты не знают *как* COM-объект выполняет свои действия. COM-объект предоставляет

свои услуги при помощи интерфейсов., В дополнение, приложению-клиенту не нужно знать, где находится COM-объект. Технология COM обеспечивает прозрачный доступ независимо от местонахождения COM-объекта.

Когда клиент запрашивает услугу от COM-объекта, он передает COM-объекту идентификатор класса (CLSID). CLSID - всего лишь GUID, который применяется при обращении к COM-объекту. После передачи CLSID, COM-сервер должен обеспечить так называемую *фабрику класса* (см. следующий раздел), которая создает экземпляры COM-объектов.

В общих чертах, COM-сервер должен выполнять следующее:

1. регистрировать данные в системном реестре Windows для связывания модуля сервера с идентификатором класса (CLSID);
2. предоставлять фабрику COM-класса, создающую экземпляры COM-объектов;
3. обеспечивать механизм, который выгружает из памяти серверы COM, которые в текущий момент времени не предоставляют услуг клиентам.

Фабрика класса

COM-объекты представляют собой экземпляры coclass. Напомним, что Coclass - это класс, поддерживающий один или более интерфейс. COM-объекты могут предоставлять только те услуги, которые определены в интерфейсах coclass. Экземпляры Cociass создаются при помощи специального типа объекта, называемого фабрикой класса.

Фабрика класса - это специальный COM-объект, который поддерживает интерфейс IclassFactory и отвечает за создание экземпляров того класса, с которым ассоциирована данная фабрика класса.

Интерфейс IclassFactory определен в модуле Delphi ActiveX так:

```
type
IClassFactory = interface (IUnknown)
['{00000001-0000-0000-COOO-000000000046}']
function CreateInstance (const unkOuter: lUnknown; const iid: TIID out obj):
HResult; stdcall;
function LockServer (fLock: BOOL): HResult; stdcall;
end;
```

Как видно из вышеприведенного куска кода, интерфейс имеет два метода: `CreateInstance` и `LockServer`.

Метод CreateInstance создает экземпляр COM-объекта ассоциированной фабрики класса.

Метод LockServer применяется для хранения COM-сервера в памяти. Если параметр метода fLock имеет значение true, то счетчик ссылок сервера увеличивается, иначе - уменьшается. Когда счетчик достигает значения о, сервер выгружается из памяти.

Всякий раз, когда услуги COM-объекта запрашиваются клиентом, фабрика класса создает и регистрирует экземпляр объекта для конкретного пользователя. Если услуга того же COM-объекта запрашивает другой клиент, фабрика класса создает второй экземпляр объекта для обслуживания второго клиента. coclass должен иметь фабрику

класса и идентификатор класса CLSID. Использование CLSID для cociass подразумевает, что они могут быть откорректированы всякий раз, когда в класс вводятся новые интерфейсы. Таким образом, в отличие от DLL, новые интерфейсы могут изменять или добавлять методы, не влияя на старые версии.

Мастер создания COM-объектов Delphi самостоятельно заботится о создании фабрики класса.

Локальные и удаленные серверы

С использованием COM клиент не должен беспокоиться о том, где располагается объект, он просто делает вызов интерфейса данного объекта. Технология COM обеспечивает все необходимые шаги для того, чтобы сделать этот вызов. Шаги могут отличаться, в зависимости от местонахождения объекта. Объект может находиться в том же процессе, где и клиент, в другом процессе на том же компьютере, где расположен клиент, или на другом компьютере в сети. В зависимости от этого применяются разные типы серверов:

1. внутренний сервер (In-process server);
2. локальный сервер или сервер вне процесса (Local server, Out-of-process server);
3. удаленный сервер (Remote server).

Внутренний сервер - это библиотека DLL, которая запущена в одном процессе вместе с клиентом. Например, элемент управления ActiveX, который внедрен на Web-страницу и

просматривается при помощи Internet Explorer или Netscape Navigator. В данном случае элемент управления ActiveX загружен на клиентскую машину и находится в том же процессе, что и обозреватель Web. Приложение-клиент связывается с сервером внутри процесса при помощи прямых вызовов COM-интерфейса. На рис. 13.2. представлена схема взаимодействия клиента с внутренним сервером.

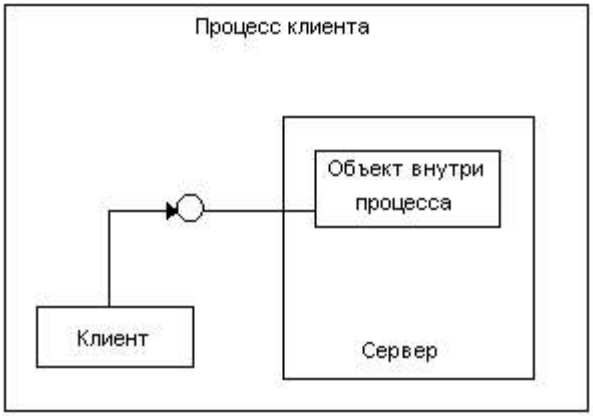


Рисунок 13.2 - Схема взаимодействия клиента с внутренним сервером

Внутренний COM-сервер должен экспортировать четыре функции:

```
function DllRegisterServer: HResult; stdcall;
function DllUnregisterServer: HResult; stdcall;
function DllGetClassObject (const CLSID, IID: TGUID; var Obj): HResult;
stdcall;
function DllCanUnloadNow: HResult; stdcall;
```

Все вышеперечисленные функции уже реализованы в модуле comserv, их нужно

только добавить в описания exports вашего проекта.

Рассмотрим данные функции более подробно:

1. DllRegisterServer - применяется для регистрации DLL COM-сервера в системном реестре Windows. При регистрации COM-класса в системном реестре создается раздел в HKEY_CZASSES_ROOTCLSID{XXXXXXXX-XXXX-XXXX-xxxx-xxxxxxxx}, где число, записанное вместо символов x, представляет собой CLSID данного COM-класса. Для внутреннего сервера в данном разделе создается дополнительный подраздел inProcserver32. В этом подразделе указывается путь к DLL внутреннего сервера (рис. 3.4).
2. DllUnregisterServer - применяется для удаления всех разделов, подразделов и параметров, которые были созданы в системном реестре функцией DllRegisterServer при регистрации DLL COM-сервера.
3. DllGetclassObject - возвращает фабрику класса для конкретного COM-класса.
4. DllcanUnloadNow - применяется для определения, можно ли в настоящий момент времени выгрузить DLL COM-сервера из памяти. Функция проверяет, есть ли указатели на любой COM-объект данной DLL, если есть, то возвращает значение S_FALSE, т. е. DLL выгрузить нельзя. Если ни один COM-объект данной DLL не используется, то функция возвращает значение SJTRUE.

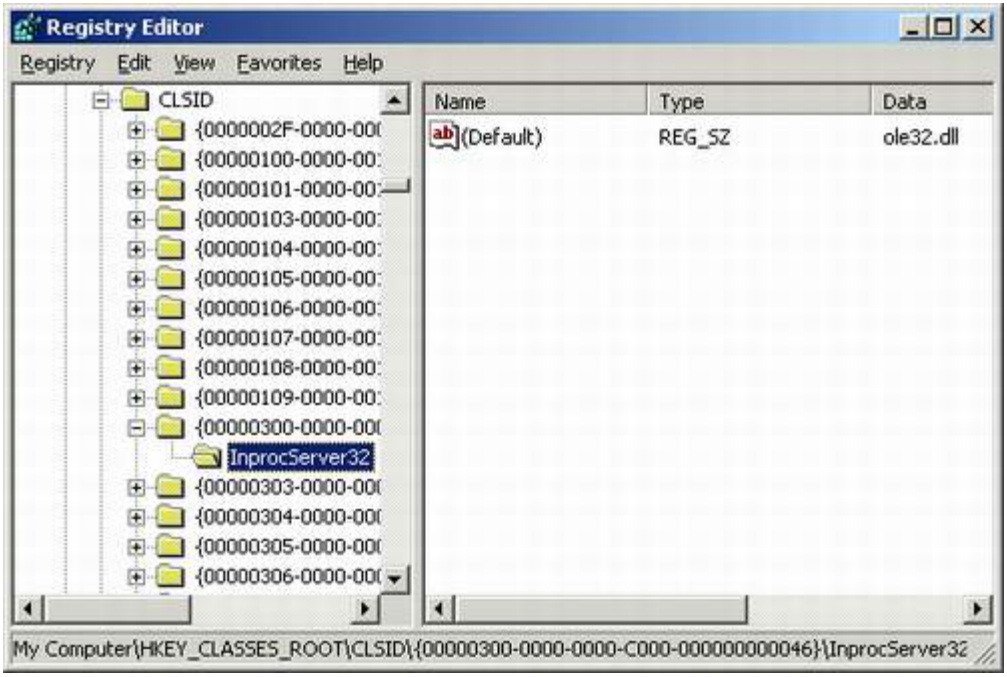


Рисунок 13.3 - Путь к локальному COM-серверу в окне редактора системного реестра

Локальный сервер - это приложение EXE, которое запущено в другом процессе, но на одном компьютере вместе с клиентом. Например, лист электронной таблицы Microsoft Excel связан с документом Microsoft Word. При этом два разных приложения работают на одном компьютере. Локальные серверы используют COM для соединения с клиентом.

Когда клиент и сервер находятся в различных приложениях, а также, когда они находятся на разных компьютерах в сети, COM использует *внутренний*

(внутрипроцессный) прокси (In-process proxy) для реализации процедуры удаленного вызова. Прокси располагается в одном процессе вместе с клиентом, поэтому, с точки зрения клиента, вызов интерфейсов осуществляется так же, как и в случае, когда клиент и сер'вер находятся внутри одного процесса. Задача прокси заключается в том, чтобы перехватывать вызовы клиента и перенаправлять их туда, где запущен сервер. Механизм, который позволяет клиенту получать доступ к объектам, расположенным в другом адресном пространстве или на другом компьютере, называется *маршалинг* (marshaling).

Функции маршалинга:

1. принимать указатель интерфейса из процесса сервера и делать указатель прокси в процессе клиента доступным;
2. передавать аргументы вызовов интерфейса таким образом, как будто они произошли от клиента и размещать аргументы в процесс удаленного объекта.

Для любого вызова интерфейса клиент помещает аргументы в стек, вызывает необходимую функцию COM-объекта через указатель интерфейса. Если вызов объекта произошел не внутри процесса, вызов проходит через прокси. Прокси упаковывает аргументы в *пакет маршалинга* и передает получившуюся структуру удаленному объекту. *Заглушка* (stub) объекта распаковывает пакет маршалинга, выбирает аргументы из стека и вызывает необходимую функцию COM-объекта.

Таким образом, маршалинг - это процесс упаковки информации, а демаршалинг -

процесс распаковки информации.

Тип маршалинга зависит от объектной принадлежности COM. Объекты могут использовать стандартный механизм маршалинга, предоставляемый интерфейсом IDispatch. *Стандартный маршалинг* позволяет устанавливать связь при помощи стандартного системного *удаленного вызова процедуры* (Remote Procedure Call, RFC).

На рис. 13.4 изображена схема, показывающая методику взаимодействия клиента и сервера в случае, когда приложения работают на одном компьютере, но в разных приложениях.

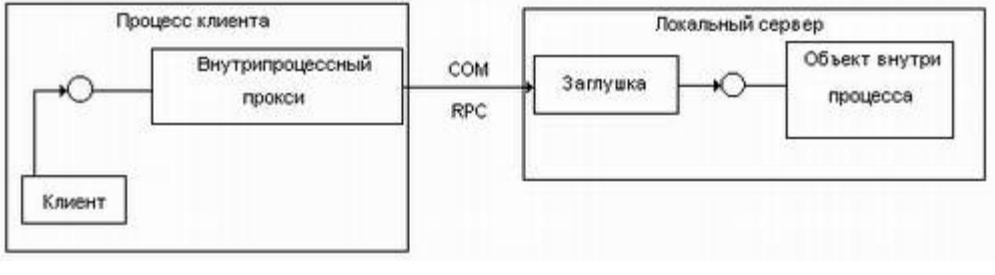


Рисунок 13.4 - Схема взаимодействия клиента с сервером в разных процессах на одном компьютере

Локальный COM-сервер регистрируется в системном реестре Windows так же, как и внутренний COM-сервер.

Удаленный сервер - это библиотека DLL или иное приложение, запущенное на другом компьютере. То есть клиент и сервер работают на разных компьютерах в сети.

Например, приложение базы данных, написанное с помощью Delphi, соединяется с сервером на другом компьютере в сети. Удаленный сервер использует *распределенные COM-интерфейсы* (Distributed COM, DCOM) для связи с клиентом.

Удаленный сервер работает также с помощью прокси. Различие в работе между локальным и удаленным сервером заключается в типе используемой межпроцессной связи. В случае локального сервера - это COM, а в случае удаленного сервера - DCOM. Схема взаимодействия клиента и удаленного сервера показана на рис. 13.5.

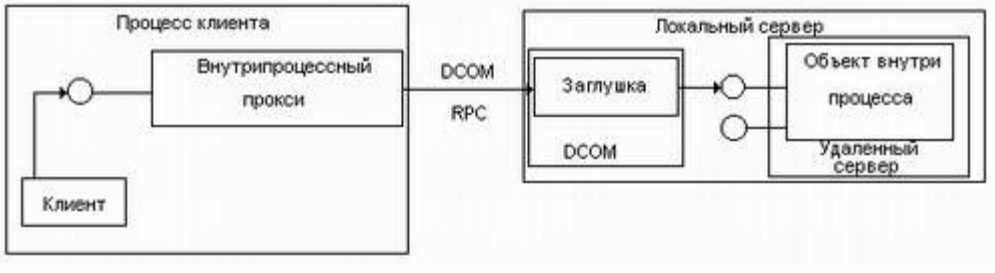


Рисунок 13.5 - Схема взаимодействия клиента с сервером на разных компьютерах

COM-клиенты

Очень важным при разработке COM-приложений является создание приложений, называемых COM-клиентами, которые могут запрашивать интерфейсы объектов, чтобы определить те услуги, которые может предоставить COM-объект.

Типичным COM-клиентом является диспетчер автоматизации (Automation Controller). *Диспетчер автоматизации* - это часть приложения, которая знает какой тип информации

необходим ему от.разных объектов сервера, и она запрашивает данную информацию по мере надобности.

Расширения COM

Технология COM изначально разрабатывалась как ядро для осуществления межпрограммного взаимодействия. Уже на этапе разработки предполагалось расширять возможности технологии при помощи так называемых расширений COM. COM расширяет собственную функциональность, благодаря созданию специализированных наборов интерфейсов для решения конкретных задач.

Технология ActiveX - это технология, которая использует компоненты COM, особенно элементы управления. Она была создана для того, чтобы работа с элементами управления была более эффективной. Это особенно необходимо при работе с приложениями Internet/Intranet, в которых элементы управления должны быть загружены на компьютер клиента, прежде чем они будут использоваться.

Технология ActiveX - не единственное расширение COM. В табл. 3.2 представлены некоторые из используемых в настоящее время расширений COM.

Перечисленные в табл. 13.1 расширения COM - это далеко не все из имеющихся. Постоянно идет доработка старых и создание новых, более совершенных технологий межпрограммного взаимодействия.

Таблица 13.1 - Список расширений COM

Расширение COM	Краткое описание
Серверы автоматизации (Automation servers)	*Серверы автоматизации*- это объекты, которыми можно управлять из других приложений во время работы приложения. Таким образом, *автоматизация*- это способность приложения программно контролировать объекты других приложений
Диспетчеры автоматизации или COM-клиенты (Automation Controllers, COM Clients)	*Диспетчеры автоматизации*- это клиенты серверов автоматизации. Они позволяют разработчику или пользователю писать сценарии для управления серверами автоматизации
Элементы управления ActiveX (ActiveX Controls)	Элементы управления ActiveX предназначены для серверов внутри процесса (in-process COM servers). Элементы ActiveX обычно используются путем встраивания в приложение-клиент
Библиотеки типов (Type Libraries)	Библиотеки типов представляют собой статичные структуры данных, которые часто сохраняются как файлы ресурсов. Они содержат детализированную информацию об объекте и его интерфейсах. Клиенты серверов автоматизации и элементы управления ActiveX используют данную информацию и всегда считают ее доступной
Страницы активного сервера (Active Server Pages)	*Активные серверные страницы*- это компоненты ActiveX, которые позволяют вам создавать динамически изменяющиеся Web-страницы
Активные документы (Active Documents)	*Активные документы* - это объекты, которые поддерживают связывание и внедрение, визуальное

	редактирование, перенос (drag-and-drop). В качестве примера таких документов можно представить документы Microsoft Word и книги Microsoft Excel
Визуальные межпроцессные объекты (Visual Cross-process Objects)	*Визуальные межпроцессные объекты*- это визуальные объекты, которыми можно управлять из других процессов

На рис. 13.6 представлена диаграмма, которая показывает связь некоторых расширений COM и их связь с технологией COM.

Использование COM-объектов имеет как преимущества, так и некоторые ограничения. COM-объекты могут быть как визуальными, так и невизуальными. Какие-то COM-объекты должны быть запущены в одном процессе с клиентом, другие - в разных процессах либо на разных,,компьютерах.

Приведенная ниже табл. 13.2 кратко описывает особенности объектов каждого из вышеприведенных расширений COM.

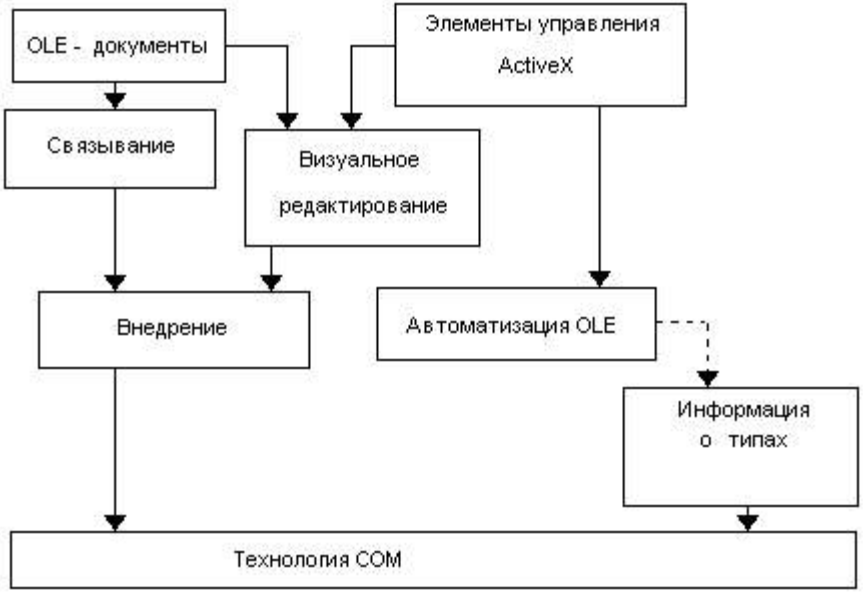


Рисунок 13.6 - Технологии, основанные на COM

Таблица 13.2 - Особенности объектов COM

COM-объект	Визуаль-ность	Процесс	Связь	Библиотека типов
Активный документ (Active Document)	Обычно визуальный	Внутренний или локальный	OLE	Нет
Автоматизация (Automation)	Может быть как визуальным, так и невизуальным	Внутренний, локальный или удаленный	Автоматический маршалинг при помощи интерфейса	Требуется для автоматического маршалмнга

			IDispatch	
Элемент управления ActiveX (ActiveX Control)	Обычно визуальный	Внутренний	Автоматический маршалинг при помощи интерфейса IDispatch	Требуется
Произвольный объект интерфейса	По выбору	Внутренний	Не требуется маршалинг	Рекомендуется
Произвольный объект интерфейса	По выбору	Внутренний, локальный или удаленный	Автоматический маршалинг в зависимости от библиотеки типов, в противном случае-ручной маршалинг	Рекомендуется

14 Сравнительный анализ технологий CORBA и COM

Данный обзор содержит сравнительный анализ двух наиболее популярных и комплексных систем создания распределенных приложений, а именно, CORBA консорциума OMG и COM (DCOM, COM+) фирмы Microsoft. Этот обзор предназначен главным образом для менеджеров проектов, руководителей информационных служб и др., т.е. для тех, кому приходится принимать ответственные, “стратегические” решения. Вследствие этого, в нем будут отсутствовать чисто технические аспекты обоих технологий, которые интересны только для разработчиков.

Кроме того, при написании обзора не ставилась задача сделать окончательный вывод типа “... таким образом, CORBA (COM), бесспорно, лучше, чем COM (CORBA)”. Связано это не с модной в наше время “политкорректностью” или отсутствием у автора своей точки зрения по этому вопросу. Дело даже не в том, что существуют определенные трудности с формализацией такого сравнения - я мог бы представить результаты сравнительных анализов, в которых используются численные оценки (баллы), выставленные экспертами, весовые коэффициенты и прочее, что придает отчету серьезный и весомый вид. Дело в том, что COM и CORBA можно сравнивать только с учетом определенных допущений. Отказ от таких допущений легко позволяет получить какой угодно результат. Вот эти допущения:

- CORBA, в отличие от COM’а, является концепцией, а не ее реализацией. Когда мы

говорим “COM”, то понимаем под этим скорее набор конкретных средств - элементов операционной системы, библиотек, утилит и т.п., являющихся составной частью того, что называется Microsoft Windows. Под термином “CORBA” понимается именно сложная и развитая концепция, сформулированная на уровне специального языка описаний - IDL. Реализации же этой концепции могут сильно отличаться друг от друга по различным критериям, наиболее важным в том или другом случае. Inprise/Corel VisiBroker и Application Server, BEA WebLogic, Iona Orbix, Oracle Application Server и “картриджи” Oracle, IBM BOSS - все эти продукты используют те или иные возможности CORBA. Технология Sun Enterpise JavaBeans создана поверх CORBA и использует такие ее возможности, как удаленный вызов объектов, службу имен, управление транзакциями. Реализация EJB фирмы Inprise связано с CORBA еще более тесным образом. Мы будем сравнивать COM и CORBA именно как концепции создания распределенных систем, а не как ту или иную их реализацию.

- При работе с реальным проектом необходимо учитывать область применения той или иной технологии. COM (как цельное и комплексное решение) способен функционировать только под Windows NT/Windows2000. Рассуждения о переносе COM на другие операционные системы в настоящий момент носят скорее рекламный характер. Ни компонентная модель COM (т.е. ActiveX), ни монитор транзакций

(Microsoft Transaction Server, MTS) не способны работать нигде, кроме Windows, и сама Microsoft не проявляет никакой активности в этом напрвлении (вопросами переноса тех или иных элементов COM на другие платформы занимается консорциум Open Group). CORBA является многоплатформенным решением просто по определению.

- Помимо чисто технологических аспектов, при принятии решения необходимо оценить затраты на закупку необходимого программного обеспечения, его сопровождения и обучение персонала. COM (в отличие от CORBA) официально является бесплатной технологией. Вы получаете все необходимое, просто приобретя, например, Windows NT Server. (Кстати, сам факт конкуренции дорогостоящей технологии с “аналогичной”, но бесплатной, многое говорит об их технических возможностях).
- Наличие готовых серверов приложений, пригодных для решения вашей задачи. Если Вы можете решить свои проблемы, используя функциональность Microsoft Office, то ничего лучше COM вы, естественно, не найдете.

Таким образом, главной задачей настоящего обзора является попытка помочь руководителю того или иного уровня принять квалифицированное решение в каждом конкретном случае. Поскольку и CORBA, и COM позиционируются соответственно OMG и Microsoft как универсальные технологии создания распределенных систем, мы будем

оценивать и сравнивать их именно с этой точки зрения. Предполагается, что для проекта используется платформа Windows (в противном случае нечего рассматривать COM) и имеется достаточно средств для закупки основных стандартных частей той или иной реализации (иначе обсуждение CORBA теряет смысл).

14.1 Концептуальный фундамент технологии COM

Технология создавалась фирмой Microsoft как средство взаимодействия приложений (в том числе составных частей операционной системы) Windows, функционирующих на одном компьютере, с последующим развитием для использования в пределах локальной сети. Главная задача на момент создания - обеспечение технологии Object Linking and Embedding (OLE 1.0). Характерно, что обмен данными между приложениями (Dynamic Data Exchange, DDE) первоначально строился не по COM-технологии, а с использованием механизма сообщений (messages). Развитие технологии идет по мере добавления новых возможностей. Как универсальная технология взаимодействия приложений COM начал использоваться с OLE 2.0 (1991). Концепция технологии неразрывно связана с ее реализацией. Появление новых возможностей - это просто появление новых библиотек, функций API и утилит Windows. “Общий знаменатель технологии” - двоичная структура объекта, хотя в настоящий момент существует язык описания структуры объекта - Interface definition Language (IDL).

CORBA

Технология создавалась консорциумом OMG как универсальная технология создания распределенных систем в гетерогенных средах. OMG представляет собой некоммерческую организацию, являющуюся содружеством разработчиков программного обеспечения и его потребителей, объединивших свои усилия для создания спецификаций этой технологии. В настоящий момент в OMG состоит более 800 членов, включая всех сколько-нибудь серьезных производителей программного обеспечения (и даже с недавнего времени Microsoft). Первая спецификация CORBA появилась в 1991 г. Новые возможности официально считаются добавленными в CORBA в момент утверждения соответствующей спецификации. Как правило, в разработке спецификации участвуют крупнейшие специалисты в данной области. Разработка реализации - задача конкретной фирмы. Обычно от утверждения спецификации до появления высококачественной реализации проходит довольно много времени - иногда несколько лет. “Общий знаменатель” технологии - объявления на языке IDL, который является “сердцем” CORBA с момента ее появления. (Существуют три различных языка описаний с одним и тем же названием - OSF IDL, Microsoft IDL и OMG IDL).

Выводы

Технология CORBA носит существенно более общий и универсальный характер, чем COM, что заложено в ее фундаменте. Опережение разработки спецификаций (по

сравнению с реализациями) позволяет добиться более связной, целостной и гармоничной системы. С другой стороны, при разработке реального проекта нужно предварительно убедиться, что высококачественная реализация того или иного сервиса CORBA уже доступна (источниками проблем могут служить, например, Persistence Service и Security Service).

14.2 Комплексность системы

COM

COM содержит все необходимое, что нужно для построения распределенной системы: технологию удаленного вызова методов (как статических, так и динамических), базы данных серверных объектов (библиотеки типов), которые могут быть импортированы для анализа структуры серверов COM, универсальный протокол обмена между клиентами и серверами, спецификации так называемых “составных документов” (ActiveDoc), объектный монитор транзакций (MTS), компонентную модель (ActiveX) и др. Все составные части прекрасно соответствуют друг другу в рамках модели COM. Уникальной возможностью COM является универсальная технология доступа к базам данных - OLE DB/ADO.

CORBA

В настоящий момент CORBA не имеет своей собственной компонентной модели; работа над ней началась в 1998 г. и еще не завершена. Это главный серьезный недостаток.

Правда, он несколько компенсируется наличием основанной на CORBA компонентной моделью Enterprise JavaBeans, так что программисты на Java находятся в привилегированном положении. Все остальное, что присутствует в COM, имеется и в CORBA, и даже более того - за исключением универсальной технологии доступа к БД. Опять-таки, Java-программисты имеют преимущество и здесь - за счет наличия общей для Java технологии доступа к данным JDBC.

Выводы

В настоящий момент COM более законченная система, но на более низком уровне и при существенно большем количестве ограничений, определяемых самой концепцией системы.

14.3 Используемые языки программирования

COM

Потенциально COM могут поддерживать самые различные языки программирования - все решает фирма Microsoft. Добавление некоторых расширений или экспертов (wizard) в систему разработки позволит использовать для работы с COM любой язык программирования. В настоящий момент наиболее широко используются Visual Basic, C++ и Delphi. Серьезные проблемы возникли при использования языка, на который возлагались особые надежды - с Java. Microsoft добилась прекрасного взаимодействия Java с COM, но достигнуто это было путем отказа от переносимости таких Java-программ

на другие виртуальные машины Java. Не случайно продукт фирмы Microsoft - J++ - не содержит в названии “Java”. Вообще, уровень стандартизации для COM достаточно слаб. Это не обязательно нужно рассматривать как недостаток - в конце концов, язык C лет пятнадцать прекрасно обходился без формального стандарта.

CORBA

Под “стандартом” применительно к CORBA понимается то, что официально утверждено консорциумом OMG. Надо сказать, что это очень высокий уровень “легитимности”, так как авторитет OMG в компьютерном мире чрезвычайно высок. В настоящий момент стандартизовано отображение языка IDL на 6 языков программирования - Ada, C, C++, Cobol, Java и Smalltalk. Существуют также отображения на Pascal (точнее, Delphi), Perl, Python и еще десяток языков.

Наиболее используемыми языками в настоящий момент являются Java (вследствие прекрасного взаимодействия Java-технологий, особенно JDBC, RMI, JNDI и EJB, с CORBA), и C++ - как самый эффективный, мощный и распространенный язык компьютерной индустрии.

Выводы

Обе технологии не испытывают особых проблем с точки зрения взаимодействия с языками программирования. Некоторые преимущества имеет CORBA - за счет более строгой стандартизации и более богатого выбора доступных средств разработки.

14.4 Уровень абстракции

COM

COM реализует высокий уровень абстракции - все вопросы низкого уровня, такие, как взаимодействия с операционной системой или сетевыми средствами, “спрятаны” от прикладного программиста.

CORBA

CORBA обеспечивает даже несколько более высокий уровень за счет базировании технологии исключительно на языке описания IDL с последующим отображением таких спецификаций на конкретный язык программирования, а также некоторых возможностей, например, автоматического (т.е. прозрачного для программиста) распространения контекста транзакций.

Выводы

Обе технологии реализуют примерно одинаковый и достаточно высокий уровень абстракций.

14.5 Поддержка компонентной модели

COM

Компонентная модель Microsoft, базирующаяся на COM-технологии, в основе имеет двоичную структуру объектов. Это не вызывает никаких проблем при ориентации на одну платформу и операционную систему. Безусловным достоинством такой модели является

простота создания компонентов с использованием различных языков программирования. С другой стороны, такая компонентная модель неизбежно связана с определенными ограничениями и недостатками. Одним из самых серьезных недостатков является то, что ее нельзя, строго говоря, назвать объектно-ориентированной.

CORBA

Как уже говорилось, CORBA в настоящее время не имеет своей компонентной модели. Пусть это не имеет практического значения для Java-программистов, но в общем случае эта та область, где OMG (и фирмам-производителям программного обеспечения) еще предстоит серьезно поработать.

Выводы

Это та область, где COM пока имеет существенные преимущества по сравнению с CORBA. С другой стороны, при разработке реальных проектов использование на стороне сервера компонентов Enterprise JavaBeans, построенных поверх инфраструктуры CORBA, предоставляет разработчику значительные преимущества по сравнению с компонентами ActiveX.

14.6 Универсальный протокол обмена

COM

Передача данных между клиентом и сервером основана на наборе типов, которые называются OLE Automation types. В принципе, схема маршалинга (в том числе типы

передаваемых данных) определяется парой стандартных интерфейсов COM. Реализовав по-своему некоторые из методов этих интерфейсов, можно определить свою схему маршалинга. Впрочем, это нетривиальная задача, и обычно разработчики используют уже готовую стандартную реализацию. Упаковка данных и их передача могут быть реализованы поверх различных сетевых протоколов.

CORBA

CORBA значительно более строго и формально подходит к механизмам обмена и передаче данных. Определен протокол CORBA (General Inter-ORB Protocol, GIOP) - и его реализация на базе протокола TCP/IP (Internet Inter-ORB Protocol, IIOP). CORBA способна передавать данные различных типов, включая структуры (struct) и объединения (union), в том числе содержащие рекурсивные определения. Предусмотрена система описания и контроля типов - как на уровне IDL-деклараций, так и динамическая. Для каждого языка используется свое отображение данных IDL. В версии 2.3 появился новый, заимствованный из RMI механизм передачи объектов CORBA - так называемая “передача по значению (objects-by-value)”. В предыдущих версиях CORBA при вызове удаленных методов объекты можно было передавать только по ссылке.

Выводы

CORBA предоставляет значительно больше возможностей, и эти возможности строго формализованы - в противном случае было бы невозможно обеспечить совместную

работу различных средств от различных производителей программного обеспечения.

14.7 Поддержка со стороны различных производителей и открытость

COM

В настоящий момент официальным “хранителем стандарта” технологии COM является консорциум Open Group, хотя “главным игроком на этом поле” является, конечно же, Microsoft. Поддержкой технологии COM занимаются многие небольшие фирмы - некоторые из них создают компоненты ActiveX, некоторые - как Software AG - занимаются переносом фрагментов COM на другие платформы, некоторые - как Borland - создают RAD-инструменты, основанные в том числе и на COM. В любом случае, только Microsoft решает, какая часть технологии и в каком виде попадает из лабораторий Microsoft в открытые спецификации.

CORBA

Ситуация с CORBA совершенно иная. CORBA является результатом совместных усилий огромного числа фирм, среди которых Sun, Oracle, IBM, Netscape/AOL, DEC/Compaq, JavaSoft, Borland/Visigenic (в настоящий момент в связи со слиянием Inprise и Corel принято решение о восстановлении имени Visigenic), BEA, Iona и многие другие. Можно сказать, что все производители программного обеспечения, которое должно функционировать на различных платформах и под управлением различных операционных систем, выбрали CORBA как стандартную инфраструктуру создания программных

продуктов. Естественно, все спецификации CORBA являются полностью открытыми. В лагере сторонников CORBA просматривается четкая тенденция к тесному взаимодействию и некоторой унификации используемых технологий (в качестве примера можно привести отказ Sun и ORACLE от создания собственного ORB и лицензирование Visigenic VisiBroker).

Выводы

COM и CORBA имеют совершенно разный уровень открытости и поддержки со стороны ведущих фирм в компьютерной индустрии.

14.8 Развитость сервисной части

COM

COM предоставляет минимальный набор совершенно необходимых средств - кодогенераторы с IDL, утилиты управления доступом (dcomcnfg) и др. Как правило, разработчики пользуются теми или иными инструментами, встроенными в конкретные средства разработки (примером может служит утилита работы с библиотеками типов или эксперт создания ActiveX-объектов в Borland Delphi/C++ Builder).

CORBA

CORBA имеет очень развитую сервисную часть; например, только для поиска серверных объектов по различным критериям можно использовать 4 различных сервиса CORBA. Кроме того, OMG стремится к максимальной стандартизации вспомогательных

возможностей CORBA.

Выводы

CORBA предоставляет разработчикам существенно большие возможности, чем COM, в области сервисов и вспомогательных средств. С другой стороны, COM-программисты обычно не испытывают какого-либо дискомфорта из-за их недостатка. Вследствие ограниченности области применения COM объективно нет необходимости в создании таких же развитых и универсальных средств, как это совершенно необходимо для CORBA.

14.9 Самодокументирование системы

COM

COM предусиатривает возможность создания локальной базы данных, хранящей информацию о COM-объектах, их интерфейсах, методах и т.д. для сервера приложений COM. Такая база данных называется библиотекой типов (type library). Использование библиотек типов не является обязательным для OLE, хотя это необходимо в технологии ActiveX. Для получения информации из type library пользователь должен явно импортировать ее, для чего необходимо выбрать соответствующую запись реестра Windows на конкретном компьютере.

CORBA

CORBA имеет аналог библиотеки типов COM - это так называемый Репозитарий

Интерфейсов (Interface Repository). Чтобы оценить принципиальную разницу в подходе CORBA по сравнению с COM, достаточно сказать, что Репозитарий Интерфейсов CORBA сам представляет из себя объект CORBA со всеми вытекающими из этого возможностями. Помимо Репозитария Интерфейсов, CORBA использует Репозитарий Реализаций (Implementation Repository), который представляет из себя базу данных, содержащую информацию о серверах приложений CORBA.

Выводы

Средства интроспекции (самодокументрования) CORBA значительно более развиты, мощны, гибки и универсальны, чем аналогичные средства COM. CORBA-программисты получают дополнительные преимущества по сравнению со своими коллегами, работающими с COM, при использовании Java (Java-технологии очень тесно связаны с CORBA). Связано это с тем, что Java предусматривает свой уровень интроспекции, дополнительный по отношению к CORBA.

14.10 Технология и описание проекта

COM

Обычно объявления на языке IDL при работе с COM не являются существенной частью спецификации проекта, так как IDL-спецификации играют вспомогательную роль в COM-технологии вследствие ее базирования на двоичном стандарте объектов. Кроме того, язык Microsoft IDL не очень развит с точки зрения объявления типов данных, из

которых строится программа.

CORBA

Проект с использованием CORBA всегда начинается с написания IDL-спецификаций (особый случай - использование Java, когда в принципе возможно генерировать стандартный код CORBA непосредственно на базе классов Java, хотя вряд ли это разумно для больших проектов.). Эти IDL-спецификации прекрасно отражают суть выполняемых действий с точки зрения функционирования распределенной системы. Синтаксис OMG IDL очень похож на синтаксис C++ (включая те же самые директивы препроцессора). Таким образом, IDL-спецификации могут с успехом использоваться как часть документации проекта.

Выводы

Описания OMG IDL более достоверны и существенно более наглядны, чем описания Microsoft IDL, в роли существенной части спецификации проекта.

14.11 Виды объектов

COM

Объекты COM всегда рассматривались (и продолжают рассматриваться) как серверные объекты, которые просто реализуют тот или иной набор методов. Различные объекты, реализующие один и тот же интерфейс и созданные с помощью обращения к одной и той же фабрике классов, не отличаются друг от друга. Объектная ссылка,

которую получает клиент, является указателем на интерфейс и не содержит информации о конкретном объекте. Другими словами, COM оперирует объектами, не имеющими состояния. В общем случае, если клиент, используя одну и ту же объектную ссылку, в цикле вызвал десять раз один и тот же метод, он не может быть уверен, что он обратился к одному, а не к двум, пяти или десяти различным объектам. Естественно, объекты без состояния не имеет смысла хранить дольше, чем существует сервер приложений, в котором они были созданы. Такие объекты применительно к распределенным системам называются временными (transient). COM-объект - это конкретная переменная C++, Visual Basic или Pascal, находящаяся в оперативной памяти и существующая, пока работает создавший ее сервер приложений. Он может быть создана по запросу клиента или заранее (например, с помощью MTS). При работе с COM сопоставить с объектом какое-либо состояние - задача прикладного программиста. Это можно сделать, используя определенный режим создания объектов (выбрав модель управления потоками), хранить состояние объекта на стороне клиента (если это возможно) или использовать так называемые моникеры, которые можно рассматривать как обобщение понятия ключа записи в базе данных. Каждый из этих способов имеет очень существенные ограничения.

CORBA

Понятие “объекта” в CORBA принципиально отличается от своего COM-аналога. Объект CORBA не является переменной языка программирования и в общем случае время

его существования не связано со временем работы серверных или клиентских приложений. CORBA-объект не занимает никаких ресурсов компьютера - оперативной памяти, сетевых ресурсов и т.п. Эти ресурсы занимает только так называемый “сервант” (servant), который является “инкарнацией” одного или нескольких CORBA-объектов. Именно сервант является переменной языка программирования. Пока не существует сервант, сопоставленный с конкретным объектом CORBA, этот объект не может обслуживать вызовы клиентов, но, тем не менее, он существует. Результатом создания объекта (при этом совершенно не обязательно при этом создается и сопоставляется с этим объектом соответствующий сервант!) является так называемая “объектная ссылка” CORBA. Объектная ссылка сопоставлена с этим, и только с этим объектом, и это сопоставление остается корректным в течение всего срока существования CORBA-объекта (может быть, в течение нескольких лет). Объектная ссылка CORBA правильно интерпретируется ORB’ами от любого производителя программного обеспечения. После уничтожения CORBA-объекта все объектные ссылки на него навсегда теряют смысл. С помощью объектной ссылки клиент вызывает методы объекта, при этом инкарнациями этого объекта могут быть различные серванты (не более одного одновременно), которые физически могут находиться даже на различных компьютерах.

Выводы

Объект COM может рассматриваться как достаточно примитивный частный случай

объекта CORBA - как по своим возможностям, так и с точки зрения его цикла жизни. COM и CORBA предлагают совершенно несопоставимые возможности по созданию и управлению объектами, что жизненно важно для создания надежных и масштабируемых приложений.

14.12 Способы взаимодействия

COM

COM поддерживает как статический, так и динамический способ взимодействия клиента и сервера. Под динамическим способом понимается подход, когда проверка, реализует ли сервер нужный метод, а также все необходимые действия по выполнению удаленного вызова выполняются на этапе работы клиентского приложения, а не этапе его компиляции. При статическом вызове эти действия выполняются именно на этапе компиляции. Разумеется, статический способ предпочтительнее во всех отношениях (когда он возможен вообще). Для его использования сервер COM должен создать и экспортировать библиотеку типов, которая затем импортируется клиентом и используется как часть клиентского процесса.

CORBA

CORBA также поддерживает статический и динамический способ организации удаленных вызовов.

Выводы

Обе технологии предлагают примерно одинаковые возможности в этом плане. CORBA имеет определенные преимущества при использовании динамических вызовов за счет более развитых средств получения информации о серверах (репозитарии интерфейсов).

14.13 Производительность

COM

COM демонстрирует очень высокую производительность. Читатель, интересующийся этим вопросом, найдет большое количество очень интересной информации в прекрасной книге R. Orfali и D. Harkey “Client/server Programming with Java and CORBA”, second edition, Wiley, 1998. Разумеется, производительность существенно зависит от того, какой способ - статический или динамический - вы используете.

CORBA

Для корректного сравнения CORBA и COM с точки зрения производительности необходимо составить целую систему тестов. Кроме того, необходимо учесть влияние использования того или иного языка программирования. На основе информации, приводимой Orfali и Harkey, а также результатов небольшого сравнительного тестирования, проведенного самим автором обзора (использовался Borland C++ Builder 4.0 и VisiBroker 3.3 для C++), можно сказать, что CORBA демонстрирует даже несколько более высокую производительность. Еще раз повторимся: производительность очень

сильно зависит от количества и типов аргументов методов (не забывайте, что их нужно упаковать и передать по сети, а затем распаковать), от выбранной модели управления потоками, от используемых языков программирования (клиент и сервер при этом не обязательно должны быть написаны на одном языке), от конкретной реализации CORBA и многих других факторов.

Выводы

И COM, и CORBA демонстрируют примерно одинаковую (и очень высокую) производительность. Для CORBA говорить о конкретных цифрах можно только для конкретной реализации. В качестве примера приведем следующий факт: Inprise/Visigenic Visibroker прозрачным для разработчика образом работает по-разному в зависимости от того, находятся ли клиентский и серверный объект в одном адресном пространстве, в разных адресных пространствах, но на одном компьютере, или на разных компьютерах. Производительность при этом может отличается на порядок.

14.14 Масштабируемость

COM

Проблемы обеспечения масштабируемости не были заложены в фундамент технологии, если не считать ориентацию на использование только объектов без состояния. Существенным препятствием для создания масштабируемых приложения является очень жесткая связь между клиентом и сервером (объект, т.е. совокупность ресурсов на сервере,

не может быть удален, пока клиент явно не укажет, что этот объект больше не нужен). В реальных проектах необходимо управлять состоянием объектов, и это затрудняет создание масштабируемых приложений, так как это обязанность не COM, а программиста. Сильной стороной COM является гибкая модель управления потоками. Основным инструментом, повышающим уровень масштабируемости COM-систем, является MTS.

CORBA

В отличие от COM, CORBA с самого начала рассматривалась как технология создания масштабируемых систем. Разделение собственно объектов CORBA и их сервантов, схемы соответствия между ними, характеристики объектных адаптеров, модели управления потоками и соединениями, схемы активации серверов приложений, универсальные решения по сохранению состояния объектов, автоматическое управление контекстом транзакций и безопасности - все это очень способствует решению данной проблемы.

Выводы

Масштабируемость системы во многом зависит от качества разработки проекта, продуманности принимаемых решений и квалификации менеджеров проекта и разработчиков. При сравнении технологий можно говорить о предпосылках, способствующих (или, наоборот, препятствующих) достижению нужных требований. При прочих равных условиях CORBA имеет громадные преимущества по сравнению с COM.

14.15 Устойчивость к сбоям

COM

Устойчивость к сбоям COM-систем находится на невысоком уровне, в том числе из-за уже упомянутой излишне жесткой привязки клиентов и серверов. Основным средством обеспечения устойчивости к сбоям (оно же средство управления нагрузкой серверов) является диспетчер, который позволяет перенаправлять вызовы клиента на различные сервера приложений COM. Не слишком содействует отказоустойчивости системы и необходимость выполнения “вручную” большого количества действий по управлению транзакциями.

CORBA

CORBA имеет несколько более высокий уровень устойчивости к сбоям за счет большей изоляции клиентов и серверов, автоматического сохранения состояния объектов, более мощной и продуманной схемы управления транзакциями (включая автоматический откат транзакций по тайм-ауту), а также автоматической привязки объектной ссылки и конкретного объекта CORBA.

Выводы

Проблема обеспечения устойчивости к сбоям, так же как и проблемы обеспечения масштабируемости, не рассматривались как первоочередные при разработке концепции COM. С CORBA ситуация обстоит во многом лучше, но проблемы остаются и здесь. Обе

технологии не имеют (или почти не имеют) стандарных средств обеспечения устойчивости к сбоям. Такие компоненты, как VisiBroker Smart Agents, не являются стандартным средством CORBA (хотя они и способны решить многие проблемы при работе с реальными проектами.)

14.16 Управление транзакциями

COM

Монитором транзакции в COM является MTS. Сервер приложений COM должен быть написан в специальном стиле для того, чтобы иметь возможность взаимодействовать с MTS (такой сервер приложений должен быть реализован в виде DLL). MTS позволяет достаточно гибко управлять режимами выполнения транзакций в системе и поддерживает двухфазное завершение транзакций. Одним из существенных недостатков схемы управления транзакциями COM является необходимость явной передачи контекста транзакции в качестве аргумента при вызове удаленных методов. Такая схема не является ни эффективной, ни гарантирующей от ошибок (особенно при вовлечении в транзакцию большого количества объектов).

CORBA

Управление транзакциями берет на себя так называемый Сервис Управления Транзакциями CORBA (Object Transaction Service, OTS). Он является существенно более гибкой, продуманной и формализованной системой, чем MTS, и содержит все

необходимое в рамках CORBA-модели. Сервер приложений CORBA и Сервис транзакций запускаются и работают независимо друг от друга. Важной особенностью CORBA является тесное взаимодействие OTS и ORB, что обеспечивает автоматическое распространение контекста транзакций в многопоточной распределенной среде. Спецификация CORBA предусматривает (необязательную) поддержку вложенных транзакций.

Выводы

На уровне спецификаций Сервис транзакций CORBA имеет определенные преимущества перед MTS. На практике для реализации этих преимуществ нужно предпринять определенные действия. Особенно это касается двухфазного подтверждения транзакций при работе с гетерогенными базами данных. Например, для реализации такой схемы при работе с Java необходимо иметь специальные JDBC-драйвера, которые, насколько мне известно, в настоящий момент не слишком доступны для широкого круга баз данных. В этом плане COM имеет серьезные преимущества за счет взаимодействия MTS со стандартной технологией доступа к базам данных OLE DB/ADO.

14.17 Обеспечение безопасности

COM

В настоящий момент система безопасности COM базируется на системе безопасности Windows NT/Windows 2000; кроме того, предусмотрена защита данных при

их передаче с использованием Socket Security Layer (SSL). Отдельная проблема - обеспечение безопасности при передаче компонентов ActiveX с использованием протокола HTTP. Здесь используется система электронных подписей, лицензий и т.п. - говоря упрощенно, клиент выполняет код компонента, который пришел с “правильного” сервера.

CORBA

С CORBA дела обстоят сложнее - главным образом, в силу того, что ставилась задача создать универсальную систему безопасности, которая могла бы использовать все основные существующие в этой области технологии. Работа над Сервисом Безопасности (Security Service) продолжалась в течение 2 лет, и ее спецификация была принята в 1996 г. Она содержит около 250 страниц. Она позволяет обеспечить уровень безопасности B2 (уровень, близкий к высшему уровню защиты, который используется в государственных учреждениях). Предусмотрена идентификация пользователя, списки прав доступа к ресурсам, система аудита и многое другое. Особенно приятно, что разработчик не должен явно взаимодействовать с этим сервисом - это задача для ORB. Основная нагрузка возложена на системных администраторов. Все это прекрасно, но существует одна небольшая проблема - где взять полномасштабную, высококачественную реализацию этого сервиса? Такие реализации существуют (Gradient, Concept-5), но их использование ограниченно за пределами США. Сервис безопасности от Borland/Visigenic в этом году

еще не появится (хотя работа над ним идет).

Выводы

В настоящий момент для реальных проектов для обеих технологий используются сходные решения в области обеспечения безопасности (идентификация на уровне операционной системы и кодирование информации с помощью SSL). Естественно, возможны варианты. Потенциально CORBA предоставляет существенно большие возможности - проблемы здесь организационного, а не концептуального плана.

14.18 Взаимодействие с Internet

COM

Основой взаимодействия через Internet при работе с COM являются расширения возможностей протокола HTTP, выполненные Microsoft. Броузеры Microsoft (Internet Explorer 3 и выше) позволяют выполнять код ActiveX-компонентов, полученных с Web-серверов. Кроме того, URL доступны при использовании COM - с ними могут работать моникеры.

CORBA

Спецификации CORBA не оговаривают использование Internet в качестве особого случая. Интеграция CORBA и Internet выполняется естественным образом - за счет использования протокола IIOP, построенного поверх TCP/IP. URL-имена могут быть использованы в качестве имен для Службы Именования CORBA. На практике

производители программного обеспечения предоставляют расширения CORBA, упрощающие работу с Internet (VisiBroker URL Naming Service) или решающие те или иные проблемы - например, “обход” ограничений, накладываемых на апплеты Java, используемых в качестве CORBA-клиентов (например, Borland/Visigenic GateKeeper).

Выводы

CORBA (особенно при использования Java) без каких-либо проблем может быть интегрирована с Internet. Взаимодействие COM и Internet основано на использовании ActiveX и требует использования только броузеров, поддерживающих тег <Object> Microsoft. Косвенным образом проблемы совместной работы COM и Internet могут возникнуть из-за несовместимости виртуальной машины Java Microsoft с другими виртуальными машинами.

14.19 Скорость разработки систем

COM

Скорость разработки COM-систем может быть очень высокой за счет интенсивного использования компонентной модели ActiveX, а также универсальных подходов, таких, как OLE DB. Не составляет особого труда создание Internet-приложений с броузером Microsoft в качестве клиентского приложения.

CORBA

Скорость разработки CORBA-систем сильно зависит от используемой технологии.

Наверное, максимально эффективным способом создания распределенных систем в настоящий момент является использование Java-технологий, основанных на CORBA - Enterprise JavaBeans и так называемых Application Server'ов, например, BEA WebLogic и Inprise Application Server. Использование этих технологий позволяет чрезвычайно быстро создавать высокоэффективные, масштабируемые, транзакционные сервера приложений. Клиентская часть таких систем может быть написана на любом языке программирования, поддерживающим CORBA.

Выводы

При прочих равных условиях CORBA позволяет создавать распределенные системы быстрее, чем COM, за счет большей функциональности middleware и, соответственно, меньшей нагрузки на прикладного разработчика.

14.20 Простота использования

COM

COM очень прост для простых небольших приложений и чрезвычайно сложен как инструмент создания комплексных систем. Он содержит большое количество "узких" мест - недостаточно гибкую стандартную схему маршалинга, отсутствие состояния объектов, низкая устойчивость к сбоям. Технология не является объектно-ориентированной в классическом смысле этого слова, что в общем случае не способствует простоте ее использования. Достоинством технологии является

комплексность и универсальность подходов в рамках COM-модели.

CORBA

Сложность CORBA заключается в ее огромных возможностях. Программисту необходимо знать большое количество интерфейсов из различных сервисов CORBA, правильно использовать возможности объектных адаптеров и многое другое. Поскольку CORBA использует различные схемы отображения IDL на разные языки программирования, то программисту в общем случае надо знать их особенности для 2-3 наиболее широко используемых языков - в первую очередь, C++ и Java.

Выводы

Объективно CORBA сложнее за счет того, что она предназначена для решения существенно более сложных задач, чем COM. При разработке реальных проектов нужно иметь в виду, что распределение “интеллектуальной” нагрузки среди участников разработки для COM и CORBA несколько отличается: в случае COM требуются более квалифицированные (но более узко специализированные) программисты, а для CORBA можно задействовать программистов среднего уровня, но чрезвычайно важно иметь квалифицированных архитектора проекта и руководителей групп программистов.

14.21 Взаимодействие с другими технологиями

COM

COM является достаточно замкнутой и "самодостаточной" системой. В последнее

время Microsoft тесно взаимодействует с OMG на базе создания спецификации моста “COM-CORBA”. Вследствие существенных различий в возможностях, не представляет труда имитировать поведение COM-объекта как CORBA-объекта, но не наоборот.

CORBA

CORBA как технология в настоящий момент (до создания спецификаций, а затем и реализаций своей компонентной модели) является скорее инфраструктурой для создания распределенных систем. Не удивительно, что в этом качестве она активно взаимодействует с другими технологиями - в первую очередь с RMI и Enterprise JavaBeans. CORBA очень тесно - на уровне протокола ESIOP - взаимодействует с широко используемой, но морально устаревшей технологией DCE.

Выводы

CORBA является существенно более открытой, универсальной и гибкой системой, чем COM. И COM, и CORBA способны тесно и эффективно взаимодействовать со стандартными средствами обеспечения безопасности.

14.22 Общие выводы

Несмотря на внешнюю похожесть, что вызвано общностью решаемых задач, между COM и CORBA, пожалуй, больше различий, чем сходства. В большинстве случаев либо нецелесообразно использовать CORBA (для небольших и простых проектов под Windows просто по причине относительно высоких затрат на приобретение программного

обеспечения, лицензий и пр.), либо практически невозможно использовать COM (для сложных, масштабируемых, высоконадежных проектов или просто при работе в гетерогенных средах, а не только в Windows). Windows-приложения, ориентированные на взаимодействие с Microsoft Office, всегда будут использовать COM; проекты с использованием Java и любых Java-технологий (кроме Microsoft J++), как говорится, “сам бог велел” строить на основе CORBA. Во многих случаях выбор технологии диктует выбор той или иной части проекта: если вы планируете работать, например, с ORACLE 8i, то, безусловно, гораздо лучше ориентироваться на CORBA. Область, где эти технологии реально конкурируют, на мой взгляд, очень невелика

15 О Java EE

Сегодняшние предприятия существуют в условиях мировой конкуренции. Им нужны приложения, отвечающие их производственным нуждам, которые, в свою очередь, с каждым днем усложняются.

В эпоху глобализации компании могут действовать по всему миру, имея представительства на разных континентах, работать в различных странах круглосуточно, без выходных, иметь по несколько центров обработки данных и международные системы, работающие с разными валютами и временными зонами.

При этом им необходимо сокращать расходы, увеличивать быстродействие своих сервисов, хранить бизнес-данные в надежных и безопасных хранилищах, а также иметь несколько мобильных и веб-интерфейсов для клиентов, сотрудников и поставщиков.

Большинству компаний необходимо совмещать эти противоречивые требования с существующими корпоративными информационными системами (EIS), одновременно разрабатывая приложения типа «бизнес — бизнес» для работы с партнерами или системы типа «бизнес — потребитель» с использованием мобильных приложений, в том числе с возможностью геолокации.

Довольно часто компании требуется координировать корпоративные данные, которые хранятся в разных местах, обрабатываются несколькими языками программирования и

передаются с помощью разных протоколов.

И конечно же, во избежание серьезных убытков необходимо предотвращать системные сбои, сохраняя при этом высокую доступность, масштабируемость и безопасность.

Изменяясь и усложняясь, корпоративные приложения должны оставаться надежными.

Именно для этого была создана платформа Java Enterprise Edition (Java EE).

Первая версия Java EE (изначально известная как J2EE) предназначалась для решения задач, с которыми сталкивались компании в 1999 году, а именно для работы с распределенными компонентами.

С тех пор программным приложениям пришлось адаптироваться к новым техническим решениям, таким как веб-службы SOAP и RESTful.

На сегодняшний день платформа Java EE отвечает этим техническим требованиям, регламентируя различные способы работы в стандартных спецификациях.

Спустя годы Java EE изменилась, стала насыщеннее и проще в использовании, а также более мобильной и интегрированной.

О Java EE

Сегодняшние предприятия существуют в условиях мировой конкуренции. Им нужны приложения, отвечающие их производственным нуждам, которые, в свою очередь, с каждым днем усложняются.

В эпоху глобализации компании могут действовать по всему миру, имея представительства на разных континентах, работать в различных странах круглосуточно, без выходных, иметь по несколько центров обработки данных и международные системы, работающие с разными валютами и временными зонами.

При этом им необходимо сокращать расходы, увеличивать быстродействие своих сервисов, хранить бизнес-данные в надежных и безопасных хранилищах, а также иметь несколько мобильных и веб-интерфейсов для клиентов, сотрудников и поставщиков.

Большинству компаний необходимо совмещать эти противоречивые требования с существующими корпоративными информационными системами (EIS), одновременно разрабатывая приложения типа «бизнес — бизнес» для работы с партнерами или системы типа «бизнес — потребитель» с использованием мобильных приложений, в том числе с возможностью геолокации.

Довольно часто компании требуется координировать корпоративные данные, которые хранятся в разных местах, обрабатываются несколькими языками программирования и

передаются с помощью разных протоколов.

И конечно же, во избежание серьезных убытков необходимо предотвращать системные сбои, сохраняя при этом высокую доступность, масштабируемость и безопасность.

Изменяясь и усложняясь, корпоративные приложения должны оставаться надежными.

Именно для этого была создана платформа Java Enterprise Edition (Java EE).

Первая версия Java EE (изначально известная как J2EE) предназначалась для решения задач, с которыми сталкивались компании в 1999 году, а именно для работы с распределенными компонентами.

С тех пор программным приложениям пришлось адаптироваться к новым техническим решениям, таким как веб-службы SOAP и RESTful.

На сегодняшний день платформа Java EE отвечает этим техническим требованиям, регламентируя различные способы работы в стандартных спецификациях.

Спустя годы Java EE изменилась, стала насыщеннее и проще в использовании, а также более мобильной и интегрированной.