НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ
"КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ"
Кафедра технической кибернетики

ЭЛЕКТРОННЫЙ КОНСПЕКТ ЛЕКЦИЙ ПО КУРСУ
"Корпоративные информационные системы и технологии"

для специальности
7.091402 Гнучкі комп'ютеризовані системи
та робототехніка

Киев - 2009 г.

СОДЕРЖАНИЕ (версия 2.0 на 7.06.2009г.)

4.3 Компоненты СЭД. Место СЭД в информационной системе предприятия
4.4 Типовые требования к СЭД
4.5 Технический документооборот и его особенности

5 Технологии обеспечения надежности корпоративных информационных систем
5.1 Аппаратные средства повышения надежности.
5.2 Программные средства обеспечения надежности
5.3 Основные методы обеспечения надежности серверов и сравнительный анализ некоторых из них

6. Современные и традиционные способы проектирования корпоративных информационных систем
6.1. Классификация CASE-средств
6.2. Каскадная схема разработки ПО
6.3. Спиральная модель проектирования
6.4. Методологии и технологии проектирования ИС
6.5 Консалтинг в области информационных технологий

7. Интеллектуальные технологии обработки информации в корпоративных сетях
7.1 Интеллект баз данных: Активные базы данных
7.2 Среда взаимодействия интеллектуальных агентов
7.3 Организация взаимодействия агентов в многокомпонентных САПР

8. Технологии интеграции современных сервисов в корпоративной сети - SOA.
8.1 Технологии SOA. Общие характеристики и проблеммы. GRID- среда.
8.2 Модель, ориентированная на сообщения (MOM).
8.3 Модель, ориентированная на сервисы (SOM). Web- сервисы и Grid- сервисы.
8.4 Модель, ориентированная на ресурсы (ROM). Особенности кластерных и суперкомпьютерных ресурсов. Центры Обработки Данных (ЦОД).
8.5 Новая технология интеграции ресурсов на базе виртуальных «Облаков»
Заключение
Литература

1 Введение в курс

Для корпоративных информационных систем, как для любых других, характерно волнообразное развитие. На первой волне (70-е годы) создавались в основном финансовые приложения, на второй (80-е годы) — системы планирования ресурсов предприятия (enterprise resource planning — ERP), на третьей (90-е годы) — системы управления отношениями с клиентами (customer relationship management — CRM). **Нынешняя, четвертая, обладает новым качеством. В информационных системах появляются самообслуживание и самоуправление, которые и могут быть реализованы средствами слабосвязанных приложений — и Web-службами в том числе. В результате эволюционного развития корпоративные системы превращаются в новый «зоопарк» — зоопарк приложений.**

Раньше подобным образом характеризовали комплекс разнородных аппаратных средств; со временем сетевые стандарты и инфраструктуры позволили связать их вместе. Однако корпоративный зоопарк не исчез, изменились его «обитатели», и ныне большинство компаний располагают набором приложений, который надо как-то упорядочить. Это могут быть старые унаследованные и плохо документированные приложения, а могут быть и современные коробочные продукты классов ERP, CRM и SCM (supply chain management — «управление цепочками поставок»), а также порталы. Обычно они представляют собой «черные ящики», которые работают каждый на собственном наборе данных, выполняют возложенные на них функции, но не могут обмениваться между собой данными в режиме реального времени и не образуют единую систему.

Развитие технологий многопользовательских распределенных БД, появление графических средств для проектирования интерфейсов пользователя, применение средств объектного программирования и мощных средств визуализации результатов породило новый виток в задаче приближения компьютерной среды к непрофессиональному пользователю, и, соответственно, увеличили разнообразие подходов и технологий для создания широкого спектра корпоративных систем.

В курсе "Корпоративные системы и технологии", читаемом для специальности 7.091402 ГНУЧКІ КОМП'ЮТЕРІЗОВАНІ СИСТЕМИ ТА РОБОТОТЕХНІКА рассматриваются различные сложившиеся и перспективные подходы в задаче создания современных информационных систем на основе классических и передовых Internet/Intranet-технологий и программно-аппаратных средств.

2 Корпоративные информационные системы и модели

Развитие рыночной экономики вызывает необходимость интенсивного использования современных компьютерных технологий любыми компаниями и фирмами, в каком бы направлении они не работали. Все они нуждаются в правильной организации работ, постановке планирования, эффективных технологиях поддержки управленческих решений, информационном обеспечении служб и руководителей, оформлению документации по национальным и международным стандартам (ISO 9000, SEI CMM, др.) и требованиям.

Для совершенствования управления компаниями, поддержки реализации функций корпоративного менеджмента, автоматизации бизнес- процессов предприятий и фирм широко применяются корпоративные информационные системы и современные Internet/Intranet технологии. С внедрением таких средств осваиваются новые технологии управления, проводится оптимизация бизнес- процессов.

Информационная система должна обеспечить инструментальную поддержку всей деятельности предприятия или фирмы и включает в себя программно-технические средства и средства их автоматизированной разработки и сопровождения (CASE-технологии). В конечном счете все данные, которые размещаются и поддерживаются в информационной системе, используются для решения задач учета, контроля и управления деятельностью предприятия. Эффективно увязать разнообразные бизнес- процессы фирмы можно только с использованием корпоративной информационной системы (КИС) и созданных на ее основе приложений.

Ядром и движущей силой любой такой системы являются системы баз данных. Несмотря на кажущуюся неизменность базовых технологий за последние 10-15 лет, в области реализации БД происходят значительные перемены, связанные с появлением хранилищ данных, применением объектно-ориентированного подхода, созданием "активных" БД и др. новых технологий.

Распределенность данных по корпоративной сети привело к появлению и развитию архитектур приложений типа "клиент-сервер" и дальнейшему развитию механизма транзакций, как основного средства управления операциями над данными в среде РБД и появлению средств репликации данных.

Следующим этапом развития КИС стало создание распределенных приложений, которые могли работать одинаково хорошо как с унаследованными системами, так и с различными по свей архитектуре источниками информации типа Web-серверов, поисковых систем и т.п. Появление таких приложений привело, в свою очередь, к появлению понятий "тонкого" и "сверхтонкого" клиентов, что создало условия для интеграции в корпоративной сети приложений, написанных на разных языках программирования. Перенос технологии "броузеров" из Internet в Intranet- среду позволяет в значительной мере сократить расходы на сопровождение приложений.

Потребности интеграции в корпоративной среде и корпоративного взаимодействия между фирмами и компаниями выдвинули на передний план разработку соответствующих стандартов и протоколов, которые бы позволяли стыковаться программно-аппаратным средствам различных фирм-производителей. В результате появились новые технологии и интерфейсы обмена данными типа DCOM, ODBC, OLE, DAO и т.п., что позволяет

динамически создавать интегрированный информационный ресурс предприятия.

Задачи управления предприятием породили также развитие разнообразных систем электронного документооборота (СЭД), включая бухгалтерско-экономические системы и средства автоматизации конструкторско-технологической документации.

Все большая степень интеграции информации и значительный рост источников данных привело к необходимости развития интеллектуальных средств и технологий обработки информации, в том числе, в корпоративных средах и сетях, что позволяет расширять круг пользователей информационных ресурсов, для которых ранее это было проблемным по многим причинам. Такие средства все больше находят свое применение в системах ERP, CRM SCM, которые развивают системы категорий B2B(baseness-to-baseness), B2C(baseness-to-customer) и другие, охватывая все большую часть человеческой деятельности.

2.1. Корпоративная модель данных

Примерно с 1987 г. опросы пользователей показали, что доступ к корпоративной информации стоит на первом месте в списке желаемых возможностей. Конечно, электронные таблицы и графические пакеты можно совершенствовать и дальше, но для многих пользователей они и так хороши. Большинство пользователей не испытывают трудностей с обработкой или эффективным представлением информации, они просто нуждаются в лучшем доступе к информации.

Проблема проиллюстрирована на рис. 3.1. В большинстве организаций данные расположены либо во внутреннем кольце, либо во внешнем.

Рис. 2.1. Распределение данных в организации

Центральная часть соответствует с большим трудом собранным и тщательно охраняемым данным, хранящимся в корпоративных БД. Рядовые пользователи могут получить доступ к этим данным с помощью специально созданных программ запросов, позволяющих ограниченно преобразовывать и вносить изменения в данные. Внешнее кольцо содержит все данные, хранящиеся на рабочих столах пользователей. Во многих компаниях количество этих данных приближается к объемам информации в центральных БД. Информация на рабочих столах в основном вводится вручную, используется абсолютно индивидуально и обычно очень быстро устаревает. На рисунке нет данных в среднем кольце. Среднее кольцо здесь символизирует информацию,

принадлежащую рабочей группе или отделу. Многие пользователи имеют возможность совместно использовать и преобразовывать эти данные, но в большинстве компаний этого среднего кольца нет.

Не существует легкого способа создать уровень данных рабочей группы, поэтому среднее кольцо представляет собой барьер, непроницаемую стену, отделяющую пользователей с их компьютерами от достоверных, свежих данных, хранящихся в корпоративном большом компьютере. Откуда взялся этот барьер? Почему Access (и Paradox, и dBase) не имеет доступа к корпоративным данным?

Без сомнения, компьютерная индустрия ощущает потребность пользователей в доступе к информации из корпоративных БД. Множество БД рабочего стола, таких как Access, Paradox, dBase IV и Approach, было создано с единственной целью – предоставить такой доступ. Эти продукты, в свою очередь, связаны с множеством продуктов межсетевых шлюзов, которые обычно работают на серверах для соединения, необходимого для обмена с большими компьютерами и их программными средствами БД. Paradox, Access и другие продукты взаимодействуют со своими пользователями посредством SQL (языка структурированных запросов). С его помощью продукты посылают запрос на большой компьютер, а затем принимают результат, скрывая при этом всю сложность процесса от индивидуумов, от которых исходил информационный запрос. Существующие в настоящий момент программные средства предоставляют пользователям доступ к информации, но сам доступ – новая, совершенно другая проблема.

При общении с аудиториями, состоящими из персонала, ответственного за БД на больших компьютерах, автору часто случалось спрашивать, как присутствующие отреагируют, если пользователям будет предоставлена возможность генерировать незапланированные запросы к БД, находящимся на их попечении. Обычным ответом было либо смущенное молчание, либо напряженное хихиканье. Почему? Потому что сама мысль о том, чтобы разрешить (не говоря уже о поощрении) пользователям делать запросы, генерируемые на лету, к работающей БД, присутствующим казалась дикой.

Большинство запросов, представляющих какой-либо интерес, содержит составные данные в той или иной форме. Составные данные представляют собой сумму, среднее арифметическое, счет или любую другую функцию, которая обязывает приложение производить арифметическое действие над каждой записью в БД. Кто крупнейший покупатель? Сколько красных шляп компания продала в прошлом году? Каков средний объем продаж у торговых агентов на данной территории? Все эти вопросы и многие другие требуют вычислений составных данных относительно полных БД или их

существенных частей. Проблема в том, что производственные БД совместно используются сотнями или тысячами пользователей и все они работают с одним большим компьютером. Незапланированные запросы очень сильно загружают большие компьютеры, так как часто требуют доступа к большей части БД. Но когда большой компьютер перегружен, бизнес буквально останавливается. Хуже всего то, что когда пользователи начинают создавать свои собственные запросы, невозможно предсказать степень загруженности компьютера. В результате большой компьютер не только останавливается, но невозможно определить заранее вероятность такой остановки. Очевидно, такая ситуация неприемлема в производственной среде. Таким образом, на практике dBase, Access и Paradox не предоставляют пользователям доступа к реальным данным.

2.2.3 Архитектуры прикладных систем

В таблице 1 перечислены наиболее часто встречающиеся архитектуры прикладных систем. Колонка «максимальное число пользователей» может рассматриваться как некоторая мера масштабируемости. Все архитектуры, кроме первой, являются архитектурами распределенных систем.

ТИП АРХИТЕКТУРЫ	МАКСИМАЛЬНОЕ ЧИСЛО ПОЛЬЗОВАТЕЛЕЙ	ДОСТОИНСТВА	НЕДОСТАТКИ
Централизованная	Тысячи	Простота	Ориентация на конкретную аппаратуру; недостаточная гибкость
Разделение файлов	Десятки	Простота	Чрезвычайная ограниченность
Двухзвенная	Сотни	Достаточная простота; очень мощные инструменты разработки	Слишком малое число пользователей
Трехзвенная	Не ограничено	Высокая масштабируемость; очень высокая производительность и надежность; основа для многократного использования существующих компонентов; гибкость	Большая сложность с точки зрения разработки

Как можно заметить, самая «древняя», централизованная архитектура обеспечивает намного более масштабируемое решение, чем две следующих. Потребовалось три поколения, прежде чем эти распределенные архитектуры смогли эффективно конкурировать с решением, ориентированным на универсальный компьютер: построить распределенную систему гораздо труднее.

Централизованная архитектура

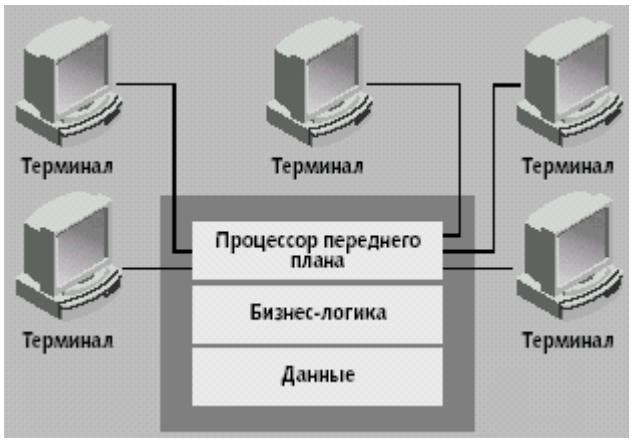


Рис. 2. Централизованная архитектура. Блок «Данные» включает алгоритмы доступа к данным, СУБД и саму базу данных

Одна мощная универсальная ЭВМ была единственной платформой, выполняющей все алгоритмы логики приложения (рис. 2). У централизованной архитектуры множество достоинств: простая разработка приложений, легкость обслуживания и управления. Именно они и обеспечили столь долгую жизнь «унаследованных» систем. Смерть универсальных ЭВМ неоднократно провозглашалась после появления четырех- и восьмиразрядных ПК в начале 80-х годов (компьютеры PET и VIC-20 компании Commodore, TRS-80 компании Radio Shack и множество других машин на базе процессоров Z-80, а также 6502 и 6800 производства Motorola). Однако они продолжали работать, переваривая десятки миллионов транзакций в день, приспосабливаясь к постоянно увеличивающимся нагрузкам.

Разделение файлов

Рис. 3. Архитектура разделения файлов

Едва появившись, ПК принесли ожидание того, что большое число маленьких машин может заменить, а, в некоторых случаях, и превзойти по производительности универсальную ЭВМ. Архитектура разделения файлов, ставшая первым шагом к реализации этого притязания, включает множество настольных ПК и файловый сервер, связанных сетью (рис. 3). Файловый сервер загружает файлы из разделяемого местоположения, а прикладные программы исполняются полностью на настольных ПК.

Подобная архитектура была особенно популярна при использовании продуктов наподобие dBASE, FoxPro и Clipper. Первоначально сети персональных компьютеров были основаны на метафоре совместного использования файлов, потому что это было просто. Однако она хорошо работала лишь в некоторых случаях. Во-первых, все приложения должны вписаться в единственный ПК. Во-вторых, совместное использование и конфликты обновления чрезвычайно снижают производительность. Наконец, учитывая пропускную способность сети, объем данных, которые могут передаваться, также невелик. Все эти факторы крайне ограничивают число параллельных пользователей, которое способна поддерживать архитектура разделения файлов [6-8].

2.3 Модель для моделирования приложений в информационных системах

Разрабатывая новую основу для проектирования и построения приложений, необходимо начать с модели общего процесса. Эта модель базируется на конкретной архитектуре приложения. Действительно, невозможно более выразительно выделить важность поддержки понимания архитектуры приложений на каждой стадии обдумывания крупных приложений. Напоминание об этом уже было и до применения архитектуры приложения может встретиться много раз.

Рис. 14.3 представляет трехслойную архитектуру приложения как трехстадийный процесс – планирование, проектирование, разработка. Этот процесс концептуально очень важен; термины, использованные для этих стадий, максимально приближены к IE. Рассмотрим детально эти стадии, одну а одной:

	Концептуальная	Логическая	Физическая
Документы	Поток работ	Поток форм	Формы
Правила бизнеса	Поток процессов	Модель компонентов	Программы
База данных	Модель данных	Схема БД	Таблицы, индексы

Рис. 14.3. Этапы планирования, проектирования и разработки приложений

- *Концептуальная.* Первой стадией в любом проекте является обзор требований и разработка общего проекта. Это эквивалентно ранним планам этажа, с множеством позиций от плана к плану, которые архитектор использует для начала грубого проектирования. В слое документа этот проект рассматривает обширные потоки работ от офиса к офису и от персоны к персоне, без учета любых детальных форм или интерфейсов. На уровне процесса рассматриваются высокоуровневые кружки и линии, которые вышестоящее руководство и консультанты по

управлению так любят рисовать. На уровне БД рассматривается высокоуровневая, интегрированная основа предприятия и подразделений.

- *Логическая*. Данное проектирование принимает во внимание детальные правила описания процесса и улучшает проект, показывая, как эти правила можно приспособить. На уровне процесса высокоуровневые объекты разбиваются на более детальные процессные действия и диаграммы запрос/действие, которые показывают, как более крупные блоки работают. На уровне рабочего стола должны быть смакетированы последовательности спроектированных форм, показывающие точную последовательность шагов, необходимых для завершения конкретных задач. На уровне БД высокоуровневая модель приобретает вид системы логически связанных реляционных таблиц с использованием программных триггеров и хранимых процедур конкретной СУБД.
- *Физическая*. Представляет собой этап проектирования, на котором логическая модель данных погружается в установленную СУБД на конкретных программно-аппаратных средствах сервера БД, а приложение - на рабочую станцию и (для трехуровневых приложений) сервер приложений.

2.4 Сетевые базы данных. Основные функции СУБД

Основные концепции обработки сетевых баз данных в среде локальной вычислительной сети сформировались в 1988-1989 гг. и заключаются в следующем:

а) распределенная (сетевая) база данных организуется на основе модели взаимодействия «рабочая станция – сервер». При этом база данных общего доступа размещается в одном или нескольких серверах. Пользовательские (локальные) базы данных хранятся и обрабатываются локально на рабочих станциях;

б) сетевые операции доступа к базе данных, обмена по сети фрагментами баз данных, обеспечение надежности и целостности базы данных реализуется на базе специального механизма транзакций и соответствующего протокола выполнения транзакций;

в) стандартным средством обслуживания базы данных, доступа к ней и получения результатов является SQL – стандартный реляционный язык пользователя для взаимодействия с базой данных;

г) центральным исполнительным элементом сетевой базы данных является SQL-сервер, реализующий высоконадежный протокол обработки транзакций, сетевые операции доступа к базе данных и реляционный язык запросов.

В результате применения этих концепций проектировщики информационных систем получают в свое распоряжение новую архитектуру распределенных информационных систем (РИС). Для современных систем класса РИС характерными свойствами являются:

- высокий уровень развития сетевых средств организации и управления базами данных: а) возможность распределения баз данных между файл-серверами; б) организация дублированных копий баз данных и обеспечение высокой надежности хранения информации; в) широкие возможности распределения рабочих станций между производственными участками без изменений в архитектуре и режимах использования РИС;
- высокий уровень средств пользователя (конечного пользователя, администратора системы и оператора) при работе с базой данных: а)

язык запросов 4-го поколения (типа SQL); б) прозрачность доступа к сетевой базе данных; в) системная поддержка целостности базы данных, помехоустойчивости и высокой интегрированности узловых фрагментов базы данных;

- высокий уровень развития инструментальных средств организации баз данных и создания приложений: а) языки программирования высокого уровня; б) словари/справочники данных; в) средства анализа производительности и эффективности использования дискового пространства.

Рассмотрим реализацию указанных концепций и свойств в совместном проекте фирм Ashton-Tate, Microsoft and Sybase – системе SQL Server.

SQL-Server

Данная система относится к классу реляционных СУБД, обеспечивающих многопользовательскую обработку распределенной базы данных. База данных функционирует в среде локальной сети ПЭВМ под управлением Oracle, Informix,OS/2 или MS SQL-server.

SQL-server основан на архитектуре «клиент – сервер», в которой могут поддерживаться «фронтальные» буфер-процессоры для различных пользовательских СУБД, табличных процессоров и расчетных программ. Фронтальные процессоры – это прикладные программы, действующие на стороне пользователя, т.е. отвечающие за функции клиента в модели «клиент – сервер».

Фронтальные процессоры действуют, как правило, на рабочих станциях и манипулируют локальными данными. Вторую компоненту базовой модели «клиент – сервер» реализует процессор заднего плана (back-end), т.е. компонент в модели «клиент – сервер».

В рекламных материалах по SQL-server объявляется, что совместно с SQL-server могут работать такие известные системы как Paradox, DataEase, DB/ACCESS, Lotus 1-2-3, MDBS, OpenAccess, Clipper и др. Доступ к различным прикладным системам осуществляется при помощи SQL-языка, средствами которого осуществляется доступ к базе данных сервера и манипулирование данными. В отличие от обычных файл-серверов SQL-сервер выполняет детальный поиск данных, сортировку, арифметическую обработку и полный

набор логических операций. В результате этого существенно сокращается трафик по пересылке наборов данных между сервером и рабочей станцией.

Еще одно преимущество SQL-сервера связано с возможностью реализации нескольких серверов баз данных на одной локальной сети. Благодаря этому все базы данных и отдельные базы могут быть дублированы на нескольких файл-серверах. Таким образом, повышается общая надежность сетевой базы или реализуются специальные технологии обслуживания базы данных.

2.5 Понятие транзакции. Механизм транзакций в СУБД

Значительные достоинства сетевых баз данных связаны, прежде всего, с высоким уровнем стандартизации основных архитектурных решений и технологий. Проектировщики сетевых баз данных освобождаются от необходимости решать многие задачи структурно-функционального характера: выбор соотношений между центральными и локальными операциями обработки баз данных, синхронизация распределенных процессов обработки данных, защита данных и поддержка целостности данных, организации коммуникаций в локальной сети и т.п. Это означает, что при проектировании сетевой базы данных проектировщик может больше внимания уделять прикладным вопросам представления информации в базе данных, технологии решения прикладных задач, вопрос организации эффективного сервиса для конечных потребителей.

Например, главными сетевыми стандартами баз данных в среде NetWare являются: а) файловая система, механизмы защиты данных, управление многопользовательским доступом и протоколы передачи данных – собственные механизмы операционной системы NetWare; б) единая архитектура сетевых взаимодействий на базе модели «клиент – сервер»; в) стандартный серверный механизм, обслуживающий многопользовательский доступ к базе данных, стандартный язык SQL; г) стандартный механизм обработки транзакций.

Аналогичным образом построены другие сетевые системы, которые используют ОС типа UNIX, Linux, Windows с такими механизмами, как обработка процессов, *.dll-библиотеки, Registry.

Таким образом, *механизм транзакций* – это общий метод сетевого управления последовательностью операций над базой данных. *Транзакция* может быть определена на уровне SQL-команд, например, в виде некоторой цепочки SQL-команд по доступу к базе данных, чтению определенных записей и корректировке данных. Механизм транзакции обеспечивает выполнение всей транзакции (т.е. всех входящих в нее операций) от начала до конца. Если после выполнения транзакции возникают сбои (отказы) или тупиковые ситуации, то транзакция «откатывается» назад и состояние базы данных не меняется.

Возможности администратора баз данных

Администратор сетевой базы данных отвечает за ее общую доступность, целостность и сохранность информации. При этом важно обеспечить такие режимы административного обслуживания базы данных, которые в минимальной степени затрудняют или прерывают работу пользователей и разработчиков.

Серверы баз данных реализуют эти требования на основе ряда новых концепций:

Динамическое архивирование данных. Архивные копии базы данных создаются без приостановки обработки текущей базы. Команды типа DUMP DATABASE воспринимается SQL-сервером и реализует мгновенную «фотографию» базы данных.

Автоматическое восстановление баз данных. В случае серьезного физического сбоя или отказа аппаратуры база данных восстанавливается на основе последней копии и журнала базы данных. SQL-сервер реализует автоматическое восстановление информации (автоматическое восстановление также выполняется при каждом запуске SQL-сервера).

Мониторинг и управление работой SQL-сервера. Администратор использует различные утилиты, а также специальные команды SQL для создания групп пользователей, назначения паролей доступа, измерения активности использования ресурсов и выбора опций и параметров обработки баз данных.

Обеспечение защиты данных. Администратор имеет средства контроля доступа к данным и ограничения доступа (на уровне таблиц, записей, подсхем и т.п.). Вся информация, контролирующая доступ, хранится в он-лайн словаре/справочнике данных. Фактически эти средства расширяют аналогичные «штатные» возможности NetWare, Ethernet, TolkenRing и др..

2.6 Многопользовательский доступ к СУБД

СУБД коллективного доступа осуществляют одновременный доступ многих пользователей к данным в клиент-серверной архитектуре приложений с использованием многих механизмов, которые обеспечивают целостность и защиту данных, из которых можно выделить следующие:

Клиент СУБД. Специальная программа доступа к СУБД, использующая "закрытый " интерфейс, через который (и только через него!) проходят SQL-запросы от приложений к БД.

Потоковый обмен (pipes). Этот механизм используется для координированного взаимообмена сообщениями. Пайп открывается станцией, получает определенный статус и может быть открыт другими станциями, желающими вступить во взаимодействие. Каждая станция может организовать очередь сообщений для обмена через пайп (до 6 сообщений). Потоковый обмен ведется по запросу принимающей станцией, причем как принимающая, так и отправляющая станции могут закрыть пайп в любой момент времени.

Семафоры. Обеспечивают механизмы синхронизации процессов. Семафоры создаются и обслуживаются файл-серверами и поэтому доступны для каждой рабочей станции. При помощи семафоров могут решаться задачи координации доступа к файлам, управления вызовом программ, контроля выполнения различных процессов в сети. Семафоры – это именованные объекты, которые открываются и закрываются со стороны рабочей станции. К уже открытым семафорам могут иметь доступ другие станции, которые обращаются к семафору только по имени.

Семафоры – это универсальный и исключительно гибкий механизм для управления работой распределенных сетевых процессов.

Интеграция системных и прикладных программ

Прикладные программы (помещаемые в среду NetWare, Ethernet, TalkenRing и др.), в архитектуре "клиент-сервер" могут иметь ряд свойств, которые необходимо поддерживать (обслуживать) средствами сетевых управляющих программ. К таким свойствам относятся: а) разделяемость прикладной программы; б) использует ли прикладная программа разделяемые данные; в) есть ли данные, которые не должны быть доступны другим программам.

Например, NetWare обеспечивает интеграцию прикладных программ и поддержку различных свойств на уровне системного супервизора. Пользователь может в

программе установки снабдить супервизор установочными параметрами: директориями и параметрами доступа к файлам, именами групп пользователей файлов и их размещений. В прикладных программах пользователя часто реализуется концепция супервизора как единого координирующего и управляющего механизма. При этом пользователь должен решить вопрос: использовать ли общий системный супервизор NetWare или разрабатывать свой «прикладной» супервизор. Предпочтительным является первый подход, при котором прикладная программа базируется на стандартном сервисе NetWare.

В состав системного сервиса входят: а) средства программирования на языке высокого уровня; б) системные диагностические сообщения; в) системные утилиты; г) командный файл. На уровне языка программирования техника интеграции системных и прикладных программ должна учитывать три аспекта: метод передачи параметров, формат системных областей памяти и внутренняя буферизация. На уровне диагностики прикладная программа взаимодействует с таблицами и сообщениями супервизора, анализирует операционную среду, выдает пользователям только определенные диагностические сообщения. На уровне командного файла формируются обращения прикладной программы к системным утилитам и специальным контрольным операциям.

Полный перечень современных технологий интеграции в современных корпоративных сетях описан в разделе 3.4 Конспекта.

3. Многоуровневые информационные системы. Клиент- серверные технологии.

Обыкновенно для небольших организаций разработчики применяют двухзвенную архитектуру клиент-сервер, когда с рабочих станций осуществляется удаленный доступ к БД, и не более того. В самых простых, примитивных системах даже не используются возможности ,предоставляемые РСУБД, такие, как триггеры и сохраненные процедуры, и, хотя разработчики именуют подобные системы клиент-сервером, они имеют весьма мало общего с истинными распределенными приложениями. Более того, идеология “толстого” клиента принуждает к установке на рабочих местах весьма дорогостоящих Wintel-компьютеров, потому что на них производятся все основные вычисления, и обмен данными с удаленным сервером БД производится сквозь толстый многоуровневый слой драйверов, которые должны быть установлены на персональной рабочей станции и лицензированы их разработчиками для каждого рабочего места. Иногда получается совершенно нелепая вещь: если пропускная способность сети недостаточно велика или недостаточно эффективно организован поток прохождения транзакций, то при этом быстродействующие процессоры клиентских машин совершенно бездействуют, а в обратном случае, наоборот, сервер БД задыхается и не успевает отвечать каждому из многочисленных и буквально забивающих его клиентов. При числе одновременно работающих клиентов более 30 необходимо переходить на трехзвенную архитектуру (рис. 3.1).

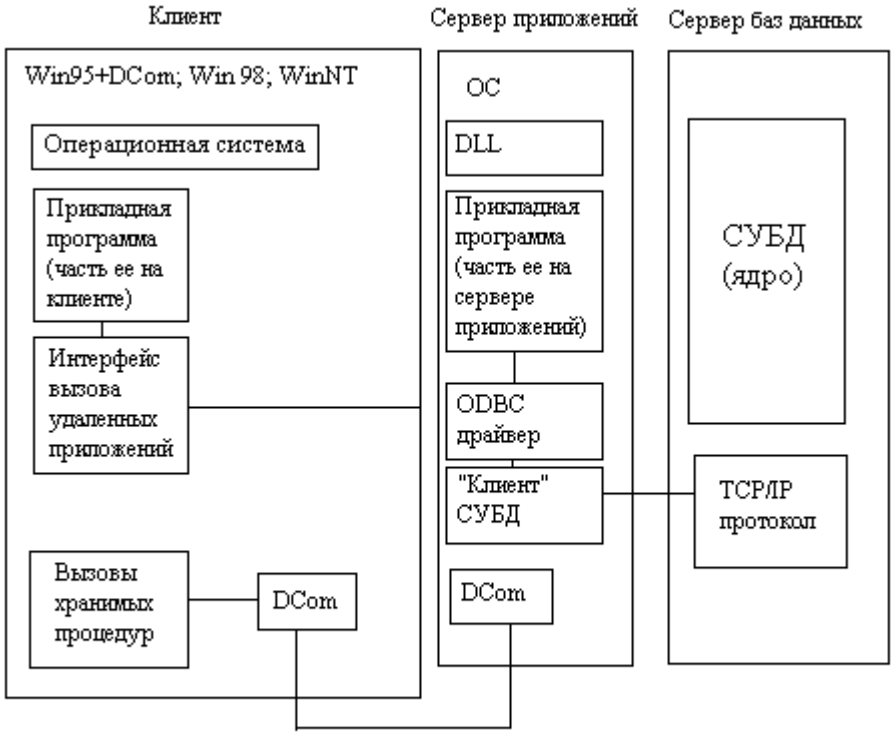


Рис. 3.1. Трехуровневая архитектура клиент-сервер

В трехзвенной архитектуре всю логику работы с сервером БД можно возложить на специальный сервер приложений, а разделенные на отдельные фрагменты приложения уменьшают нагрузку и на машину-клиента, и на сервер БД, перенося соответствующие операции на специализированный сервер приложений. Серверная часть приложений лучше защищена, а сами приложения могут либо непосредственно адресоваться к другим серверным приложениям, либо маршрутизировать запросы к ним . Достоинства этой архитектуры (рис. 3.2) можно разбить на 2 категории – клиентскую часть в виде броузера и серверную часть.

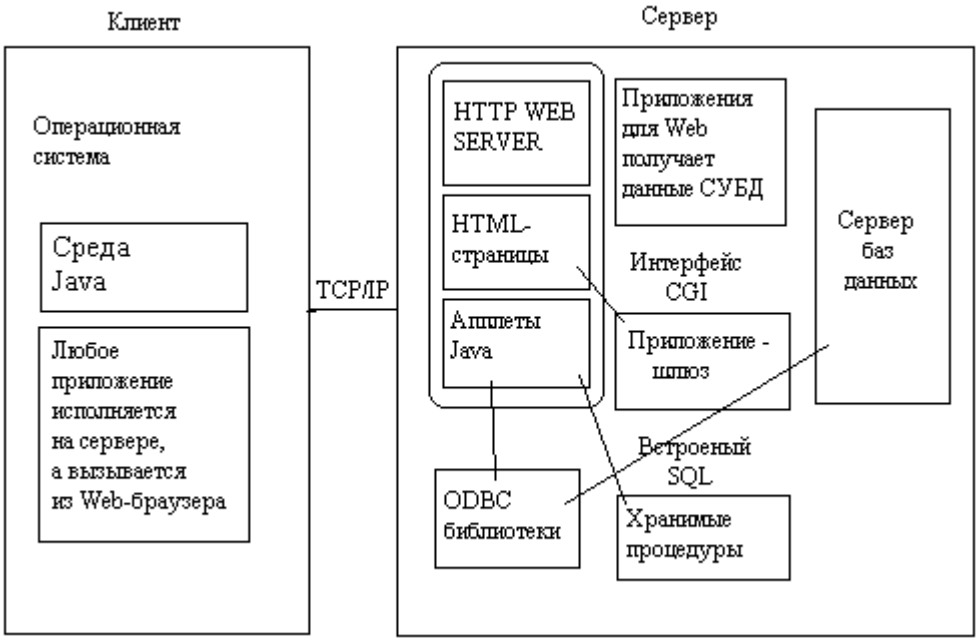


Рис. 3.2 Клиент-сервер с броузером

3.1 Двухуровневая архитектура "клиент-сервер"

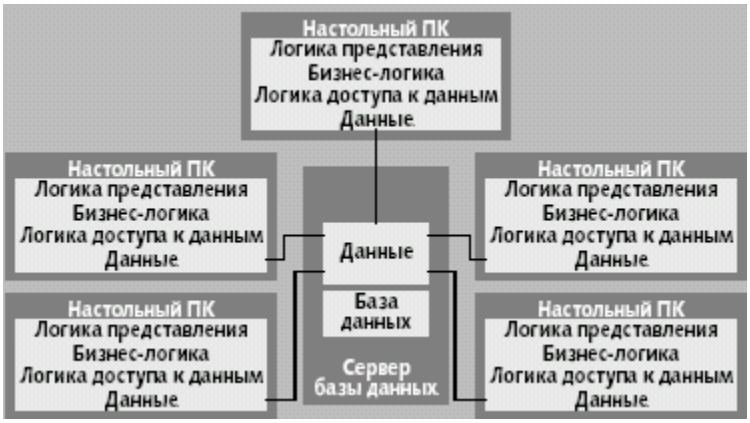


Рис. 3.3. Ранние архитектуры клиент-сервер

Стремление исправить архитектуру разделения файлов привело к замене файлового сервера сервером баз данных (рис. 3.3). Вместо передачи файлов целиком он пересылает только ответы на запросы клиентов, уменьшая нагрузку на сеть. Эта стратегия вызвала появление архитектуры клиент-сервер. Появившись в 80-х годах, она ввела понятие «клиента» (сторона, запрашивающая функции/обслуживание) и «сервера» (сторона, предоставляющая функции/обслуживание). На уровне программного обеспечения разделение на клиента и сервер является логическим: процессы клиента и сервера могут физически размещаться как на одной, так и на разных машинах. Под общим концептуальным названием скрываются три варианта архитектуры: двухзвенная, трехзвенная и многозвенная.

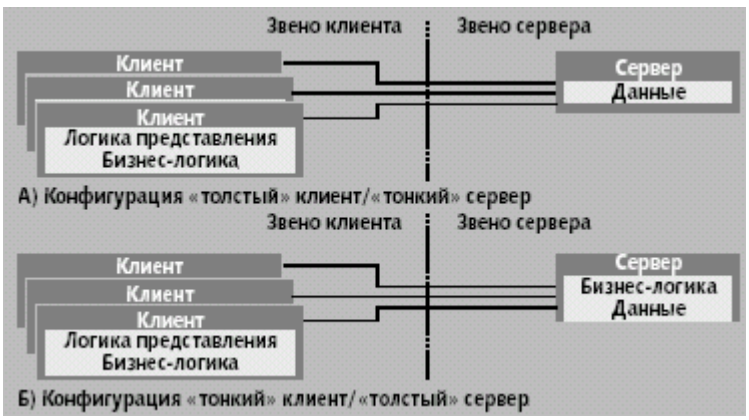


Рис. 3.4. Двухзвенная архитектура

Самая старая — двухзвенная (рис. 3.4). Она разделяет приложение на две части, клиентскую и серверную. Сторона клиента содержит логику представления, а логика доступа к данным, СУБД и сама база находятся на стороне сервера.

Остаются алгоритмы бизнес-логики, которые могут быть размещены как на машине клиента вместе с логикой представления (конфигурация «толстый клиент»), так и на стороне сервера (конфигурация «тонкий клиент»), или даже могут быть разделены

между ними. Конфигурация «толстый клиент» более распространена: суммарная вычислительная мощность клиентов, по крайней мере, в теории, предполагается большей, чем мощность единственного сервера. Подобный ход рассуждений привел некоторых из разработчиков к созданию приложений, где даже логика доступа к данным размещается на клиенте, оставляя серверу только поддержание самой базы данных.

Двухзвенная архитектура, особенно конфигурация «толстый клиент», имеет ряд недостатков. Например, как и в архитектуре разделения файлов, — это ограничение, вытекающее из вычислительной мощности отдельных машин клиентов. Но еще хуже имеющее фундаментальный характер ограничения на число одновременных соединений с сервером. Сервер поддерживает открытое соединение со всеми активными клиентами, даже если никакой работы нет. Это необходимо, чтобы сервер мог получать сигналы тактового импульса, что не так страшно, когда клиентов менее 100 [9-12]; однако сверх этого числа производительность начинает быстро деградировать до недопустимо низкого уровня. Хорошим примером возникновения подобной проблемы может служить работа прокси- сервера.

В некоторых прикладных системах бизнес-логику пытаются реализовать, используя хранимые процедуры. Идея состоит в том, чтобы в соответствии с «тонкой» конфигурацией клиента переместить алгоритмы бизнес-логики на серверную машину, ближе к данным, которые требуются им постоянно. Это выглядит как хорошая идея, однако только усугубляет главную проблему. Так как осуществляющие бизнес-правила процессы теперь управляются СУБД, число пользователей, которых может поддерживать такая система, ограничено максимумом возможных активных соединений с СУБД. Кроме того, от СУБД к СУБД механизмы хранимых процедур разнятся. Тем не менее, двухзвенная архитектура хорошо работает в маленьких рабочих группах [9-12]. С начала 90-х годов появилась масса инструментальных средств, упрощающих создание систем в такой архитектуре, в том числе Delphi и PowerBuilder.

3.1.1. Место и функции сервера БД

Информационная система в двухуровневой архитектуре, использующая сервер с базой данных, включает два вида программного обеспечения:

- Программное обеспечение общего назначения для поддержания БД, называемое системой управления базами данных - СУБД;
- Прикладное ПО, которое использует средства СУБД для выполнения конкретных задач.

Прикладные программы используют средства СУБД для обращения к данным и их обработки, решая те или иные задачи, создавая различные документы и отчеты. СУБД предназначена для обеспечения управления БД. Сервер БД выполняет следующие функции:

- Одновременный доступ к данным для многих приложений;
- Защиту данных и обеспечение их целостности механизмами СУБД;
- Ориентированные на пользователей типовые запросы к соответствующим БД, в том числе и распределенной информации по разным БД;
- Ориентированные на программистов сервисы для создания приложений.

С развитием и расширением функций СУБД двухуровневая архитектура клиент-сервер прошла несколько стадий, которые характеризуются различным распределением выполняемых функций приложений и, наконец, установилось следующее разделение:

- Сервер БД
  * Хранение данных и управление ими;
  * Контроль доступа к данным согласно назначаемым правам пользователей и приложений;
  * Выполнение части логики приложений, которая определена в виде связей между реляционными таблицами и реализуется в виде триггеров и хранимых процедур;
  * Выполнение репликаций данных для сегментов распределенных БД или систем БД, расположенных в различных узлах корпоративной сети.
- Рабочая станция
  * Ввод и модификация данных хранимых в БД;
  * Выполнение основной обработки данных приложений;
  * Хранение и реализация графических компонент приложений;
  * Доступ к данным в БД через "клиента" СУБД.

Такая архитектура оставалась эффективной при эксплуатации БД, которые размещались на больших компьютерах, типа меинфреймов или IBM-360, но с появлением серверных систем, оказалось допустимым при количестве рабочих станций до 30.

3.1.2. Понятие клиента СУБД. Функции СУБД.

Каждая многопользовательская СУБД предоставляет доступ к базам данных через специальную интерфейсную программу, которая называется "клиентом" СУБД. Эта программа является, обычно, лицензированным продуктом для СУБД типа ORACLE, OS/2, DB2, Informix, Sybase и др. Только MS SQL-server имеет встроенные средства. Поэтому только через клиента, явного и скрытого от глаз пользователя, может быть осуществлен доступ к СУБД и к БД.

СУБД обычно выполняет следующие функции:

- Централизованное определение и контроль данных определяемое как словарь/каталог данных;
- Обеспечение целостности данных и их защита;
- Выполнение операций по управлению транзакциями (журнализация транзакций, "откат" транзакций и т.п.);
- Управление доступом к данным одновременно для множества пользователей;
- Выполнение архивирования служебных данных и ведение системных журналов;
- Создание и поддержание виртуальных моделей данных для приложений.

3.1.3. Основные принципы и критерии при разработке программных компонентов систем клиент-сервер

При разработке приложений в технологиях клиент-сервер желательно придерживаться следующих критериев [9]:

1. *Сохранение автономности сервера.* Клиенты не должны ограничивать доступность серверов (например, путем блокирования больших объемов данных) и не должны нарушать целостность каких-либо данных сервера.
2. *Сохранение автономности клиента.* Функционирование клиента не должно зависеть от местоположения данных.
3. *Поддержка независимости приложений от сервера.* Клиент приложения должен быть независим от аппаратной платформы, вида операционной системы и используемых сервисов серверов БД.
4. *Доступность специфических средств сервера.* Клиенту должны быть доступны специфические функции конкретного сервера для более качественного выполнения работы.
5. *Минимум дополнительных требований к рабочей станции.* Программное обеспечение клиента не должно быть ресурсоемким и не иметь особенностей для доступа к конкретному серверу БД.
6. *Полнота вариантов соединения.* Должна быть обеспечена возможность прямых соединений с сервером.
7. *Возможность прототипирования приложений.* Удаленность данных не должна препятствовать возможности прототипирования прикладного ПО.
8. *Полнота пользовательского инструментария.* В состав среды приложения должны входить инструментальные средства для создания экранных форм, генерации запросов и создания источников данных.
9. *Полнота среды разработки приложений.* Среда разработки должна включать средства для установления межсетевых соединений и управления ими, доступа к сервисам глобального назначения.
10. *Открытая среда включающих языков.* Приложение не должно быть критическим к расширению его компонентов с использованием других языков.
11. *Следование стандартам.* Создание приложений должно максимально опираться на международные стандарты и рекомендации (стандартные интерфейсы клиент-сервер образуют программное обеспечение промежуточного слоя).

3.1.4. Основные недостатки двухуровневой архитектуры

К основным недостаткам двухуровневой архитектуры можно отнести:

- Невозможность создания распределенных приложений;

- Критичность сервера БД к числу пользовательских одновременно работающих приложений (оптимальным является около 30);
- Критичность сервера к модификации больших объемов данных в одной транзакции (т.е. к длине транзакции);
- Сложности при создании приложений работающих с разнотипными СУБД (необходимость использования разных клиентов СУБД);
- Невозможность использования составных транзакций (т.е. сложных транзакций, которые подлежат раздельному исполнению в разных БД);
- Трудности в сопровождении и модификации клиентов приложения (необходимость синхронного обновления dll-библиотек, ОС, SQL-запросов, новых компонентов приложения и интерфейсов);
- Повышенные требования к ресурсам рабочей станции (все вычисления делаются на клиентской РС, графические средства поддерживаются там же и т.д.).

Несмотря на приведенные недостатки, для малого количества РС можно использовать двухуровневую архитектуру при небольшом числе приложений и стабильности деятельности организации.

3.2.1. Понятия "тонкого" и "сверхтонкого" клиентов

По мере развития архитектуры "клиент-сервер" и появлением трехуровневой и других многозвенных архитектур возникло понятие "тонкого" клиента, которое обозначает отсутствие на рабочей станции пользователя некоторых компонент, которые раньше требовались для взаимодействия с сервером БД для нормальной работы приложений. Причем, эти компоненты переместились с клиента на сервер приложений и стали использоваться многими клиентами одновременно.

К таким компонентам можно отнести:

- "клиент" СУБД;
- ODBC- драйверы для соединения с БД;
- части приложений, реализующую SQL-запросы к базам данных;
- части приложений, которые используют для расчетов и вычислений одни и те же алгоритмы, т.е. создавался общий вычислительный ресурс.

Все эти технологические изменения привели к требованию дальнейшего увеличения мощности сервера приложений и увеличению сетевого трафика между рабочими станциями и серверами.

На рабочей станции оставались средства графической поддержки приложений и стандартные приложения вида MS Office. Поэтому к ресурсам рабочих станций стали предъявляться меньшие требования, хотя появление ОС более поздних модификаций и развитие мультимедийных приложений быстро компенсировали этот "недостаток".

Появление новых Web-технологий и их развитие привело к внедрению Internet-технологий в корпоративную компьютерную среду "броузеров", которые стали основой клиентской части приложений Web- архитектур нового уровня, получивших название "сверхтонкого" клиента.

В такой архитектуре все данные и все приложение целиком располагалось на сервере приложений и сервере БД. Броузер "подкачивал" данные из Web- сервера и "визуализировал" экраны приложения. Так появилась среда ***слабоструктурированных*** приложений, в которой картинки приложений строятся динамически, в зависимости от данных вводимых пользователем. Ясно, что использование броузера предъявляет еще меньше требований к ресурсам рабочих станций.

Кроме того, при такой технологии отпадает необходимость модифицировать клиентские части приложения, что упрощает их сопровождение и отпадает необходимость в периодическом обучении пользователей. Можно говорить об интеграции пользовательских интерфейсов в результате этого.

3.2. Трехуровневая архитектура "клиент-сервер"

По мере развития технологии интегрирования распределенных данных в клиент-серверных приложениях и появлением все большего числа пользователей, которые контактировали непосредственно с СУБД, проявилось, что при числе пользователей в корпоративной сети более 30 необходимо удваивать ресурсы серверов БД потому, что резко возрастает время реакции системы на запросы приложений. Появление и развитие распределенных БД поставило вопрос о такой организации данных, при которой приложение могло бы с помощью специальных средств выполнять распределенные транзакции, направляя запросы к разным БД, расположенным в разных узлах сети и, часто, имеющих различные программно-аппаратные платформы.

Кроме того, начали интенсивно развиваться графические приложения, которые требовали перекачки больших объемов информации между рабочей станцией и сервером БД. Появились аудио- и видеоданные, требовавшие передачи и обработки гигабайтов информации. Развитие дорогих кластерных систем резко ограничивалось их высокой стоимостью.

Увеличение числа приложений, в которых использовались многочисленные одинаковые фрагменты программ, породило развитие объектного программирования, что в свою очередь поставило задачу размещения указаных фрагментов в одном месте с целью удобства их модернизации и сопровождения.

Появление и развитие сетевого вычислительного ресурса, когда отдельные фрагменты вычислительных программ размещаются на высокопроизводительном оборудовании, также показало необходимость создания серверов, которые бы специализировались на этом классе задач.

Все это породило необходимость создания специального класса серверов, которые получили название серверов приложений и которые давали возможность разгрузить многочисленные приложения от типовых задач обработки информации и делали бы это лучше, чем рабочие станции. Так была создана трехзвенная архитектура клиент-сервер, которые являются средним, промежуточным звеном между клиентской частью приложений и серверами БД. Через компоненты DCOM,сокеты и т.п. они взаимодействуют между собой, упрощая создание новых и сопровождение старых приложений в корпоративной сети предприятий.

3.2.2 Функции сервера приложений и его назначение

Приложения доступны любому пользователю сети Internet/Intranet , имеющему право обращаться к ним. Поскольку все операции по созданию и усовершенствованию системы производятся на сервере, возникает необходимость сопровождать и модернизировать части приложений, находящиеся на машинах-клиентах. Такая конфигурация способна обеспечить работу десятка тысяч или даже миллиона пользователей, являясь идеальной архитектурой для унаследованных программ.

Клиентские приложения обращаются не непосредственно к серверной СУБД путем вызова функций клиентских API, а к серверу приложений, являющемуся для них источником данных. При этом собственно клиентская часть серверной СУБД и библиотеки типа BDE на рабочей станции, где используется такое клиентское приложение, присутствовать не обязаны. Вместо них используется одна-единственная динамически загружаемая библиотека dbclient.dll размером около150 Кбайт. Таким образом, созданная информационная система становится трехзвенной, а сервер приложений является средним звеном в цепи “тонкий” клиент – сервер приложений – сервер БД.

Данная технология может быть реализована с помощью набора компонентов и классов Delphi 3, обеспечивающих создание серверов приложений и клиентских прикладных программ. Сначала все-таки рассмотрим создание классического клиент-серверного приложения, а уже потом перейдем к схеме Intranet с применением Web.

Подготовка данных для сервера приложений

Для создания сервера приложений требуется СУБД, клиентом которой будет сервер приложений. В принципе можно использовать любую серверную СУБД (и вообще говоря, даже набор таблиц Dbase или Paradox). В примерах, приводимых ниже, используется СУБД Oracle для платформы Windows NT.

Создадим нового пользователя Oracle и перенесем с помощью утилиты Data Migration Wizard таблицы CLIENTS.DBF и HOLDINGS.DBF из БД DBDEMOS, входящей в комплект поставки Delphi 3, в СУБД Oracle. После этого можно приступить к созданию простейшего сервера приложений.

Создание сервера приложений

Создадим простейшую форму со значением свойства FormStyle, равным fsStayOnTop. Главное назначение этой формы – быть индикатором запущенного

сервера приложений. Далее создадим обычный модуль данных, содержащий компонент TDataBase, указывающий на серверную СУБД. Для простоты установим свойство LoginPrompt этого компонента равным False, а в свойстве Params укажем пользователя, от имени которого будет запущен сервер приложений, и его пароль: Username = user1 Password = user1

Далее из репозитория объектов (доступ к которому осуществляется выбором пункта меню File/New…) со страницы New следует выбрать *** Remote Data Module.

Главное отличие Remote Data Module от обычного DataModule заключается в том, что объекты, помещенные в него, могут иметь COM-интерфейс.

Создание Remote Data Module начинается с запуска Remote Data Wizard, в котором определяется имя класса, с которым сервер приложения (OLE-сервер по терминологии Microsoft) будет зарегистрирован в реестре.

Далее в созданный удаленный модуль данных поместим 2 компонента TTable, связанных с компонентом Database 1 и указывающих на только что созданные таблицы CLIENTS и HOLDINGS, а также 2 компонента Tprovider установим равными именами соответствующих компонентов установим равными именами соответствующих компонентов TTable. Установим связь master/detail между таблицами по полю ACCT_NBR. Откроем таблицы.

Далее из контекстного меню компонентов TTable и TProvider нужно выбрать опции

- Export MyProv1 from data module,
- Export MyProv2 from data module,
- Export Table 1 from data module,
- Export Table2 from data module.

Последние 2 опции являются необязательными, так как компоненты TProvider обеспечивают другим приложениям, функционирующим в ***RDM доступ к данным, содержащимся в компонентах Ttable и TQuery.

В библиотеке типов (Type library) для COM-объекта, который будет создаваться при запуске данного приложения, автоматически прописывается при этом интерфейс, осуществляющий доступ извне к компонентам MyProv1, MyProv2, Table 1и Table2.

Далее нужно сохранить проект и скомпилировать приложение. Следует убедиться, что оно зарегистрировано в реестре под выбранным ранее именем. Если по каким-либо причинам этого не произошло, надо запустить приложение с параметром . После этого проект следует закрыть.

3.2.3. Основные компоненты сервера приложений.

Серверная часть

Приложения доступны любому пользователю сети Internet/Intranet , имеющему право обращаться к ним. Поскольку все операции по сопровождению и усовершенствованию системы производятся на сервере, отпадает необходимость сопровождать и модернизировать части приложений, находящиеся на машинах-клиентах. Такая конфигурация способна обеспечить работу десятков тысяч или даже миллиона пользователей, являясь идеальной архитектурой для унаследованных программ.

Клиентские приложения обращаются не непосредственно к серверной СУБД путем вызова функций клиентских API, а к серверу приложений, являющемуся для них источником данных. При этом собственно клиентская часть серверной СУБД и библиотеки типа BDE на рабочей станции, где используется такое клиентское приложение, присутствовать не обязаны. Вместо них используется одна-единственная динамически загружаемая библиотека dbclient.dll размером около150 Кбайт. Таким образом, созданная информационная система становится трехзвенной, а сервер приложений является средним звеном в цепи “тонкий” клиент – сервер приложений – сервер БД.

Данная технология может быть реализована с помощью набора компонентов и классов Delphi 5(6), обеспечивающих создание серверов приложений и клиентских прикладных программ. Сначала все-таки рассмотрим создание классического клиент-серверного приложения, а уже потом перейдем к схеме Intranet с применением Web.

Подготовка данных для сервера приложений

Для создания сервера приложений требуется СУБД, клиентом которой будет сервер приложений. В принципе можно использовать любую серверную СУБД (и вообще говоря, даже набор таблиц Dbase или Paradox). В примерах, приводимых ниже, используется СУБД Oracle для платформы Windows NT.

Создадим нового пользователя Oracle и перенесем с помощью утилиты Data Migration Wizard таблицы CLIENTS.DBF и HOLDINGS.DBF из БД DBDEMOS, входящей в комплект поставки Delphi 3, в СУБД Oracle. После этого можно приступить к созданию простейшего сервера приложений.

Создание сервера приложений

Создадим простейшую форму со значением свойства FormStyle, равным fsStayOnTop. Главное назначение этой формы – быть индикатором запущенного сервера приложений. Далее создадим обычный модуль данных, содержащий компонент TDataBase, указывающий на серверную СУБД. Для простоты установим свойство LoginPrompt этого компонента равным False, а в свойстве Params укажем пользователя, от имени которого будет запущен сервер приложений, и его пароль: Username = user1 Password = user1

Далее из репозитория объектов (доступ к которому осуществляется выбором пункта меню File/New…) со страницы New следует выбрать *** Remote Data Module.

Главное отличие Remote Data Module от обычного DataModule заключается в том, что объекты, помещенные в него, могут иметь COM-интерфейс.

Создание Remote Data Module начинается с запуска Remote Data Wizard, в котором определяется имя класса, с которым сервер приложения (OLE-сервер по терминологии Microsoft) будет зарегистрирован в реестре.

Далее в созданный удаленный модуль данных поместим 2 компонента TTable, связанных с компонентом Database 1 и указывающих на только что созданные таблицы CLIENTS и HOLDINGS, а также 2 компонента Tprovider установим равными именами соответствующих компонентов установим равными именами соответствующих компонентов TTable. Установим связь master/detail между таблицами по полю ACCT_NBR. Откроем таблицы.

Далее из контекстного меню компонентов TTable и TProvider нужно выбрать опции

- Export MyProv1 from data module,
- Export MyProv2 from data module,
- Export Table 1 from data module,
- Export Table2 from data module.

Последние 2 опции являются необязательными, так как компоненты TProvider обеспечивают другим приложениям доступ к данным, содержащимся в компонентах Ttable и TQuery.

В библиотеке типов (Type library) для COM-объекта, который будет создаваться при запуске данного приложения, автоматически прописывается при этом интерфейс, осуществляющий доступ извне к компонентам MyProv1, MyProv2, Table 1и Table2.

Далее нужно сохранить проект и скомпилировать приложение. Следует убедиться, что оно зарегистрировано в реестре под выбранным ранее именем. Если по каким-либо причинам этого не произошло, надо запустить приложение с параметром . После этого проект следует закрыть.

3.2.4 Основные компоненты "клиента" в 3-х уровневой архитектуре

Прикладная программа доступна с любого компьютера, на котором установлена программа – сетевой навигатор (броузер). Пользователю нет необходимости изучать интерфейс прикладной программы, потому что он всегда преобразуется к стандарту HTML-странички. Это помогает снизить затраты на обучение. Кроме того, пользователя совершенно не заботят особенности аппаратной платформы и операционной системы, поскольку он имеет дело только с браузером, который умеет делать все.

Создание клиентской части приложения

(на том же ПК, на котором находится и сервер приложений)

После создания сервера приложений, можно приступить к созданию клиентского приложения. Создадим главную форму приложения – обычный модуль данных, который должен быть видимым из главной программы. В модуль данных поместим 1 компонент TRemoteServer, 2 связанных с ним компонента TclientDataSet и 2 связанных с ними компонента TDataSource.

Установим свойство ServerName компонента RemoteServer1 (выбранных из списка или введя вручную). После этого, если в реестре есть соответствующая запись, автоматически будет установлено значение свойства ServerGUID. При этом сервер приложений, созданный ранее, должен автоматически запуститься, о чем будет свидетельствовать появление на экране его главной формы. После этого можно установить нужные значения свойств ProviderName компонентов Tclient-DataSet, выбрав их из ниспадающего списка (поскольку сервер приложения выбран, все его свойства, доступные через интерфейс, теперь становятся видны в Delphi). После этого можно установить связь master/detail между таблицами по общему полю ACCT_NBR.

Далее нужно установить значения свойства Active-компонентов TclientDataSet равными True и связать интерфейсные элементы главной формы приложения с соответствующими полями таблиц, доступных через запущенный сервер приложений.

Теперь можно скомпилировать и запустить приложение и с помощью утилиты SQL Monitor проследить, какие именно запросы созданный нами сервер приложений направляет серверу БД.

Использование DCOM для создания "тонкого" клиента
(разнесение приложения)

Для того, чтобы разнести клиентскую часть приложения и серверную его часть на разные компьютеры локальной сети, можно использовать либо технологию OLEnterprise (составная часть Delphi Client/Server), либо непосредственно технологию DCOM (Distributed Component Object Model) корпорации Microsoft. Использование этой технологии для запуска серверов приложений возможно лишь при условии наличия первичного контроллера домена в сети и выбранной на рабочих станциях установки User Level Access Control на странице Access Control раздела Network панели управления Windows (это необходимо для переноса с контроллера домена списка пользователей для разрешения им доступа к серверу приложений). Помимо этого, сервер приложений может быть доступен для клиентских приложений, но только если он выполняется под управлением Windows NT, а не Windows 95.

Сведения о сервере приложений должны быть занесены в реестр клиента (этого можно добиться, например, путем запуска сервера приложений непосредственно на рабочей станции с ключом /regserver). После этого следует внести изменения в реестр так, чтобы зарегистрированный таким образом OLE-объект воспринимался не как локальный, а как удаленный объект. Это можно сделать, отредактировав реестр вручную или с помощью утилиты конфигурации DCOM, указав явно, на каком компьютере запускается сервер приложений и кто из пользователей имеет право его запускать. Утилита запускается дважды: на сервере и на клиенте.

При желании можно также попытаться создать более сложную реализацию сервера приложений, существующего в виде многопоточного приложения. Если при создании сервера выбрана опция Multiple Instanse, каждый клиент на сервере может запустить свой процесс. Если же при создании сервера была выбрана опция Single Instanse, то каждый клиент в рамках единого процесса может создавать свой поток. В этом случае при создании сервера рекомендуется использовать компонент TSession со значением AutoSessionName, равным True, так, чтобы потоки, созданные разными клиентами, не использовали одну и ту же сессию.

В палитре компонентов Delphi, начиная с версии 3.01, имеется компонент MidasConnection, который может быть использован вместо TRemoteServer. Он полезен главным образом тем, что обладает свойством ConnectType, позволяющим определить, каким путем осуществляется соединение с сервером – посредством DCOM, сокетов TCP/IP или OLEnterprise.

Intranet-подход к разработке СУБД- ориентированных приложений на базе технологии клиент-сервер (Delphi C/S)

Создание сложных программных продуктов в технологии Intranet представляет собой далеко не тривиальную задачу. В процессе работы могут обнаружиться различные проблемы и подводные камни, причем в таких местах, где при обычном программировании не бывает никаких сложностей. С другой стороны, многие аспекты разработки интерактивных программ оказываются чрезвычайно легки по сравнению с традиционным программированием именно в технологии Intranet. Главный вывод, который можно сделать при этом, заключается в том, что разработка сложных систем в технологии Intranet требует совершенно иного стиля мышления, чем проведение аналогичной работы в технологии клиент-сервер. Одно дело – простые Intranet-программы (типа гостевой книги), и совсем другое – реализация сложных задач – тут уже требуются специальные подходы и тщательно отработанные последовательности действий. Рассмотрим, например, некоторые из наиболее типичных и поучительных моментов в процессе создания развернутого Intranet-приложения.

Общая характеристика программного комплекса Intranet-InfoVisor

Программный комплекс Intranet-InfoVisor является гибким инструментом аналитической обработки реляционных хранилищ данных и относится к классу продуктов OLAP (OnLine Analytical Processing – оперативная аналитическая обработка данных). Комплекс InfoVisor ориентирован на использование специалистами служб информационных технологий и системного анализа для подготовки отчетов, прогнозов и рекомендаций, способствующих поддержке принятия управленческих решений.

Целью применения программного комплекса является извлечение полезной информации из накопленных фактов и поддержка принятия решений.

Комплекс Intranet-InfoVisor целесообразно применять в рамках информатизации сложных социально-экономических систем, от организации или предприятия до ведомства или региона, в целях обеспечения объективного комплексного взгляда на состояние и функционирование объекта управления в статике и динамике. Кроме того, он может быть использован для обработки результатов социологических, демографических, экологических и тому подобных исследований, т.е. везде, где требуется аналитическая надстройка над большими массивами собранных сведений.

В состав программного комплекса Intranet-InfoVisor входят 3 системы.

1. InfoVisor-аналитик (основное клиентское приложение) – инструмент пользователя-аналитика, предназначенный для изучения содержимого реляционных БД через формируемые в них информационные модели,

многомерного анализа агрегированной информации, генерации табличных, графических и географических отчетов, а также для подготовки исходных данных для динамически подключаемых внешних модулей интеллектуального информационного анализа.

2. InfoVisor-администратор – основное приложение администратора системы, предназначенное для создания, редактирования и поддержания логической целостности аналитических метаданных в реляционных БД произвольной структуры, т.е. для описания информационных моделей, обеспечивающих многомерный агрегированный взгляд на содержащуюся в них информацию.
3. InfoVisor-загрузка – дополнительное средство администратора, предназначенное для наполнения и регулярного обновления хранилища данных из информационных источников согласно заданным сценариям.

Приложения InfoVisor-аналитик и InfoVisor-администратор выполнены в виде Web-модулей, работающих в среде Internet/Intranet. Приложение InfoVisor-загрузка является клиент-серверной программой, состоящей из интерактивной подсистемы задания сценариев обновления хранилищ данных и резидентной подсистемы выполнения разработанных сценариев.

Приложения InfoVisor-администратор и InfoVisor-загрузка являются инструментами администрирования. С их помощью администратор системы может поддерживать актуальность и непротиворечивость хранилища данных, а также полноту и целостность информационных моделей, составляющих аналитические метаданные хранилища и обеспечивающих эффективную работу конечных пользователей с основным клиентским приложением InfoVisor-аналитик.

3.2.4 Основные компоненты "клиента" в 3-х уровневой архитектуре

Прикладная программа доступна с любого компьютера, на котором установлена программа – сетевой навигатор (броузер). Пользователю нет необходимости изучать интерфейс прикладной программы, потому что он всегда преобразуется к стандарту HTML-странички. Это помогает снизить затраты на обучение. Кроме того, пользователя совершенно не заботят особенности аппаратной платформы и операционной системы, поскольку он имеет дело только с браузером, который умеет делать все.

Создание клиентской части приложения

(на том же ПК, на котором находится и сервер приложений)

После создания сервера приложений, можно приступить к созданию клиентского приложения. Создадим главную форму приложения – обычный модуль данных, который должен быть видимым из главной программы. В модуль данных поместим 1 компонент TRemoteServer, 2 связанных с ним компонента TclientDataSet и 2 связанных с ними компонента TDataSource.

Установим свойство ServerName компонента RemoteServer1 (выбранных из списка или введя вручную). После этого, если в реестре есть соответствующая запись, автоматически будет установлено значение свойства ServerGUID. При этом сервер приложений, созданный ранее, должен автоматически запуститься, о чем будет свидетельствовать появление на экране его главной формы. После этого можно установить нужные значения свойств ProviderName компонентов Tclient-DataSet, выбрав их из ниспадающего списка (поскольку сервер приложения выбран, все его свойства, доступные через интерфейс, теперь становятся видны в Delphi). После этого можно установить связь master/detail между таблицами по общему полю ACCT_NBR.

Далее нужно установить значения свойства Active-компонентов TclientDataSet равными True и связать интерфейсные элементы главной формы приложения с соответствующими полями таблиц, доступных через запущенный сервер приложений.

Теперь можно скомпилировать и запустить приложение и с помощью утилиты SQL Monitor проследить, какие именно запросы созданный нами сервер приложений направляет серверу БД.

Использование DCOM для создания "тонкого" клиента
(разнесение приложения)

Для того, чтобы разнести клиентскую часть приложения и серверную его часть на разные компьютеры локальной сети, можно использовать либо технологию OLEnterprise (составная часть Delphi Client/Server), либо непосредственно технологию DCOM (Distributed Component Object Model) корпорации Microsoft. Использование этой технологии для запуска серверов приложений возможно лишь при условии наличия первичного контроллера домена в сети и выбранной на рабочих станциях установки User Level Access Control на странице Access Control раздела Network панели управления Windows (это необходимо для переноса с контроллера домена списка пользователей для разрешения им доступа к серверу приложений). Помимо этого, сервер приложений может быть доступен для клиентских приложений, но только если он выполняется под управлением Windows NT, а не Windows 95.

Сведения о сервере приложений должны быть занесены в реестр клиента (этого можно добиться, например, путем запуска сервера приложений непосредственно на рабочей станции с ключом /regserver). После этого следует внести изменения в реестр так, чтобы зарегистрированный таким образом OLE-объект воспринимался не как локальный, а как удаленный объект. Это можно сделать, отредактировав реестр вручную или с помощью утилиты конфигурации DCOM, указав явно, на каком компьютере запускается сервер приложений и кто из пользователей имеет право его запускать. Утилита запускается дважды: на сервере и на клиенте.

При желании можно также попытаться создать более сложную реализацию сервера приложений, существующего в виде многопоточного приложения. Если при создании сервера выбрана опция Multiple Instanse, каждый клиент на сервере может запустить свой процесс. Если же при создании сервера была выбрана опция Single Instanse, то каждый клиент в рамках единого процесса может создавать свой поток. В этом случае при создании сервера рекомендуется использовать компонент TSession со значением AutoSessionName, равным True, так, чтобы потоки, созданные разными клиентами, не использовали одну и ту же сессию.

В палитре компонентов Delphi, начиная с версии 3.01, имеется компонент MidasConnection, который может быть использован вместо TRemoteServer. Он полезен главным образом тем, что обладает свойством ConnectType, позволяющим определить, каким путем осуществляется соединение с сервером – посредством DCOM, сокетов TCP/IP или OLEnterprise.

Intranet-подход к разработке СУБД- ориентированных приложений на базе технологии клиент-сервер (Delphi C/S)

Создание сложных программных продуктов в технологии Intranet представляет собой далеко не тривиальную задачу. В процессе работы могут обнаружиться различные проблемы и подводные камни, причем в таких местах, где при обычном программировании не бывает никаких сложностей. С другой стороны, многие аспекты разработки интерактивных программ оказываются чрезвычайно легки по сравнению с традиционным программированием именно в технологии Intranet. Главный вывод, который можно сделать при этом, заключается в том, что разработка сложных систем в технологии Intranet требует совершенно иного стиля мышления, чем проведение аналогичной работы в технологии клиент-сервер. Одно дело – простые Intranet-программы (типа гостевой книги), и совсем другое – реализация сложных задач – тут уже требуются специальные подходы и тщательно отработанные последовательности действий. Рассмотрим, например, некоторые из наиболее типичных и поучительных моментов в процессе создания развернутого Intranet-приложения.

Общая характеристика программного комплекса Intranet-InfoVisor

Программный комплекс Intranet-InfoVisor является гибким инструментом аналитической обработки реляционных хранилищ данных и относится к классу продуктов OLAP (OnLine Analytical Processing – оперативная аналитическая обработка данных). Комплекс InfoVisor ориентирован на использование специалистами служб информационных технологий и системного анализа для подготовки отчетов, прогнозов и рекомендаций, способствующих поддержке принятия управленческих решений.

Целью применения программного комплекса является извлечение полезной информации из накопленных фактов и поддержка принятия решений.

Комплекс Intranet-InfoVisor целесообразно применять в рамках информатизации сложных социально-экономических систем, от организации или предприятия до ведомства или региона, в целях обеспечения объективного комплексного взгляда на состояние и функционирование объекта управления в статике и динамике. Кроме того, он может быть использован для обработки результатов социологических, демографических, экологических и тому подобных исследований, т.е. везде, где требуется аналитическая надстройка над большими массивами собранных сведений.

В состав программного комплекса Intranet-InfoVisor входят 3 системы.

1. InfoVisor-аналитик (основное клиентское приложение) – инструмент пользователя-аналитика, предназначенный для изучения содержимого реляционных БД через формируемые в них информационные модели,

многомерного анализа агрегированной информации, генерации табличных, графических и географических отчетов, а также для подготовки исходных данных для динамически подключаемых внешних модулей интеллектуального информационного анализа.

2. InfoVisor-администратор – основное приложение администратора системы, предназначенное для создания, редактирования и поддержания логической целостности аналитических метаданных в реляционных БД произвольной структуры, т.е. для описания информационных моделей, обеспечивающих многомерный агрегированный взгляд на содержащуюся в них информацию.
3. InfoVisor-загрузка – дополнительное средство администратора, предназначенное для наполнения и регулярного обновления хранилища данных из информационных источников согласно заданным сценариям.

Приложения InfoVisor-аналитик и InfoVisor-администратор выполнены в виде Web-модулей, работающих в среде Internet/Intranet. Приложение InfoVisor-загрузка является клиент-серверной программой, состоящей из интерактивной подсистемы задания сценариев обновления хранилищ данных и резидентной подсистемы выполнения разработанных сценариев.

Приложения InfoVisor-администратор и InfoVisor-загрузка являются инструментами администрирования. С их помощью администратор системы может поддерживать актуальность и непротиворечивость хранилища данных, а также полноту и целостность информационных моделей, составляющих аналитические метаданные хранилища и обеспечивающих эффективную работу конечных пользователей с основным клиентским приложением InfoVisor-аналитик.

3.2.5 Основные проблемы разработки сложных интерактивных Intranet-приложений

Сложность выполнения разработки, а также ее специфика (оперативная аналитическая обработка реляционных данных для поддержки принятия решений) определили круг проблем, возникших в процессе реализации проекта.

- Сохранение параметров сеанса. Большинство операций при проведении аналитической обработки данных должно выполнятся в виде последовательной цепочки действий, заключающихся в передаче пользователем программе определенных параметров, получении реакции системы, передаче последующей порции параметров и т.д. вплоть до получения желаемого результата анализа. Трудность заключена в том, что, как было выше показано, Web-модуль состоит из набора методов обработки только элементарных действий. То есть сеанс работы программы начинается с приема данных от клиента, затем следует внутренняя обработка поступившего сигнала, и с выдачей клиенту реакции системы в виде гипертекстовой страницы программа завершает свою работу и выгружается из памяти. В общем случае при обработке следующего сигнала программа ничего не знает о своем предыдущем запуске (в отличие от обычной интерактивной программы Web-модуль не имеет в явном виде того, что называется контекстом работы программы). Следовательно, необходимо реализовать полностью скрытый от конечного пользователя механизм преемственности работы, т.е. обеспечить сохранение параметров сеанса и их повторную загрузку при каждом последующем действии пользователя (проблема осложняется тем, что каждое Intranet-приложение является многопользовательским и одновременно с ним может работать несколько клиентов).
- Работа с интерактивной графикой. Комплексный информационный анализ для поддержки приложений возможен только при использовании графического интерфейса, включающего генерацию отчетов, вывод результатов анализа в деловую графику и – что особенно актуально – в геоинформационный интерфейс. Вследствие бедности выразительных средств языка HTML обеспечение работы с графикой, особенно интерактивной, требует особого внимания.
- Поддержка различных кириллических кодировок. Благодаря наличию в настоящее время множества исторически сложившихся стандартов на представление букв кириллицы вообще и русского алфавита в частности,

создание в среде Internet/Intranet русскоязычных приложений, как и простых статических гипертекстовых страниц, наталкивается на проблему одновременной поддержки различных кодировок. В каждом конкретном случае эта задача решается по-своему (или вообще не решается – в этом случае клиентские станции, не поддерживающие текстовый ввод/вывод в предлагаемом стандарте, теряют возможность работы с данным информационным ресурсом). Что касается создавшейся системы оперативной аналитической обработки данных, ее серверная версия Intranet разрабатывалась именно в целях одновременной поддержки разнородных клиентов, т.е. одновременная поддержка нескольких стандартов кодировки была необходима.

- Обеспечение многооконного интерфейса. В отличие от традиционных интерактивных программ, предлагающих многооконный диалоговый интерфейс, классический обмен информацией посредствам гипертекстовых страниц традиционны осуществляется в одном окне браузера. Возможности обеспечения многооконного интерфейса существуют (они основаны на использовании встроенного языка JavaScript), но недостаточно стандартизованы, вследствие чего некоторые действия, выполняемые одним классом браузеров, не поддерживаются другим, и наоборот. Проблема состоит в тщательной отладке всех критически к клиентскому браузеру операций в разных вариантах конфигурации клиента и выборе подмножества методов, действительных для всех распространенных браузеров.

*Недостатки *.dll - библиотек и "registry" ОС Windows*

Использование технологий создания "тонких" приложений с использованием *.dll - библиотек породило другую группу проблем, связанных с синхронизацией этих компонентов на стороне клиентов и на серверах приложений.

Идея Microsoft использования *.dll - компонентов выглядела поначалу привлекательной, т.к. давала возможность обобщить многие функции и методы и, объединив их в библиотеки, упорядочить их использование, экономя в значительной степени труд программистов. Однако, как показала дальнейшая практика их применения, стандартизировать процесс создания этих компонент между разными фирмами не удалось, результатом чего обнаружились следующие коллизии:

- Добавление определенной*.dll - компоненты, которая жестко связана с другими *.dll - компонентами приводит к необходимости обновления всей этой группы как на клиентских рабочих станциях, так и на серверах, с которыми работает

приложение, причем, синхронно, что увеличивает значительно сопровождение приложений;

- При изменении старых *.dll - компонент на новые с расширенными функциями, старые приложения иногда перестают работать, т.к. "точки входов" в *.dll - ках не совпадают. Это приводит к значительным трудностям при развитии информационных систем;

- Необходимо до установки у заказчика, приложение обязательно тестировать на наличие требуемых *.dll - компонент и включать их все в инсталяцинный пакет, иначе их приходится "на ходу" доставлять на рабочие станции, что усложняется при наличии разных версий ОС (Windows 98/Windows NT/Windows 2000).

Использование идеи общей памяти для служебных параметров программных пакетов различных фирм- производителей предложенной в ОС Windows, часто приводит к изменениям установленных в ячейка параметров на новые при инсталляции или "подгрузки" других пакетов и прекращению работы ранее установленных на компьютере средств. Это порождает, в свою очередь, массу вопросов при сопровождении приложений, создаваемых разными фирмами-разработчиками.

3.3 Технологии доступа к SQL-ориентированным БД

До недавнего времени основным методом посылки операторов SQL в сетевых СУБД был интерфейс CLI (Call-Level Interface). Интерфейс CLI, принятый в качестве международного стандарта, обеспечивает библиотеку функций СУБД, которые могут вызываться прикладной программой. Таким образом, вместо того чтобы смешивать SQL с другим языком программирования, интерфейс CLI подобен другим стандартным библиотекам, которые большинство программистов уже привыкли использовать.

Интерфейс CLI подобен динамическому SQL, в котором операторы SQL передаются в СУБД для обработки во время выполнения, но он отличается от встроенного SQL, поскольку в нем не имеется никаких вложенных операторов SQL и никакой предварительный компилятор не требуется.

При использовании интерфейса CLI обычно выполняются следующие шаги:

1. Вызов функции CLI из прикладной программы, чтобы соединиться с системой управления БД (открытие).
2. Формирование оператора SQL и помещение его в буфер. Он впоследствии вызывает одну или более функций CLI, передаваемых в СУБД.
3. Если оператором является оператор SELECT, то приложение вызывает CLI-функции, чтобы возвратить результаты в буфер прикладной программы пользователя. Обычно эта функция возвращает одну строку или один столбец данных.
4. Вызовы из прикладной программы функции CLI на рассоединение с БД (закрытие).

Архитектура доступа к базе данных

Один из вопросов при разработке стандарта ODBC, с которым впоследствии вышла на сцену Microsoft, был – какую часть архитектуры доступа к БД необходимо стандартизировать. Интерфейсы программирования SQL, описанные в предыдущем разделе – встроенный (embedded) SQL, SQL-модули и CLI-функции, - это только одна часть этой архитектуры. В действительности, поскольку ODBC был прежде всего предназначен для соединения прикладных программ на персональных компьютерах с миникомпьютерами и универсальными системами управления БД, имелся также ряд компонентов сети, некоторые из которых могли бы быть стандартизированы.

Сетевой доступ к СУБД

Работа с СУБД в локальной сети требует ряда компонентов, каждый из которых независим от вышележащего слоя и поддерживает определенный интерфейс программирования. Эти компоненты показаны на рис. 2.1.

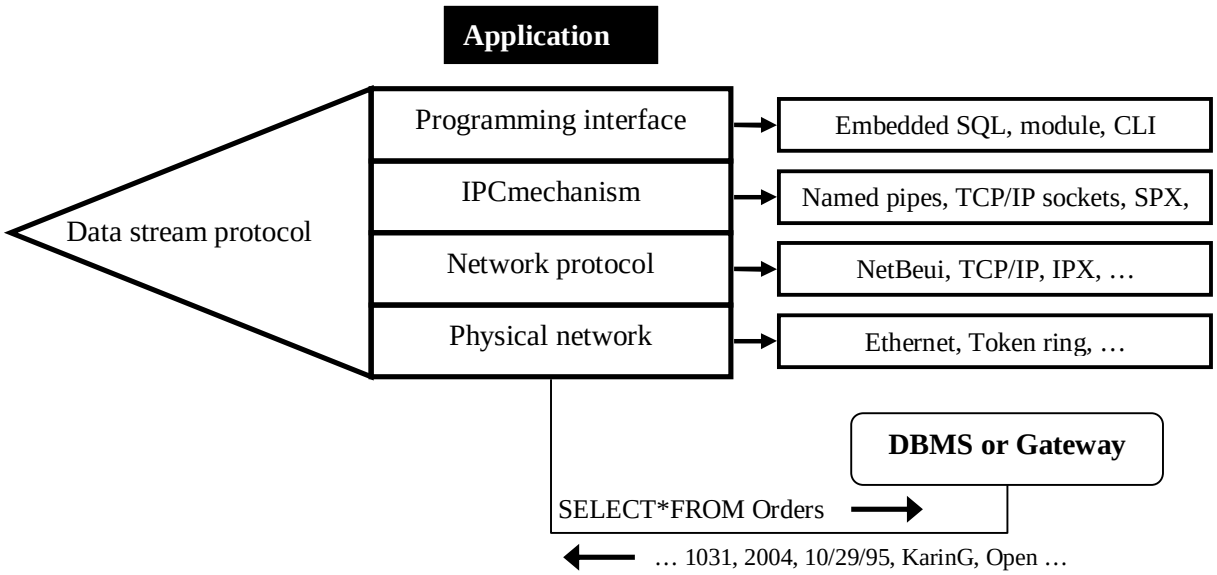


Рис. 2.1. Связь компонентов сетевого доступа к СУБД

Интерфейс программирования содержит обращения, сделанные прикладной программой. Эти интерфейсы (вложенный SQL, SQL-модули и интерфейсы CLI) являются специфическими для каждой системы управления БД, хотя они обычно основаны на стандарте Международной организации по стандартизации (ISO) или на ANSI.

Протокол потока данных. Протокол потока данных описывает поток данных, перемещенных между системой управления БД и пользователем. Например, протокол мог бы требовать, чтобы первый байт описывал то, что содержится в остальной части потока: оператор SQL, который должен быть выполнен, возвращаемое значение ошибки или возвращаемые данные. Формат остаточной части данных в потоке зависел бы от этого флажка. Например, ошибка могла бы кодироваться флажком с целочисленным кодом ошибки и битами, определяющими целочисленную длину сообщения об ошибке, и т.д.

Протокол потока данных – это логический протокол, и он независим от протоколов, используемых основной сетью. Таким образом, одиночный протокол потока данных может с успехом использоваться в ряде различных сетей. Протоколы потока данных, обычно составляющие собственность разработчиков, были оптимизированы, чтобы работать со специфическими системами управления БД.

Коммуникации между процессами. Механизм IPC (InterProcess Communication) обеспечивает связи между процессами. Например, могут использоваться именованные каналы, сокеты TCP/IP и сокеты DECnet. Выбор механизма IPC зависит от типа используемой сети и операционной системы.

Сетевой протокол. Сетевой протокол используется для транспорта потока данных по сети. К сетевым протоколам относятся NetBEUI, TCP/IP, DECnet, и SPX/IPX, и они являются специфическими для каждой сети.

Стандартная архитектура доступа к базе данных

Если внимательно посмотреть на компоненты доступа к БД, описанные в предыдущем разделе, то можно отметить, что два из них – программный интерфейс и потоковый протокол – являются хорошими кандидатами на стандартизацию. Другие 2 компонента – механизм IPC и сетевые протоколы – слишком зависят от сети и операционной системы и находятся на недопустимо низком уровне в 7-уровневой сетевой модели. Имеется также и еще одна цель – gateways, т.е. маршрутизаторы, которые также обеспечивают возможности для стандартизации.

Стандартный интерфейс программирования

Интерфейс программирования – возможно, наиболее очевидный кандидат на стандартизацию. Фактически, когда ODBC развивался, ANSI и Международная организация по стандартизации (ISO) уже обеспечивали стандарты для встроенного SQL и SQL-модулей. Хотя никаких стандартов для универсального доступа к различным БД еще не существовало, SQL Access Group – консорциум производителей серверов СУБД – внес на рассмотрение стандарт CLI – первооснову будущего ODBC.

Одно из требований для ODBC было то, чтобы прикладная программа в двоичном коде могла работать с многочисленными серверами СУБД. Это было возможно, поскольку ODBC не использует встроенный SQL или языки модулей. Хотя язык во встроенном SQL и языках модулей стандартизирован, каждый из них связан с dbms-специфическим прекомпилятором. Таким образом, прикладные программы должны были быть перетранслированы для каждой системы управления БД, чтобы возникающий в результате этого двоичный код мог работать только с одиночной системой управления БД. Однако если это было приемлемо для небольшого количества прикладных программ, разработанных специально для мини-компьютеров, то это было совершенно недопустимо в мире персональных компьютеров, рост числа которых увеличивался с каждым годом экспоненциально.

Поскольку интерфейс CLI может быть реализован через библиотеки или драйверы БД, которые резидентны на каждой локальной машине, можно использовать

различные драйверы, которые требуются для каждой системы управления БД. Поскольку современные операционные системы могут загружать такие библиотеки (типа динамических библиотек Windows) и вызывать из них необходимые функции непосредственно во время выполнения, одиночная прикладная программа может не только обращаться к данным из различных систем СУБД без перетрансляции, но может также обращаться к данным из многих БД одновременно. Новые драйверы для новых СУБД всегда можно купить у разработчиков или у третьих фирм, и пользователи могут устанавливать их на свои компьютеры без необходимости что-то изменять или перетранслировать. В силу этих причин интерфейс CLI был хорошим кандидатом на будущий стандарт ODBC, так как Windows-платформа идеально подходила для реализации заложенных в нем идей.

3.3.1 ODBC - технология доступа к БД

Зачем нужен ODBC

ODBC-драйверы могут понадобиться в любой операционной системе на сервере или на клиенте, когда вы разрабатываете собственные приложения, которые могут извлекать данные из различных БД, хранящихся как на SQL-серверах, Так и в специализированных реляционных таблицах. В принципе таблицы, существующие в виде отдельных файлов, на манер *.dbf (bBase) или *.db (Paradox) таблиц, - это уже дела «давно минувших дней». Теперь на SQL сервере можно сохранять все! Однако и отдельные таблицы еще довольно часто испольлзуются, чтобы обеспечить доступ к так называемым legasy systems – наследуемым системам. Тем не менее в нынешние времена, пожалуй, нет ничего более привлекательнее, как испольлзовать мощный SQL-сервер в качестве Data Warehouse и OLAP-сервера. Приложения для Unix или для Windows, написанные на обыкновенном C, Бейсике или Java, могут получать доступ к многочисленным данным в самых различных форматах через ODBC-источники. Эти приложения могут заниматься реконструкцией и преобразованием данных, реализовать различные мосты между данными, обеспечивать Data Pump, т.е. «перекачку» данных между различными базами, и т.д. – словом, создавать условия для складирования и инвентаризации данных. Освежим в памяти концепцию ODBC.

На рис. 2.2 схематически изображен эволюционный процесс разработки SQL-ориентированных приложений.

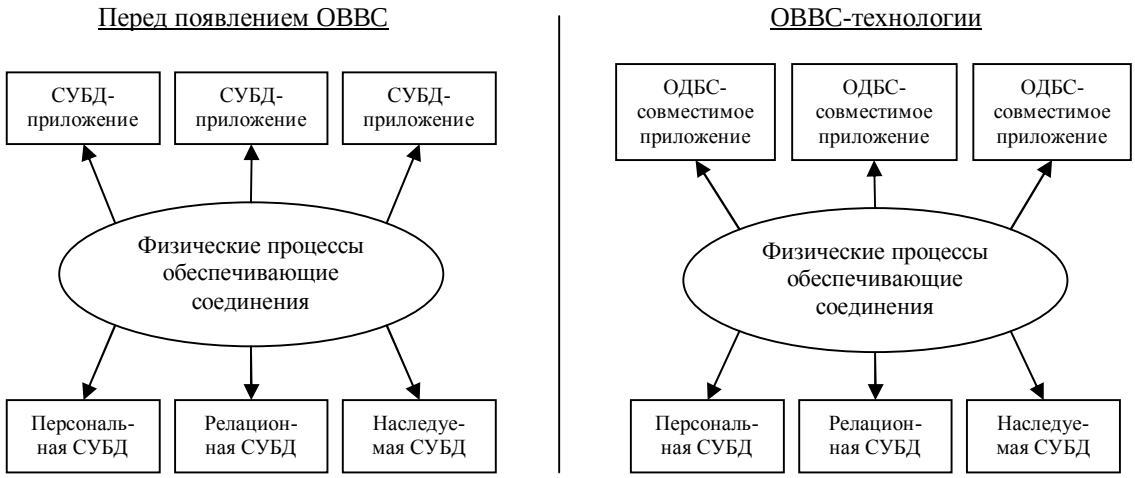


Рис. 2.2. Эволюция доступа к БД

Рис. 2.2 хорошо иллюстрирует, как изменился подход: вместо разработки в каждом отдельном случае уникального интерфейса доступа к различным БД, теперь

можно воспользоваться одним общим интерфейсом, позволяющим вместо трех сложных СУБД-приложений написать одно и достаточно простое.

ODBC Driver Manager

ODBC обеспечивает общий API-интерфейс для доступа к самым разнообразным БД. Этот интерфейс был разработан совместно в двух комитетах – X/OPEN и SQL Access Group (SAG) – и получил первоначальное название как стандарт CLI (Call-Level Interface).

Написать приложение, использующее стандартный интерфейс для доступа к любой произвольной БД, всегда и проще и экономнее, чем использовать конкретные, так называемые «родные» интерфейсы, которые быстро устаревают.

В концепции ODBC всегда используется один ODBC Driver Manager и несколько ODBC-драйверов или разделяемых библиотек для доступа к конкретным БД (рис. 2.3).

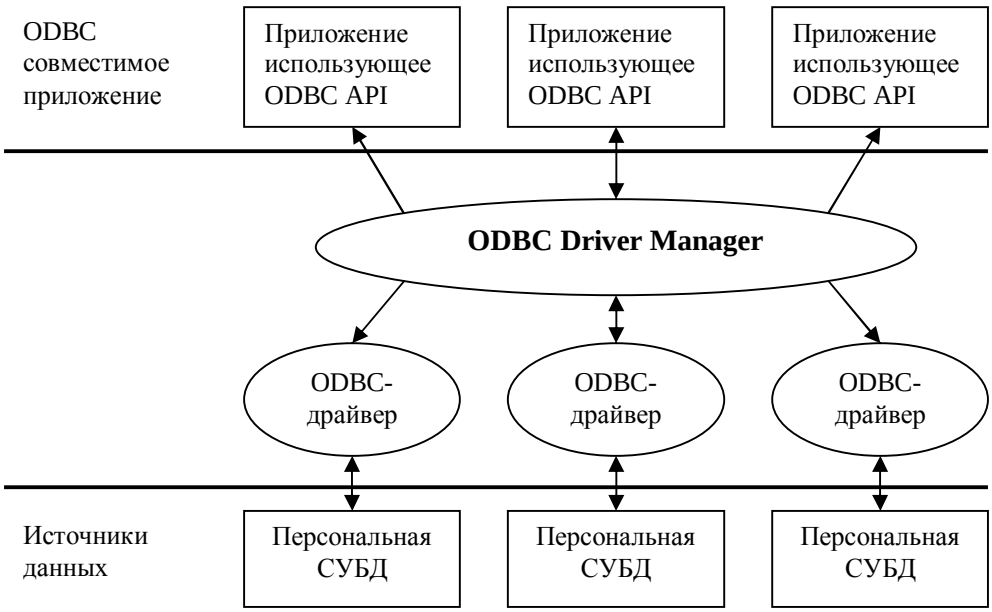


Рис. 2.3. Структура ODBC

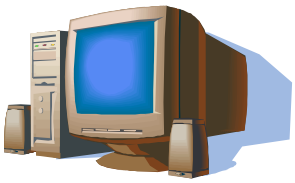
ODBC Driver Manager является центральным компонентом, который связывает ODBC-совместимые приложения и разделяемые объекты (вспомните, что в MS Windows это обычная библиотека DLL), которые выполняют специфические функции по обработке SQL-запросов к конкретной БД.

Главное при этом заключается в том, что подобный подход является вполне универсальным и механизм ODBC может работать практически в любой ОС.

3.3.2 Технология применения ODBC- драйверов

ODBC-архитектура

Существуют различные способы использования ODBC-архитектуры в сетях. Например, на рис. 2.4 показано, что все программные компоненты ODBC располагаются на одной машине. Там же находится и БД.



System A

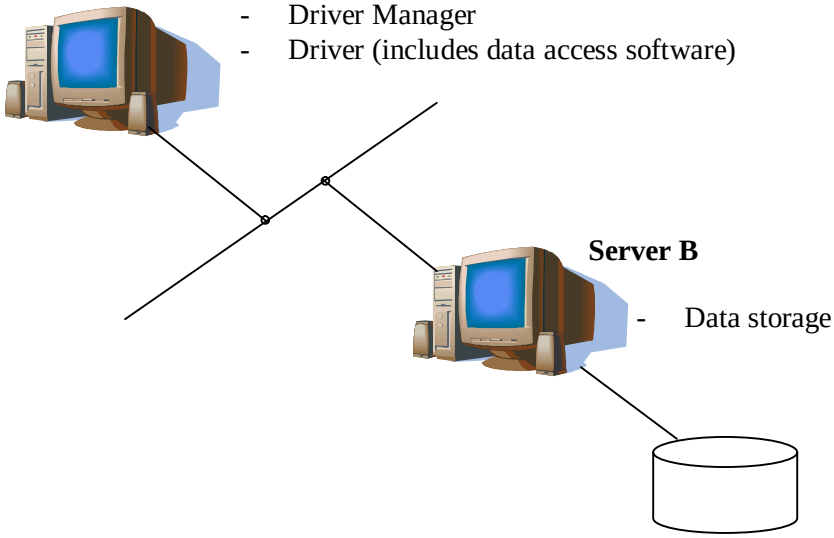
- Application
- Driver Manager
- Driver (includes data access software)
- Data storage

Рис. 2.4. Конфигурация одной машины

Во втором случае БД отделена от клиентского компьютера и для связи используется сетевой протокол (обычно TCP/IP или любой другой, который можно выбрать при инсталляции драйвера (рис. 2.5)).

Client B

- Application
- Driver Manager
- Driver (includes data access software)



Server B

- Data storage

Рис. 2.5. Конфигурация двух машин

Наконец, рис. 2.6 показывает использование еще одного промежуточного слоя – GateWay, который выполняет чисто служебную задачу и связывается с БД по асинхронной линии.

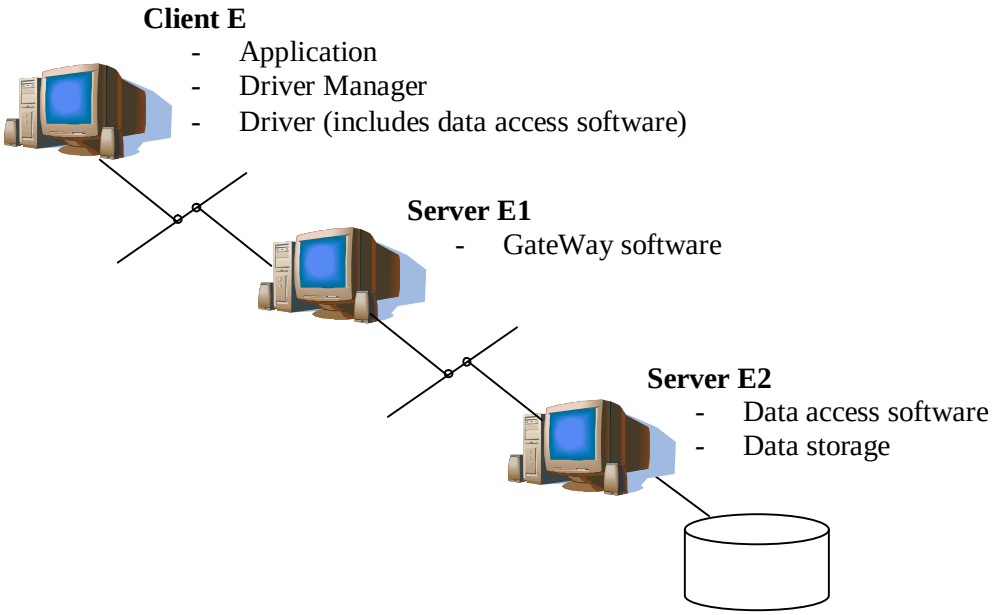


Рис. 2.6. Трехуровневая схема доступа через ODBC

Имеется и довольно высокая цена, которую необходимо заплатить за интерфейс высокого уровня, который маскирует различие между локальным файлом БД и присоединенным удаленным SQL-сервером. В этом случае недостатком будет более низкое быстродействие по сравнению с использованием ODBC.

3.3.3 Пример работы с ODBC-менеджером MS Office

Как уже говорилось ранее, для работы с ODBC-компонентами можно использовать ODBC-администратор, который располагается на Панели инструментов, носит название ODBC Data Sources и позволяет настроить для работы приложения один или несколько источников данных. Через эти источники, каждый из которых имеет уникальное имя, приложение может осуществлять доступ к базам данных, расположенным на различных серверах и управляемых конкретными СУБД.

Приложение может также конечно обращаться прямо к конкретной БД по ее имени, если заранее известно ее местонахождение, но тогда программист должен включить в текст прикладной программы специальные операторы обращения к БД, ее открытия и закрытия, и т.д. Такое обращение приложения вызывает необходимость участия программиста, что вызывает снижение общего уровня использования приложения с уровня пользователя-непрограммиста до уровня программиста и увеличивает детализацию приложения.

Использование ODBC-администратора упрощает настройку приложений на различные источники данных, делая ее, с использованием шаблонов, доступной для многих категорий пользователей. Ниже приводится пример такой методики настройки ODBC- источника на MS SQL-server:

1) войти через Панель инструментов в ODBC-администратор;
2) выбрать закладку "System DSN" (DSN-Data Source Name);
3) кнопкой "Add" перейти на следующую стрницу;
4) выбрать "SQL-server" и перейти кнопкой "Finish" к следущей странице;
5) задать в окне:
   * имя ODBC- источника (или указать из списка);
   * IP-адрес сервера БД через клавишу-"Client Config";
   * Имя сервера;
   * Указать Ваш "login" и "Password";
6) На следующей странице указать базу данных и отметить флажок в нижнем окошке;
7) Кнопкой "Next" перейти к следующей странице и в окне "Change the languish" указать - Русский и кликнуть кнопку "Finish";
8) Появится окно с кнопкой "Test Data Sourse", которую необходимо кликнуть;
9) В результате проверки правильности настройки ODBC-источника, который создавался, тестирование должно быть успешным, и тогда необходимо кнопкой "Finish" перейти на начальную страницу, чтобы убедиться в наличии источника на закладке;
10) Если тестирование отрицательное, то требуется возврат к предыдущим страницам-шаблонам ODBC-администратора для их проверки и коррекции.

На этом настройка ODBC-источника завершается. Аналогично настраиваются другие источники данных при любой их распределенности по узлам корпоративной сети.

3.3.4 Преимущества ODBC-архитектуры

Преимущества ODBC, главным образом, состоит в том, что вы можете использовать ODBC API, чтобы обратиться к любому источнику данных, для которого имеется ODBC-драйвер. Одна из главных целей проекта ODBC должна была обеспечить эквивалент эффективности, сопоставимый с родным API любого сервера СУБД. Это означает, что если однажды утром в понедельник ваш босс решит перевести архив данных всего отдела из Microsoft SQL Server в Oracle, то все приложения, использующие ODBC API, не потребуют фактически никакой модификации. Однако, чтобы использовать ODBC, вы должны хорошо знать SQL и ODBC API.

В модели ODBC приложение на Visual Basic делает запросы к диспетчеру драйвера через ODBC API. Диспетчер драйвера (ODBC32.DLL) является посредником между вашим приложением и фактическим ODBC-драйвером. Цель ODBC-диспетчера тройная: он загружает и выгружает ODBC-драйверы, отображает вызовы ODBC API в соответствующие функции ODBC-драйвера и дает возможность конечному пользователю создавать свои имена источников данных (DSN).

DSN – это фактически псевдоним к ODBC-источнику данных. Идея состоит в том, что пользователям не нужно знать имя БД и место ее расположения, чтобы соединиться с ODBC-источником данных. Вместо этого вся информация о различных БД должна быть собрана и сохранена в одном месте (запись Windows) и затем просто упоминаться, когда надо, через данное пользователем имя – DSN. Пользователь или администратор может создавать DSN через иконку ODBC в панели управления Windows. Можно также создать DSN программно, используя ODBC API ConfigDSN.

Диспетчер драйверов ODBC (ODBC Manager) можно испольлзовать, чтобы загрузить несколько ODBC-драйверов сразу. Например, несколько приложений на вашей машине могли бы одновременно обращаться к различным источникам данных, требующим различных драйверов. Это одна из главных причин использования диспетчера драйверов: диспетчер драйверов может управлять множественными параллельными соединениями со множеством ODBC-драйверов.

ODBC-драйвер ответственен за:

- выполнение функций ODBC API;
- установление соединения и посылку запросов к источнику данных;
- возврат результатов приложению;
- обработку ошибок;
- управление курсорами;
- управление транзакциями.

Главная цель состоит в том, чтобы изолировать приложение от частных интерфейсов конкретных серверов СУБД.

3.3.5 OLE- технологии доступа

Программирование с помощью стандарта OLE DB

Где вы храните ваши данные? Вы можете сказать, что это слишком общий вопрос и наиболее важные данные вы сохраняете на SQL-сервере, но вопрос-то был относительно ваших персональных данных. Почти любой скажет, что подавляющее количество данных приватного характера или повседневного спроса находится в электронных таблицах, документах, файлах-проектах и БД в виде обычных файлов на его собственном ПК. Если бы вас спросили, где вы храните ваши групповые неструктурированные данные, вы бы, вероятно, сообщили, что они находятся в папках Lotus Notes или в Microsoft Exchange или еще где-нибудь на сервере. Очевидно, что данные, с которыми вы постоянно имеете дело, разбросаны повсюду. Каким образом вы оформляете запрос к данным, когда обращаетесь к ним и изменяете что-нибудь? Проблема состоит в том, что каждый источник данных имеет различный интерфейс и язык запросов для доступа к данным, подобно SQL или QBE. До сего времени имелось несколько хороших стандартов, чтобы использовать универсальный интерфейс доступа к данным, и один из них – Open Database Connectivity (ODBC). Теперь, OLE DB дает еще один шанс для множества выборов. OLE DB – это метод для доступа ко всем имеющимся данным через стандартный COM-интерфейс, независимо от того, где и как данные сохраняются. Данные при этом могут быть самые разнообразные. Это могут быть реляционные БД, документы, электронные таблицы, файлы и сообщения электронной почты. В OLE-технологии, БД, как известно, становится компонентом, называемым провайдером (т.е. поставщиком информационных услуг). Фактически любой компонент, который непосредственно выставляет (экспозирует) свои функциональные возможности через OLE DB-интерфейс поверх локального формата данных является самым настоящим OLE DB-провайдером. Это может быть SQL-сервер, ISAM-файл, обычный текстовый файл или даже поток данных (рис. 2.7).

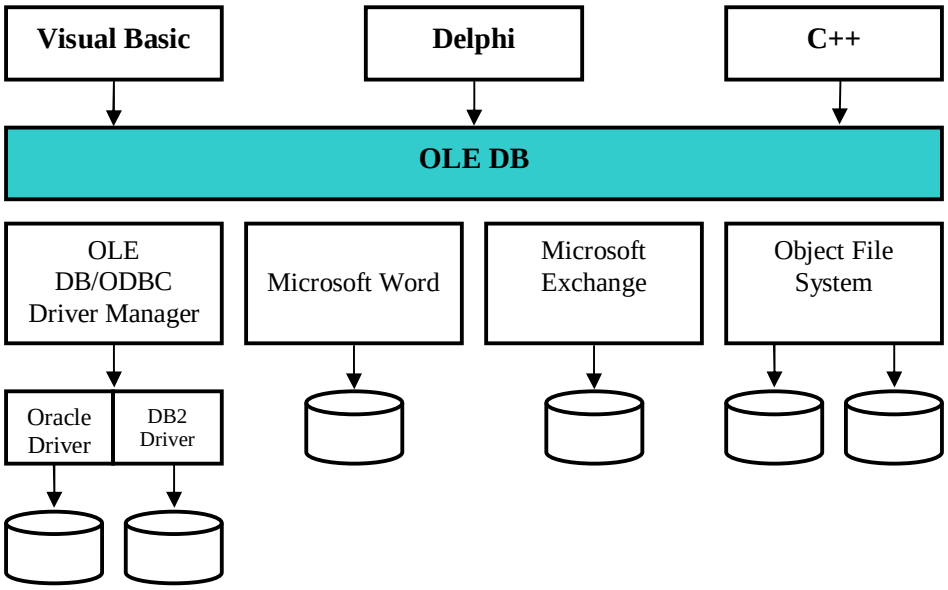


Рис. 2.7. Общая схема работы универсального интерфейса OLE DB

Точно также, как проводник (Explorer) в Windows 95/98 позволяет исследовать файловые системы, будущие проводники позволят исследовать любые данные.

Например, Microsoft мог бы построить провайдер данных, который знает, как обращаться и управлять данными, сохраненными в Microsoft Excel. Этот провайдер можно было бы включить как подфункцию в другой универсальный провайдер. Кроме того, индивидуальные компоненты OLE DB могут быть построены так, чтобы использовать наиболее передовые особенности стандарта на базе простых провайдеров данных, - такие компоненты называются сервис-провайдерами.

Благодаря этим сервис-провайдерам, подобных процессорам запросов или генераторам отчетов, можно обеспечить взаимодействие самых различных гетерогенных данных, представленных как таблицы. В отчетах можно использовать самые различные типы данных с удаленных машин без обязательного их транспортирования на локальную машину.

Точно так же, как имеются различные типы провайдеров данных, имеются различные типы OLE DB-потребителей данных. Потребителями данных могут быть заказные программы, написанные для работы с одним провайдером, или целые семейства программ, написанных для работы со множеством провайдеров. Так, в будущих версиях Microsoft Office такие программы, как Word, Excel MS Project, могут стать потребителями данных так же, как и их поставщиками, т.е. провайдерами. В этом случае Microsoft Word мог бы непосредственно обращаться к данным в Microsoft Excel и Microsoft Excel мог бы непосредственно обращаться к данным в файлах Microsoft Project.

Что касается инвестиций, вложенных в ODBC, то и они не были забыты. Microsoft просто дополнила ODBC Driver Manager соответствующим провайдером для доступа к ODBC-данным. Этот компонент немедленно обеспечивает OLE DB-потребителей SQL-данными, расширяя при этом класс прикладных программ, взаимодействующих с ODBC-драйверами.

3.3.6 Особенности работы с OLE DB-драйверами

В данном подразделе дается попытка описания основных интерфейсов и стандартов OLE DB, причем внимание фокусируется на главных из них. Кроме того, приводится пример разработки пользовательского приложения, применяющего интерфейс OLE DB и взаимодействующего с провайдером данных, являющимся частью этого примера, доступного в исходных кодах. Сам по себе, конечно, стандарт OLE DB слишком велик и сложен, чтобы рассмотреть его во всех деталях.

Предполагается, что читатели, приступившие к ознакомлению с OLE DB, уже неплохо разбираются в технологиях OLE и БД. Если к тому же они хорошо себе представляют, что такое ODBC, SQL и Microsoft’s Data Access Objects (DAO), это даже лучше. Им будет намного легче ориентироваться в новой, революционной технологии Microsoft.

Объектный тип определен как набор интерфейсов, которые объект должен включать в себя или экспозировать. Например, тип объекта Rowset определен группой интерфейсов, посредством которых пользователь может управлять табличными данными. Чтобы играть роль Rowset, объект должен уметь выполнять IRowset, IAccessor, IRowsetInfo и интерфейсы IColumnsInfo. Дополнительно сервисный провайдер может выступать в качестве хоста для реализации других взаимосвязанных интерфейсов.

Огромное число объектов и интерфейсов не должно угнетать разработчика. В сущности разработка спецификации, обеспечивающей доступ ко всем типам данных, сама по себе задача слишком монументальная, чтобы вообще не использовать функциональные возможности провайдера данных. Ведь ясно, что функционально полный SQL-провайдер СУБД всегда будет предлагать значительно большее количество возможностей, чем провайдер данных для текстового файла. Следовательно, вы не должны использовать все без исключения интерфейсы. При разработке OLE DB фирма Microsoft учла возможные подобия между провайдерами и обеспечила базовый уровень интерфейсов.

Интерфейсы базового уровня

Интерфейсы базового уровня – это набор минимума объектов и интерфейсов, которые должны поддерживаться как провайдерами данных, так и сервисными провайдерами (рис. 2.8).

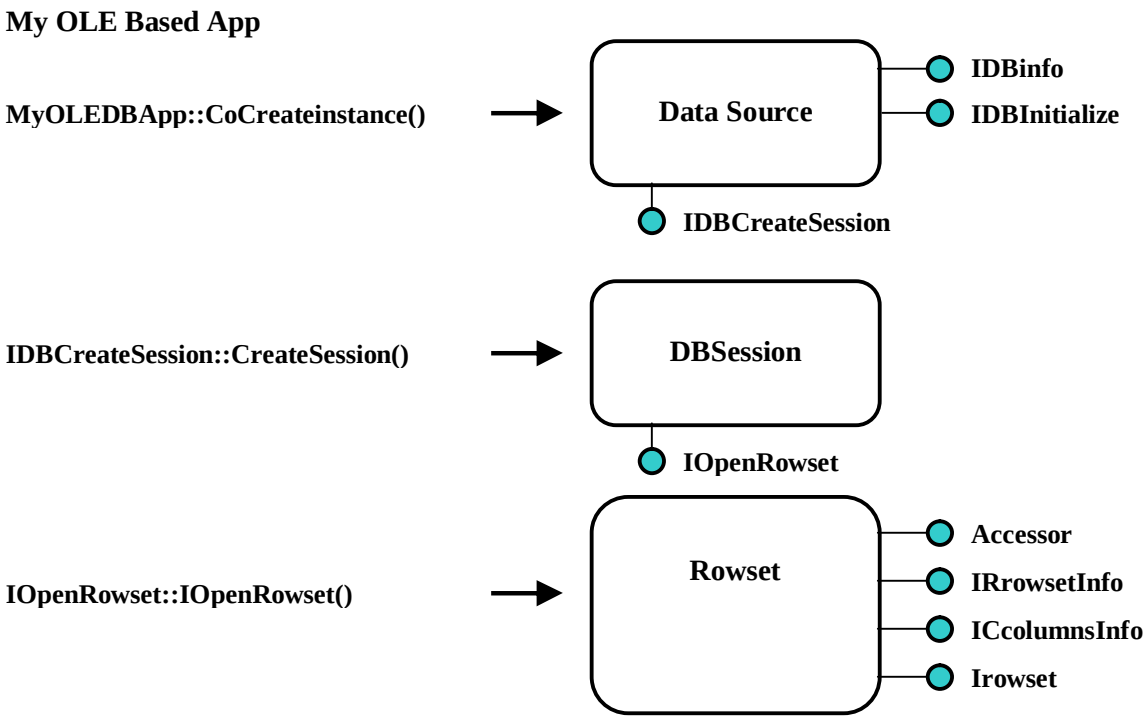


Рис. 2.8. Интерфейсы базового уровня

Так простой провайдер данных, который не поддерживает транзакции или запросы через объекты типа Command, может поддерживать только требуемые ему интерфейсы, в то время как более сложные провайдеры данных и сервисные провайдеры будут пользоваться этой возможностью.

Все провайдеры данных и сервисные провайдеры должны поддерживать объект источника данных – Data Source Object (DSO). DSO представляет соединение с источником данных, через который вы можете воздействовать на данные. DSO создает многократные сеансы связи посредством объекта DBSession. Область связи с источником включает информацию об имени и расположении источника данных и опознавательную информацию, если данные находятся вне защищенной среды.

3.3.7 Доступ по сети. Объектные интерфейсы Microsoft на базе ODBC – DAO, RDO

Хотя стандарт ODBC утвердился в мире во многом благодаря стараниям Microsoft, в самой корпорации этот стандарт никогда не был определяющим. У Microsoft имеются довольно много способов эффективно работать с БД, опираясь на разработанный еще со времен MS Access 1.0 процессор БД Microsoft Jet. Все предыдущие средства разработки Microsoft справедливо критиковали за непременное стремление подгрузить библиотеки MS Jet, даже когда это не нужно разработчику. Начиная с версии Enterprise Edition of Visual Basic 4.0 все кардинально изменилось. Хотя теперь с появлением VB5 и VB6 названия некоторых библиотек изменились (например, вместо MSJ3032.dll и DAO3032.dll стали использоваться MSJET35.dll и DAO350.dll), суть архитектуры, основанная на Data Access Objects (DAO) и Remote data objects (RDO), осталась прежней. Эта архитектура основывается на следующих компонентах:

1. машине БД (Microsoft® Jet database engine)
2. средствах контроля и управления (The Jet data control and data-bound rols)
3. объектах доступа к данным (Data Access Objects)
4. прикладном интерфейсе ODBC (ODBC and the ODBC API)
5. управлении удаленными объектами RDO (Remote data objects)
6. дистанционной связи модулей (Remote automation)

Разберем теперь подробно, в чем суть этой архитектуры, и в конце подраздела приведем пример VB-приложения для сравнения скорости выполнения трех методов доступа к БД, обсуждаемых ниже (DAO, RDO и ODBC)

Объектный интерфейс Microsoft на базе OLE DB-ADO

В настоящее время не имеется недостатка в средствах для разработки полноценных Internet/Intranet приложений. Более того, наблюдается даже некоторый их переизбыток. Тем не менее большинство разработчиков продолжают пользоваться проверенными средствами: Java-апплетами, Java-скриптами или, на худой конец, CGI-приложениями, продолжают вершить подавляющую часть работы в Internet. И все-таки заметно больше стало узлов, построенных исключительно на удобных технологиях Microsoft. Еще недавно .ASP–расширения в именах файлов вместо привычных .HTM и .HTML были многим попросту непонятны, однако теперь вместе с подавляющим

распространением Web-узлов, построенных на базе ASP (Active Server Pages), многие специалисты вынуждены более внимательно присмотреться ко всем инновациям, внедряемым Microsoft в этом стремительно расширяющемся секторе рынка. Основным недостатком всех прошлых решений в части разработки приложений для Internet был их громоздкий интерфейс для доступа к БД, основанный на JDBC/ODBC, и неудобные средства визуализации данных, получаемых из баз. Предложив в качестве универсального интерфейса для доступа к различным источникам данных OLE DB, а в качестве интерфейса к интерфейсу – удобную и компактную модель ADO (Active Data Objects), Microsoft разом решила трудные задачи и поставила многих программистов перед нелегким выбором: осваивать им Visual Basic или нет. Дело в том, что если соблазниться удобными и привлекательными сторонами Web-сервера Internet Information Server 3.0/4.0, то поневоле придется выбирать между JavaScript и Visual Basic Script, а уж сделав выбор в пользу последнего, придется полностью забыть о Netscape Navigator и предпочесть ему IE 4.0/5.0. Многие программисты пошли именно этим путем вовсе не оттого, что они внезапно разлюбили Netscape, нет, они вынуждены были сделать это из-за того, что многие вещи в Netscape стали трудно выполнимы, особенно когда приходиться посещать сайты с ASP-страницами. Кроме того, ADO-интерфейс лучше и проще всего реализован в VB и в его диалекте VBScript – основном языке сценариев MS Internet Explorer.

Прежде чем перейти к описанию ADO, вначале все-таки вспомним наиболее характерные черты интерфейса OLE DB, а затем уже перейдем к конкретным примерам программирования с использованием ADO.

ODBC, OLE DB и ADO

ODBC и OLE DB – это прикладные интерфейсы Windows для осуществления доступа к данным. Более старая спецификация ODBC обеспечивает доступ к данным, и прежде всего к реляционным базам, основанным на использовании языка SQL. OLE DB – это спецификация следующего поколения Microsoft, которая позволяет осуществлять доступ к данным через так называемых провайдеров, т.е. поставщиков данных, тем самым делая круг данных намного шире. Эти поставщики могут включать нереляционные системы БД, системы электронной почты и системы CAD/CAM, а также классические реляционные системы БД.

OLE DB не заменяет ODBC. Фактически OLE DB включает поставщик данных, который позволяет вам использовать новый стандарт вместе с традиционными источниками данных ODBC. Важно отметить, что Microsoft планирует использовать

OLE DB в качестве универсального стандарта для доступа к любым данным, независимо от того где или как они сохраняются на предприятиях.

Как ADO вписывается в эту картину? ADO – это интерфейс высокого уровня к стандарту OLE DB. Вы можете умело использовать его в своих программах, чтобы получать доступ через источники ODBC к провайдерам данных OLE DB's. Пока что чаще всего используются источники, указывающие на реляционные данные. В будущем, однако, вы сможете использовать ADO, чтобы обратиться к вашей системе электронной почты или некоторому другому нереляционному поставщику данных.

Microsoft публично декларировала, что ADO в конечном счете заменит мириады разнообразных моделей доступа к данным различных компаний, включая методы самой Microsoft, такие, как DAO и RDO, так что имеется в виду, что ADO – это не только метод для реализации Internet/Intranet доступа к данным, ADO – это модель доступа к данным, предлагаемая в будущем.

3.3.8 Объектно-распределенные системы на базе технологий MTS и MSMQ.

Средства создания распределенных приложений

В состав Windows NT 4.0 Enterprise Edition вошли Microsoft Transaction Server 1.1(MTS) и Microsoft Message Queue Server 1.0 (MSMQ). Microsoft Transaction Server сочетает в себе функции монитора транзакций и брокера объектных запросов и предназначен для управления компонентами приложения, вынесенными в промежуточное звено между клиентом и сервером.

Приложение, построенное из компонентов, имеет массу преимуществ по сравнению с традиционным программированием. Главное достоинство состоит в возможности распределения разных компонентов по разным компьютерам. Тем самым достигается балансировка нагрузки, и вычислительные ресурсы сети используются максимально эффективно. Затраты на создание компонентных приложений существенно ниже, так как компоненты реализующие основные прикладные функции, могут повторно использоваться в различных приложениях. Благодаря тому, что разные компоненты могут создаваться на разных языках программирования, отпадает необходимость переводить всех разработчиков в проекте на единый инструментарий - каждый вправе использовать то средство, в котором он наиболее уверенно себя ощущает (лишь бы оно позволяло создавать компоненты), и следовательно, может добиться наибольшей эффективности. Microsoft Transaction Server позволяет быстро разрабатывать такие приложения, обеспечивая их согласованную работу и транзакционную целостность.

Таким образом, выгоды от применения MTS одинаково очевидны как для разработчиков, так и для администраторов. С точки зрения разработчиков много уровневых приложений применение MTS обусловлено необходимостью поддержки транзакционной целостности на уровне приложения. С точки зрения администратора MTS хорош тем, что берет на себя автоматическое управление системными ресурсами: процессами, потоками, памятью, соединениями с БД экземплярами компонентов и т. д. Известно, что дефицит ресурсов крайне негативно сказывается на масштабируемости, так как при возрастании числа клиентов они вынуждены либо подолгу простаивать в ожидании, либо попросту отваливаться. Применение MTS гарантирует высокую масштабируемость приложений в многопользовательской среде, притом, сами компоненты могут разрабатываться как однопользовательские.

Возможности Microsoft Transaction Server

Transaction Server поддерживает структуру многоуровневых приложений и компоненты. Клиенты первого уровня обеспечивают пользовательский интерфейс и некоторое количество логики приложения. Серверы приложений среднего звена берут на себя большую часть логики приложения. Они работают под управлением Transaction Server. БД конечного звена занимаются хранением информации для приложения.

Transaction Server предполагает компонентные приложения и компонентный подход к их построению. Предполагается, что компоненты дают программистам наилучший способ создания распределенных приложений. Их объектно-ориентированные свойства, такие, как модульность, инкапсуляция и наследование, позволяют легко расширять приложения и облегчают поддержку. Следовательно, при разработке они легко используются повторно, а также упрощают разбиение приложения и его распределение.

COM и ActiveX

Transaction Server поддерживает Microsoft Common Object Model (COM) и технологию ActiveX. COM обеспечивает соединяемость и определяет интерфейсы и протоколы внутри и между компонентами приложения. Разработчики проектируют взаимодействие составных частей приложения на основе ActiveX-модулей. Таким образом, упрощенно можно представлять Transaction Server как законченную двухуровневую серверную среду для ActiveX.

Когда пользователь или программа ссылается на интерфейс компонента ActiveX, посылая сообщение или событие, COM определяет компонент и перенаправляет ему данное сообщение или событие. Компонент, на который происходит ссылка, может быть локальным или находиться на удаленном компьютере. Локальная соединяемость обеспечивается механизмами межпроцессного взаимодействия самой операционной системы. Удаленная соединяемость обеспечивается протоколом TCP/IP.

Разработчикам остается только вставить в своем коде ссылку на соответствующий объект ActiveX. Все остальное сделают COM и DCOM (распределенное сетевое расширение COM). Описанные ниже сервисы службы каталогов предоставляют COM и DCOM всю необходимую информацию для разрешения ActiveX-ссылок.

Transaction Server задействует механизмы безопасности Windows NT. Windows NT обеспечивает вход в систему через пользовательский ID и пароль, а также управление доступом к ресурсам с помощью администрирования пользователей и групп пользователей. Transaction Server расширяет эти возможности за счет управления доступом к индивидуальным компонентам приложения. Контроль может выполняться как с помощью административных функций Windows NT по отношению к пользователям и группам, так и программным путем внутри компонентов приложения.

Службы каталогов обеспечиваются в Transaction Server и COM через Windows Registry, который содержит информацию о пользователях, группах, приложениях, именах и интерфейсах ActiveX. Он предоставляет все необходимое для поддержки многоуровневых приложений в Microsoft Windows. Создание и поддержка registry является административной функцией, разработчики избавлены от необходимости следить за этим.

Разработчики в своих приложениях просто пишут код на SQL для получения информации.

Transaction Server имеет монитор транзакции, который контролирует доступ транзакций к менеджерам ресурсов. Транзакции могут обращаться к единственному менеджеру ресурсов или через поддержку протокола Microsoft Dislnhuled Transaction Coordinator (DTC) транзакция могут согласовывать и синхронизировать свои доступ к нескольким менеджерам ресурсов. В начальной версии Transaction Server в роли менеджеров ресурсов могут выступать Microsoft SQL Server и БД, поддерживающие интерфейс ODBC. Следующие версии будут поддерживать протоколы XA, SNA LU6.2 и TIP (Transaction Internet Protocol) и, следовательно, будут расширены понятия менеджеров ресурсов и среды обработки, могущие участвовать в транзакциях.

Программистские усилия по поддержке транзакций сводятся к минимуму. Пишущим код не нужно вставлять в программу операторы Beg transaction. End transaction или Abort transaction. Транзакции определяются путем задания свойства компонента приложения. Если компоненты помечены как "transactional". Transaction Server будет управлять транзакциями на всем периоде его выполнения. Все компоненты, на которые он ссылается, автоматически принимают участие в транзакции. Если компоненты помечены как транзакционные, то все их выполнение будет заключено внутри единой транзакции. Shared Property Manager (Менеджер разделенных свойств) в составе Transaction Server предоставляет механизм разделения доступа к глобальной информации между серверными процессами. Разработчики могут его использовать для широкого круга вопросов, включая мониторинг и управление исполнением, чередование процессов и т. д. Shared Property Manager выполняет все необходимые действия по разделению информации между приложениями. Разработчики могут спокойно концентрировать усилия на программной логике. Поскольку действия выполняются централизованно, процесс отладки и сопровождения распределенных приложений проходит более гладко, нежели в тех случаях, когда созданием и управлением глобальными данными занимаются отдельные разработчики.

MTS обеспечивает плотную целостность распределенных транзакций к протоколу двухфазной фиксации. Тем не менее существуют ситуации, когда более выгодно работать асинхронно. К ним относятся приложения, в которых инициатор транзакции должен продолжать работу, не дожидаясь результатов обработки запроса. Приложения, обменивающиеся сообщениями могут не

быть постоянно в режиме online. Наконец, требования асинхронной коммуникации могут диктоваться плохими каналами связи, мобильными пользователями и т. д. Во всех этих случаях для обеспечения асинхронного взаимодействия необходимо иметь некий механизм промежуточного хранения и диспетчеризации пересылки (store and forward).

К преимуществам подобного подхода следует отнести гарантированную доставку, возможность маршрутизации сообщений и назначения им приоритетов и известную независимость от качества каналов связи. Такой механизм разрабатывался в Microsoft под кодовым названием Falcon и вошел в Windows NT 4,0 Enterprise Edition как сервер очередей сообщений - Microsoft Message Queue Server (MSMQ 1.0). Под сообщением MSMQ понимается запрос произвольной природы (или ответ на запрос), самодостаточный в плане координат отправителя/получателя и содержания. Сообщения, запрашивающие участие менеджера ресурсов в транзакции, попадают в очереди MSMQ. Очереди сообщений схожи с почтовыми ящиками сервера электронной почты. Менеджер очередей MSMQ также работает подобно серверу электронной почты, решая, кому, когда и как доставить сообщение. Топология MSMQ напоминает структуру Microsoft Exchange Server: единая распределенная БД на Primary Enterprise Controller (PEC) предоставляет в общее пользование информацию о доступе к сообщениям для всех компьютеров, на которых запущен MSMQ.

Несмотря на достаточно близкие аналогии с электронной почтой, MSMQ имеет несколько существенных отличий. Он разрабатывался специально для взаимодействий "приложение-приложение", содержание сообщения имеет значение только для его отправителя и получателя, существует возможность настройки гарантии доставки (экспресс, гарантированная и транзакционная), наконец, MSMQ является более простым, компактным, быстрым (и как следствие, более дешевым), чем традиционный почтовым сервер. Технология очередей сообщений может служить шлюзом к серверам БД, работающим на других операционных системах, обеспечивая многоплатформенные распределенные транзакции. Поддержка MSMQ API на плата мах IBM MVS, CICS, Sun Solaris, HP-UNIX, AIX UNIX, OS/2, VMI, AS/400 обеспечивается продуктами Level 8 Systems (http://www.level8.com). Возможно также отображение прикладного интерфейса сервера очередей IBM MQSeries а вызовы MSMQ API.

Программный интерфейс MSMQ достаточно прост: отправитель имитирует возможность открыть очередь, установить режим доставки и продолжительность попыток, записать сообщения в очередь, задать их свойства, значить очередь для приема ответа. Аналогично получатель открывает очередь, читает сообщения и т, д. Одну очередь могут читать и модифицировать при наличии соответствующих прав несколько приложений, при этом можно читать и изменять как свои, так и удаленные очереди. MSMQ имитируя дополнительные интерфейсы а Microsoft Exchange Server, MAPI. MTS рассматривает MSMQ как менеджер ресурсов, что дает возможность включать операции над сообщениями серверов очередей в контекст единой распределенной транзакции.

Возможности Message Queue Server

Схема обобщенной архитектуры Microsoft Message Queue Server приведена па рис. 3.20.

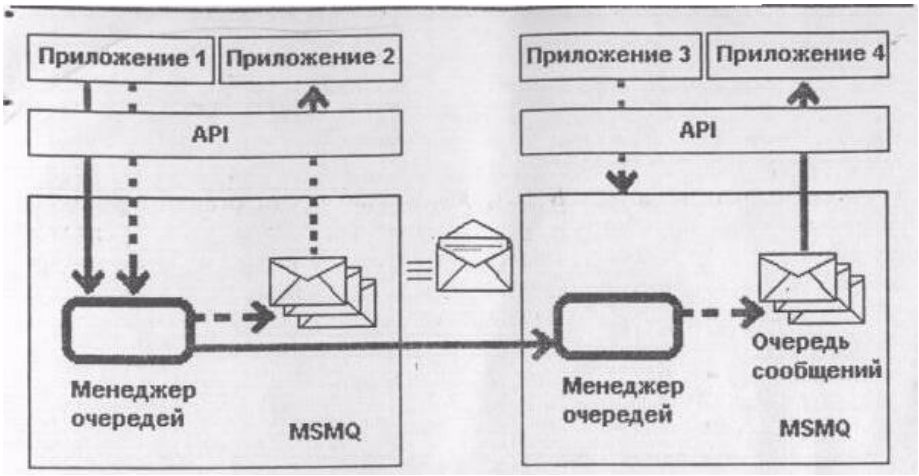


Рис. 3.20. Обобщенная архитектура сервера сообщений

Передача сообщений при отсутствии прямых соединений (Connectionless messaging)

Наличие очередей для сохранения и передачи сообщений избавляет приложения от необходимости обрабатывать нештатные ситуации поведения сети. Сервер очередей использует модель соединения вне сеанса связи (sessionless model), поэтому не требуется поддержка адресатом и отправителем одних и тех же сетевых протоколов. Сервер очередей поддерживает IP и IPX. Кроме того, фирма Level 8 Systems предполагает обеспечить поддержку SNA для обращений к компьютерам IBM (MVS и CICS). Сервер очередей обеспечивает следующий базовый набор функциональных возможностей:

1) Установка приоритетов передачи (Network traffic prioritization). Возможность расстановки приоритетов позволяет в первую очередь отправлять более срочные сообщения, что гарантирует хорошее время отклика для критических приложений за счет менее важных.

2) Динамические очереди (Dynamic queues). Информация об очередях хранится в БД, которая может изменяться динамически и реплицироваться. Подобный подход позволяет администраторам менять протокол и положение (помимо других параметров очереди), не оказывая влияния на приложения передачи/приема сообщений. Администратор может вносить указанные изменения с помощью MSMQ Explorer, где бы он ни находился.

3) Маршрутизация (Routing). Сервер очередей поддерживает динамическую маршрутизацию и доставляет сообщения, используя путь с наименьшей ценой доставки. Когда работа сети на этом маршруте нарушается, сервер автоматически выбирает следующий по стоимости маршрут. Назначение цены участков передачи сообщений производится администратором при помощи административной консоли. Администраторы также могут назначить серверу очередей роль коммуникационного концентратора для обработки всего графика между двумя узлами.

4) Гарантированная доставка (Guaranteed delivery). Сообщения могут сохраняться в дисковой очереди для обеспечения гарантированной доставки (Joumaling) или уведомления о невозможности доставки (letter). Для очень срочных сообщений сервер очередей предоставляет возможность сохранять сообщения в памяти до того момента, когда могут быть доставлены адресату. В последнем случае остановка сервера вызовет потерю сообщения.

5) Защищенность (Security). Сервер очередей использует списки очередей доступа (ACL) Windows NT для определения прав доступа к очереди. Применяются также другие системы защиты, основанные на использовании открытых (Public) ключей (система защиты основана на применении Crypto API с RSA-провайдером). Для проверки целостности сообщений применяются контрольные суммы. Сервер очередей использует алгоритмы шифрования RC2 и RC4; по умолчанию выбирается алгоритм RC2. Для шифрования сообщения приложение просто присваивается, атрибут "private", все остальное будет сделано автоматически.

6) Поддержка ActiveX. Как любой сервер, сервер очередей должен иметь открытую архитектуру. В нашем случае разработка приложений , сервер очередей средствами VB облегчается использованием трех MQ- компонентов:

Queue (MQQueue) - открывает доступ к очереди;

Message (MQMsg) - предоставляет доступ посланным и принятым сообщениям; **Directory Service (MQDS)** - используется для поиска существующих и создания новых очередей.

Все эти компоненты обеспечивают доступ к API для программирования всех функций сервера. Поддержка ActiveX позволяет обращаться к серверу очередей из Microsoft Transaction Server, Internet Information Server, активных Web-страниц (ASP) и любых приложений, поддерживающих C вызовы. Средствами C и C++ можно обращаться к API напрямую.

7) Поддержка транзакций. Разработчики могут включать операции сервера очередей в контекст транзакции при обращении к другим источникам данных, способным восстанавливать состояние, например к серверу БД. Для сохранения целостности данных операции сервера очереди фиксируются или откатываются назад вместе с остальными ресурсами в пределах транзакции. Например, если приложение вносит модификации в БД и посылает сообщение другому приложению внутри одной транзакции, то может возникнуть ситуация, когда транзакция в БД должна быть отменена. При этом операция по передаче сообщения также будет отозвана сервером сообщений. Никакая операция по передаче сообщения не считается завершенной до тех пор, пока не зафиксирована транзакция, в которой эта операция участвует. Это значит, что адресату никогда не будет доставлено сообщение из отмененной транзакции. Аналогичные действия сервер очередей предпринимает при приеме сообщения. Если транзакция отменена, отменяются и все операции приема. Очень важно, что сами сообщения при этом не теряются, а возвращаются в очередь и остаются доступными для последующих транзакций.

Сервер очередей тесно интегрирован с Microsoft Transaction Server, и все вызовы от ActiveX-компонентов, участвующих в транзакциях, автоматически включаются в активную транзакцию MTS. Когда приложения используют транзакции для "упаковки" сообщений, сервер очередей гарантирует, что каждое сообщение будет доставлено только 1 раз и сообщения внутри транзакции будут доставлены в том порядке, в котором они были посланы.

Администрирование

Графический интерфейс администратора позволяет динамически управлять всеми компьютерами корпоративной сети с единой консоли.

Административная консоль использует разделяемую и реплицируемую БД для

поддержания актуальной информации. Администратор всегда имеет возможность добавления, удаления, модификации и перемещения очередей. Отсутствие статических таблиц конфигурации и предопределенных путей улучшают масштабируемость и снижают затраты на администрирование системы.

Вся система администрирования сервера очередей концентрируется вокруг индивидуальных менеджеров очередей. Сообщения перемещаются между менеджерами очередей посредством очередей передачи (transmission queue). Менеджеры очередей группируются в узлы (site) и связаны не "каждый с каждым", а через общее сетевое пространство предприятия. Информация об узлах и связях является разделяемой и используется всеми компонентами, участвующими в процессе. Это служит гарантией того, что сообщения будут маршрутизироваться корректно и оптимально.

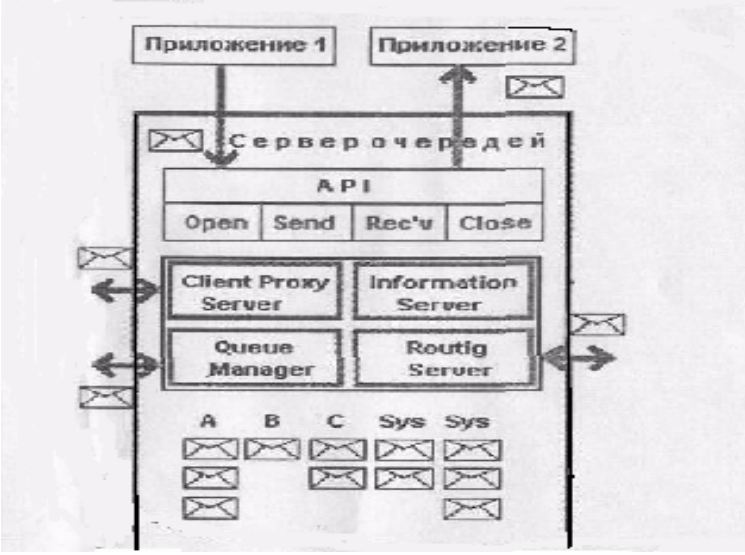
Конфигурация

Сервер очередей может быть сконфигурирован в одной из трех модификаций: сервер, рабочая станция и клиент.

MSMQ как сервер

Сервер (рис. 3.21) поддерживает всю доступную функциональность. Сервер способен:

• обслуживать API-вызовы от приложений, выполняющихся на той же самой машине;



• выступать как клиентский "прокси-сервер" (client proxy server) от имени клиентов сервера очередей на других машинах;

• сохранять сообщения в очередях на машине, где выполняется сервер очередей;

• выполнять функции маршрутизации сообщений;

• поддерживать соединения с другими платформами через MSMQ Connector

Кроме того, серверная конфигурация поддерживает MSMQ Information service - каталог для информации о маршрутизации и конфигурации MSMQ Explorer - основную административную консоль.

Рис. 3.21. Схема работы MSMQ-сервера

MSMQ как рабочая станция

В конфигурации "рабочая станция", или MSMQ Independent **Client** (рис. 3.22), сервер очередей может обслуживать API-вызовы от приложений, выполняющихся на той же машине, и способен обслуживать локальные очереди сообщений, т.е. сообщения хранятся на той же машине, где и выполняется. Рабочая станция MSMQ не содержит служб маршрутизации "прокси" , или служб MSMQ Information Services. Обычно рабочая станция MSMQ выступает как клиент "полного" сервера очередей.

Если отказала сеть или если клиент вовсе не подключен к сети (например, мобильный пользователь), приложения могут продолжать отправлять и принимать сообщения из локальной очереди.

После восстановления или инициализации соединения рабочая станция MSMQ отправляет все накопленные в локальной очереди сообщения, предназначенные для другой машины. Точно так же все сообщения, предназначенные для рабочей станции MSMQ, поступают к ней в очередь с других машин. Подобная функциональность делает рабочую станцию MSMQ очень привлекательным решением, когда необходимо обеспечить хранение и повторяющуюся передачу сообщений с обычного настольного компьютера.

Рабочая станция MSMQ не может выполнять функции маршрутизации (т. е. выступать в качестве промежуточного агента). Пересылка и прием сообщений от других рабочих станций могут выполняться только при точном указании имен очередей назначения. Рабочая станция MSMQ выполняется в среде Windows 95, Windows NT Workstation и Windows NT Server.

Рис. 3.22. MSMQ в конфигурации « рабочая станция « MSMQ-клиент

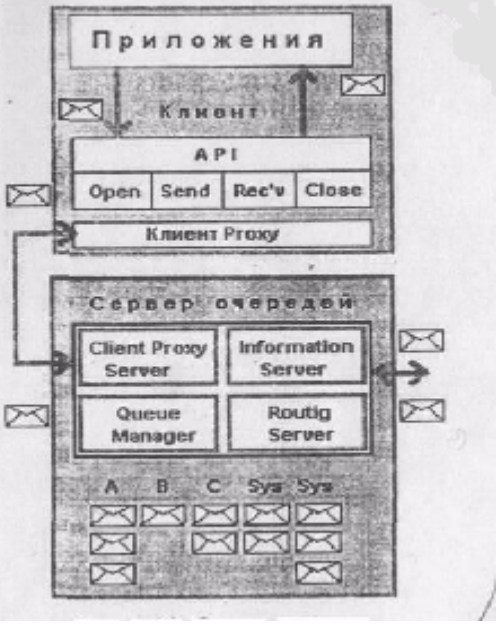


Рис. 3.23 Схема работы MSMQ как клиент

Конфигурация "клиент", или MSMQ Dependent Client (рис. 3.23), предоставляет все API-сервера очередей (такие, как Open, Close, Send, Receive) для приложений, выполняющихся па машинах, не имеющих ни сервера, ни рабочей станции для обслуживания очередей. Когда приложение выполняет API-вызов, клиент осуществляет пересылку этого вызова к клиентской "прокси"-машине, на которой исполняется серверная конфигурация сервера очередей (конфигурация "рабочая станция" не может обслуживать запросы от MSMQ-клиентов). Клиентская "прокси"-машина повторяет API-вызов серверу очередей, который на этот раз исполняется применительно к конкретной очереди. В данной ситуации сервер отвечает за все вопросы связанные с управлением сообщениями, хранением и маршрутизацией.

Основным преимуществом использования MSMQ-клиентов являться минимальные требования к дисковому пространству и объему памяти. Однако при отказе сети ни одно приложение не сможет выполнить вызов к API-сервера.

Конфигурация "клиент" поддерживается на Windows 95, Windows Workstation и Windows NT Server.

3.3.9 Мониторы обработки транзакций. Серверы транзакций. Серверы сообщений

Мониторы обработки транзакций

Мониторы обработки транзакций (transaction processing monitor, TPM) — самый старый тип технологии распределенных систем, которая появилась в 70-х годах в среде больших универсальных ЭВМ [9-12]. Одной из первых прикладных систем была интерактивная среда поддержки, созданная компанией Atlantic Power and Light для совместного использования прикладных служб и информационных ресурсов в режимах пакетной обработки и с применением среды с разделением времени. TPM (например, IBM CICS) стали главным инструментом построения высоко масштабируемых решений для сетевых прикладных систем. Однако в начале 90-х популярность TPM начала падать; причиной стало появление продуктов категории TPM «Lite» — мониторов обработки транзакций внутри СУБД. Их функциональные возможности были близки к обычным TPM, и с возрастающей тогда популярностью двухзвенной архитектуры они первоначально обеспечили хорошее решение для создания распределенных прикладных систем.

К концу десятилетия ситуация изменилась. Оказалось, что в то время как обычный монитор транзакций может легко поддерживать тысячи пользователей, их производительность становится недопустимо низкой, когда совокупность пользователей превышает 100. Сегодня TPM снова вернулись, но теперь это технология, которая воплощает среднее звено в трехзвенной архитектуре.

Клиенты связываются с мониторами, посылая запросы на транзакции. Так как коммуникации асинхронны, после того, как запрос послан, клиент может выполнять другие задачи. Каждый запрос ставится в очередь, и монитор может обработать его позднее. Поддерживая различные схемы приоритетов, можно определить, в каком порядке запросы принимаются из очереди на обработку. Мониторы постоянно поддерживают установленное число подключений к базам данных, поэтому непосредственного соответствия между клиентами и подключениями к базе данных нет. Вместо этого мониторы мультиплексируют запросы от различных клиентов по одному и тому же подключению — точно так же, как и серверы приложений.

Одно из достоинств процессора транзакций — способность обращаться к гетерогенным источникам данных (реляционные и сетевые базы данных, плоские файлы и т.д.). Поэтому каждый TPM содержит логику доступа к данным, которая выполняет соответствующее преобразование, связанное с различными источниками.

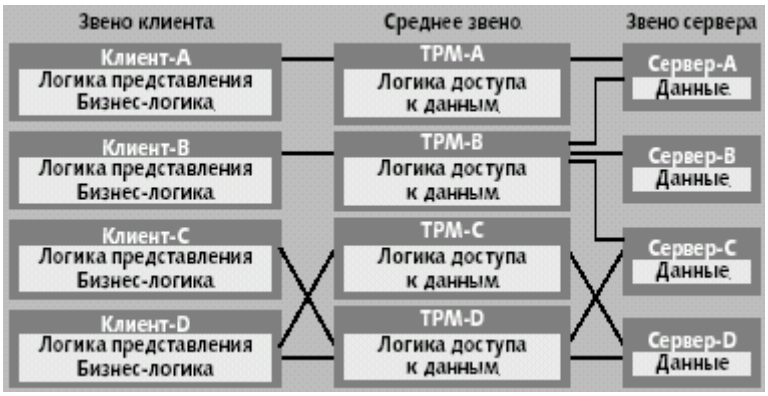


Рис. 8. Трехзвенная архитектура с монитором обработки транзакций

Самая простая конфигурация — клиент-A взаимодействует только с одним монитором TPM-A (рис. 8), который обеспечивает доступ к данным, расположенным на одном компьютере (Сервер Данных A). При помощи двухфазного протокола фиксации монитор может обеспечивать семантику транзакций и с несколькими базами данных. Таким образом, транзакция одного клиента будет фактически разбита между несколькими базами данных; эта ситуация показана в случае с TPM-B, который взаимодействует с источниками данных на нескольких машинах (Сервер Данных A, Сервер Данных B и Сервер Данных C). В этом случае, если подтранзакция терпит неудачу на одном из серверов, все другие подтранзакции также откатываются назад, а Клиент-B получает сообщение о таком событии.

Отношения между клиентами и мониторами, так же как и между мониторами и источниками данных, фактически являются отношениями «многие-ко-многим». Эта конфигурация часто используется, чтобы обеспечить балансировку нагрузки. Когда число пользователей увеличивается, система может запускать дополнительные экземпляры мониторов, обеспечивающих доступ к одним и тем же источникам данных.

В этой конфигурации мониторы могут также обеспечить инфраструктуру для разработки систем, способных обрабатывать ошибки. Например, как видно из рис. 8, Клиент-C и Клиент-D могут обращаться к одним и тем же источникам данных (Сервер Данных C и Сервер Данных D) либо через TPM-C, либо через TPM-D, причем каждый монитор выполняется на своем собственном компьютере. Если один из мониторов выходит из строя, то все клиенты могут переключаться на все еще работающий монитор. Система в этом случае замедлится, но будет способна обеспечить исходный набор функций.

Сервер сообщений

Вместо монитора транзакций приложение может использовать сервер сообщений. Он обладает большинством существенных преимуществ монитора транзакций. Клиенты взаимодействуют с сервером сообщения асинхронно, помещая в очередь сообщения, которые обрабатываются позднее. В свою очередь, сервер сообщений поддерживает пул доступных подключений к базе данных, которые он может использовать многократно. Но вместо предопределенных запросов, с которыми должен работать монитор, само сообщение содержит достаточно информации относительно того, что нужно с ним делать.

У каждого сообщения есть заголовок и тело. Заголовок определяет, что должно быть сделано с сообщением (скажем, выполнить бизнес-правило или хранимую процедуру, послать его в базу данных или передать его другому серверу сообщений). Последнее позволяет осуществлять коммуникации не только между клиентом и базой данных, но и между клиентами или между клиентом и базами данных, доступными через различные серверы сообщений (рис. 9).

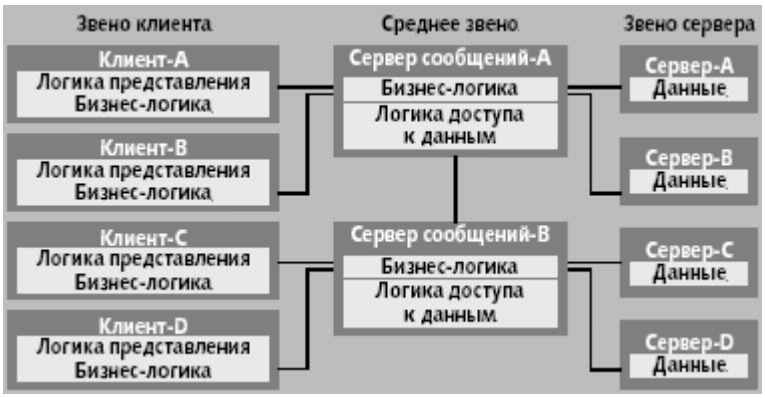


Рис. 9. Трехзвенная архитектура с сервером сообщений

Используя механизмы передачи с промежуточным хранением, сообщения могут обходить точки отказов по альтернативным маршрутам и дополнительным серверам сообщений. Это похоже на использование нескольких мониторов, обслуживающих одних и тех же клиентов и связанных с одними и теми же источниками данных. Но в отличие от мониторов, сообщение может быть пропущено через несколько серверов сообщений прежде чем оно, наконец, достигнет своего адресата. Более того, «интеллект», который направляет сообщение по дополнительному пути, находится не на клиенте, а на сервере сообщений. Таким образом, отказы могут оставаться полностью скрытыми от клиента.
Большинство серверов сообщений поддерживает несколько протоколов связи и выполняется на различных платформах. В результате **прикладная система, построенная с серверами сообщений, может охватывать весьма сложные гетерогенные среды, состоящие из компьютеров разных платформ, различных операционных систем, сетей и протоколов коммуникаций.** Ясно, что помимо создания таких переносимых приложений, серверы сообщений играют очень важную роль в интеграции, позволяя совместно использовать сообщения, данные и бизнес-правила.

Архитектура с брокером объектных запросов

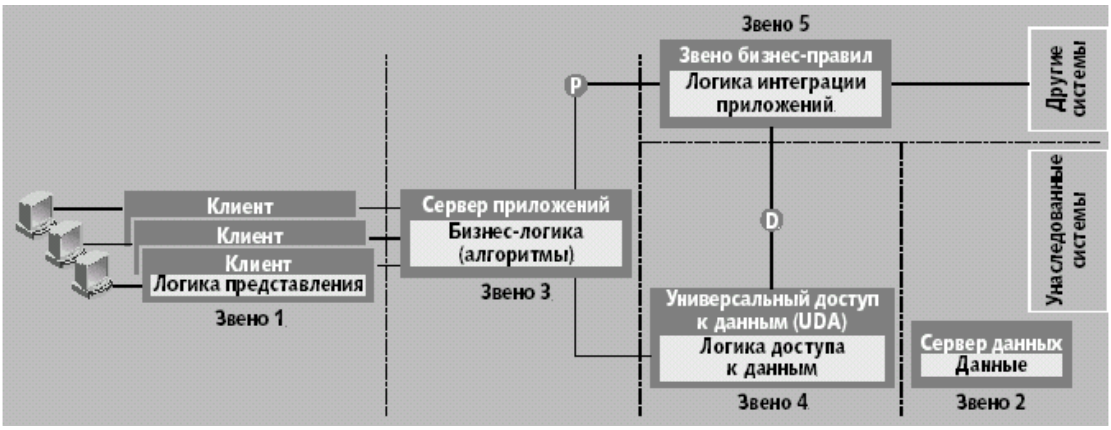
Брокер объектных запросов (object request broker, ORB) обеспечивает инфраструктуру, поддерживающую распределенные объекты, которыми можно управлять как и объектами, расположенными «рядом» с процессом, работающим с ними. По вызову метода (в смысле объектно-ориентированного программирования) на одном компьютере может фактически выполняться некоторый программный код другого, притом что доступ к данным в пределах распределенного объекта может потребовать получения соответствующей информации из удаленной базы данных. Все эти детали остаются скрытыми от прикладной программы [15].
Использование брокера объектных запросов в трехзвенной архитектуре очень похоже на использование сервера сообщений. Единственное различие состоит в том, что взаимодействие осуществляется на объектном уровне, а не на уровне

отдельных сообщений. Эта парадигма, объединенная с мощью объектно-ориентированного программирования, делает распределенные объекты очень привлекательной стратегией для разработки распределенных систем.

Архитектура с пятью звеньями

Нетрудно заметить, что функция среднего звена как части архитектуры клиент-сервер в интегрированной системе должна быть значительно расширена. Во-первых, слой доступа к данным должен управлять не только данными, связанными с «родной» постоянной памятью системы (ее базой данных), но и быть способным обрабатывать информацию из внешних источников, которые представляют данные от других прикладных программ. Кроме того, рассматривая три различных уровня, на которых может проходить интеграция, и соответствующие способы, при помощи которых извлекаются эти данные, можно заключить, что сложность слоя доступа к данным непомерно велика.

Другие факторы, наподобие «упаковки» внешних прикладных программ, могут также требовать разработки локального слоя бизнес-логики. Кроме работы с бизнес-правилами, локальными для текущего приложения, он должен также знать внешние бизнес-правила, которые могут применяться к внешним (и, иногда, к локальным) данным. Вместе с расширенным (продленным) слоем доступа к данным, мы можем теперь получить чересчур «толстое» среднее звено.
Одно из возможных решений состоит в том, чтобы ввести в рассмотрение дополнительные звенья, которые помогли бы ему стать более «тонким». На рис. 11 представлено расширение трехзвенной архитектуры с сервером приложений в среднем звене. Здесь слой доступа к данным управляется звеном Универсального Доступа к данным (universal data access, UDA), показанным как четвертое звено. Он обеспечивает стандартное представление локальных данных, как и данных, постоянно находящихся в удаленных «унаследованных» системах (специализированный механизм доступа к данным унаследованных систем в четвертом звене, иногда называют legacyware — программное обеспечение доступа к унаследованным системам) или в других внешних системах (они доступны через звено бизнес-правил — businessware).



Цель состоит не просто в том, чтобы обеспечить коммуникацию между программами вообще (это задача промежуточного программного обеспечения), а в том, чтобы установить связь между прикладными программами, что включает совместное

использование данных и логики. Передача данных при этом осуществляется через звено UDA (связь D), в то время как совместное использование логики реализуется связью между звеном серверов приложений и звеном бизнес-правил (связь P).

Литература

1. А. Кононов, Е. Кузнецов, Онтология промежуточного ПО. // Открытые системы, 2002, № 3
2. Н. Дубова, Все про промежуточное ПО. // Открытые системы, 1999, № 7-8
3. Ф. Бернштейн, Middleware модель сервисов распределенной системы. // Системы управления базами данных, 1997, № 2
4. Л. Розенкранц, Промежуточное ПО. // Computerworld Россия, 2000, № 39
5. Е. Карташева, Средства интеграции приложений. // Открытые системы, 2000, № 1-2
6. G. Schussel, Client-Server Past, Present and Future. http://news.dci.com/geos/dbsejava.htm, 1995
7. S. Guengerich, G. Schussel, Rightsizing Information Systems, SAMS Publishing, 1994
8. H. Edelstein, Unraveling Client-Server Architecture. DBMS, 1994, № 5: 34 (7)
9. David S. Linthcum, Enterprise Application Integration, Addison-Wesley, 2000
10. R. Orfali et. al., Client-Server Survival Guide, 3rd edition, Wiley Computer Publishing, 1999

A. Berson, Client-Server Architecture, New York: McGraw-Hill, 1992

11. R. M. Adler, Distributed Coordination Models for Client-Server Computing, Computer, 28, 1995, № 4

A. Dickman, Two-Tier Versus Three-Tier Apps. Informationweek 553, 13, 1995

12. J. Gallaugher, S. Ramanathan, Choosing a Client-Server Architecture. A Comparison of Two-Tier and Three-Tier Systems. Information Systems Management Magazine 13, 1996, 2
13. Evans, Lacey, Harvey, Gibbons, Client-Server: A Handbook of Modern Computer Design. Prentice Hall, 1996
14. Martin and Leben, Client-Server Databases: Enterprise Computing, Prentice Hall, 1996

Алексей Кононов (aik762@hotmail.com) — независимый консультант. Евгений Кузнецов (eugene.kuznetsov@hotmail.ru) — сотрудник Института проблем управления РАН (Москва).

3.4.1 Классификация технологий интегрирования информации

Технология ERP весьма сложна и дорогостояща во внедрении, требует значительного времени на настройку. Однако до недавнего времени это был единственный способ интеграции данных и приложений при проведении транзакций. Другими словами, ERP обеспечивает взаимодействие приложений при совершении транзакций, а хранилище данных — интеграцию данных из различных приложений. Обе технологии дополняют друг друга, решая разные проблемы: для оперативной обработки и анализа данных используются хранилища данных, а для интегрированной обработки транзакций — системы ERP.

Развивающиеся системы категорий B2B, B2C и аналогичные им требуют представления полной информации в приемлемое время. Необходимость в коммуникации с внешним миром является первостепенным по значимости требованием для создания интегрированной системы приложений предприятия (enterprise application integration — EAI). Процесс создания EAI — это процесс объединения систем ERP, CRM, SCM, баз данных, хранилищ данных и других внутренних подсистем предприятия.

В консалтинговой компании Hurwitz Group **выделяют шесть уровней традиционной интеграции:**

1. **Интеграция на уровне платформ.** Реализует взаимосвязь между различными аппаратными платформами и операционными системами с помощью таких технологий, как обмен сообщениями, объектные брокеры ORB, удаленный вызов процедур RPC.
2. **Интеграция данных.** Обеспечивается инструментами ETL (extract, transfer, load — - «извлечение, передача, загрузка») и средствами управления метаданными. До недавнего времени существовало два типа решений: доступ к различным базам данных с помощью SQL-запросов и инструменты ETL, извлекающие данные, помещающие их в хранилища напрямую, минуя логику приложений, и предлагающие определенный набор интерфейсов для визуализации.

Интеграция компонентов. Охватывает управление транзакциями, бизнес-логику, работу с бизнес-приложениями, обеспечивая быструю интеграцию унаследованных приложений с новыми компонентами, расширяющими функциональность систем ERP в клиент-серверной архитектуре. В основе этого подхода — серверы приложений, обеспечивающие доступ к различным базам данных, а также интерфейсы к набору стандартных приложений.

Интеграция приложений. Обеспечивает оперативное взаимодействие различных приложений между собой.

Интеграция бизнес-процессов. Дает возможность определять, отслеживать и изменять текущие бизнес-процессы.

Интеграция B2B. (Web-интеграция).Выходит за рамки компании и подразумевает связывание информационных систем заказчиков, поставщиков и партнеров. При этом взаимодействие может быть организовано как в рамках частных сетей, так и по Internet. Web-интеграция может рассматриваться как наиболее современный подход к интеграции данных и приложений.

Можно привести и другие варианты классификации уровней интеграции приложений. Например, в [1] дается следующее представление этой задачи:

Существует несколько основных взаимодополняющих подходов и групп технологий для решения задачи интеграции приложений, которые могут использоваться по отдельности или совместно.

Интеграция пользовательских интерфейсов (User Interface Integration). Это, в конечном счете, не что иное, как замена уникальных для каждой из подсистем графических или текстовых интерфейсов единообразным интерфейсом, **обычно на основе браузеров.** Бизнес-логика остается распределенной по тем же системам ERP, CRM и SCM. Иногда такой подход образно обозначают словом refacing ("изменение лица"); для подобной косметической операции хватает функциональности порталов.
Интеграция данных (Data Integration). Эта наиболее распространенная из существующих на сегодняшний день форма интеграции предполагает объединение баз данных и источников данных. Чаще всего сводится к той или иной форме миграции данных, а бизнес-логика, как и в случае интеграции пользовательских интерфейсов остается распределенной по первичным системам. Существуют некоторые виды программного обеспечения промежуточного слоя, которые обеспечивают процесс репликации данных. Одно из наиболее современных решений, которое называют ETL (Extract, Transform, Load), выделяет, преобразует, фильтрует и сохраняет данные в хранилищах или киосках данных.
Интеграция бизнес-процессов (Business Process Integration). Этот тип интеграции не затрагивает бизнес-логики отдельных процессов, существует специальное промежуточное программное обеспечение, которое согласовывает работу отдельных приложений, используя для этого брокеры сообщений, отвечающие за синхронизацию.
Функциональная интеграция (Function Integration). Функциональная интеграция как наиболее системный подход предполагает непосредственное взаимодействие кросс-платформенных приложений (application-to-application, A2A), работающих в сети. Для этого могут быть использованы просто языковые средства (Cobol, C++, Java), API-интерфейсы, вызовы удаленных процедур (remote procedure call, RPC), распределенное программное обеспечение промежуточного слоя, брокерные архитектуры на базе брокеров (CORBA, DCOM), удаленные вызовы Java (remote method invocation, RMI), промежуточное ПО, ориентированное на сообщения (message oriented middleware, MOM).
Наряду с перечисленными методами для функциональной интеграции могут использоваться Web-службы.
Полная интеграция приложений предполагает использование всех четырех подходов.

Подчеркнем, что Web-службы не следует путать с собственно EAI. **Web-службы — это всего лишь еще одна технология, обеспечивающая создание интегрированной платформы, но в отличие от более традиционных подходов использующая для этой цели слабо связанные приложения.** При этом не стоит принижать возможности действующих средств, предназначенных для интеграции приложений, которые сложились за последние годы: они обладают вполне достаточными возможностями и сохранят применимость еще на годы.

3.4.2 Web-интеграция и ее проблемы

Семь принципов Web-интеграции

Можно отметить наметившиеся следующие принципы Web-интеграции:

1. Интеграция данных и приложений (финансовые, системы CRM, ERP и унаследованные) в корпоративный портал.

2. Визуализация информации при помощи Web-браузера, благодаря чему исчезает необходимость в установке и поддержке «тяжелых» клиентских приложений.

3. Гибкость, масштабируемость и открытость создаваемых решений, которые достигаются благодаря использованию XML/XSL и Java; быстрое и легкое внедрение, оптимизация и персонификация.

4. Единый и настраиваемый интерфейс работы с данными и приложениями, возможный благодаря созданию корпоративного портала; возможность входа в портал из различных систем, в том числе территориально удаленных.

5. Реализация системы на основе технологий Java и XML, что обеспечивает переносимость и независимость приложений от операционных систем сервера и клиента. Хранение и передача данных происходит с помощью языка XML, специально созданного для организации взаимодействия с различными приложениями. Формат данных в XML не зависит от способа его дальнейшей визуализации — за это отвечают хранимые отдельно файлы таблицы стилей XSL, которые преобразуют файлы XML для их представления в виде Web-страницы.

6. Создание в процессе интеграции пользовательских Web-интерфейсов к корпоративным приложениям. При дальнейшем развитии системы обеспечивается не только доступ для чтения данных, но и полноценный документооборот.

7. Безопасность и защита информации обеспечиваются единой точкой входа, каталога и разделением прав доступа. База описаний пользователей также хранится в едином кросс-платформенном каталоге, что создает унифицированную систему безопасности для всех приложений.

Проблемы интеграции

Иногда противопоставляют Web-службы «закрытым» технологиям DCOM, CORBA [4]. Но и CORBA является открытым стандартом, так что это противопоставление не верно. Среди достоинств Web-служб отмечают отсутствие необходимости в сложной инфраструктуре, которая требуется, например, для CORBA. Это способно облегчить задачи интеграции слабосвязанных систем, которые обмениваются, вообще говоря, небольшими объемами данных и не столь интенсивно, чтобы накладные расходы сильно влияли на производительность. Таким образом, сразу стоит ограничить область применения Web-служб только слабосвязанными системами, обменивающимися информацией, не критичной к задержкам. Поставщиками подобных услуг могут выступать различные системы бронирования и продажи услуг, а также системы, торгующие легко идентифицируемым товаром. Кроме того, товаром такого рода может являться и подписка на информационные услуги, например, на ежедневные аналитические отчеты. Потребителями могут выступать информационные порталы или

системы, торгующие сопутствующими товарами. В этих случаях трафик не слишком велик, а информация не носит критического характера.

Однако наиболее принципиальной является проблема согласования семантики данных в различных системах. Если абстрагирование от API-интерфейса является чисто технической задачей, то уход от конкретной семантики сильно затруднен как различием в типах данных (например, в форматах представления чисел с плавающей запятой или денежных единиц), так и их значением. Например, имя заказчика в одной системе может быть представлено в виде сочетания двух полей first-name и last-name, в то время как в другой — подобного разделения по полям нет, а имя и фамилия представлены в виде одного поля name. Возникает необходимость в преобразовании двух полей в одно или, что еще сложнее, наоборот. Некоторые авторы настаивают на построении сложных структур метаданных, нацеленных на решение подобных проблем [5]. Однако появление еще одного промежуточного звена только усложняет интеграцию.

Таким образом, весь выигрыш от использования легковесной технологии Web-служб может быть полностью потерян, ведь во многих случаях более эффективным будет использование CORBA или Sun RPC. Примерами таких ситуаций может служить тесная межсистемная интеграция, предполагающая обмен специфичными данными и требующая доработки интегрируемых систем с целью согласования форматов и семантики передаваемых данных. При этом объем работ и сложность инфраструктуры в случае применения Web-служб резко возрастает, что снижает положительный эффект их «легковесности», но оставляет недостатки, связанные с повышенными накладными расходами вычислительных ресурсов.

Web-портал

Процесс Web-интеграции заключается в создании корпоративного информационного портала путем сведения воедино данных из различных источников. Портал представляет собой окно к структурированным, персонифицированным, корпоративным и другим данным, доступ к которым осуществляется посредством Web-интерфейса. Для обеспечения персонифицированного доступа может быть организована так называемая **«матрица доступа»**, в которой прописываются возможности пользования теми или иными приложениями, базами данных и службами по каждому подразделению компании. В рамках каждого подразделения для отдельных сотрудников могут быть описаны их индивидуальные права.

При внедрении корпоративного портала могут возникать проблемы, обусловленные существующей информационной системой:

- наличие в организации нескольких операционных систем вызывает необходимость проведения кросс-платформной интеграции;
- старые и «унаследованные» базы данных, построенные не на базе архитектуры клиент-сервер, не всегда могут быть интегрированы в портал;
- приложения, прежде всего написанные сторонними разработчиками и небольшими фирмами, могут не иметь Web-интерфейсов и/или документированных API-интерфейсов, что затрудняет их интеграцию.

Могут возникать и другие сложности, затрудняющие или удорожающие проведение Web-интеграции. Тем, не менее, в силу того, что корпоративный информационный портал изначально разрабатывался как кросс-платформенное решение, внедрение которого не зависит от приложений, как правило, трудности эти вполне преодолимы.

Доступ к порталу может быть организован для нескольких групп пользователей.

Руководители с помощью портала получают инструмент контроля за ключевыми параметрами деятельности организации.
Сотрудники получают агрегированную внутреннюю и внешнюю информацию и доступ к корпоративным приложениям. Через корпоративный портал пользователи получают всю необходимую для работы информацию и инструменты. Информация на портале отображается в зависимости от заранее заданного «профиля» пользователя.
Поставщики получают возможность повысить эффективность взаимодействия с компанией, оптимизировать объемы и время поставок. Таким образом, портал становится инструментом для планирования и прогнозирования расходования запасов и поставок. Это становится возможным благодаря оперативному доступу к информации.
Партнеры получают более эффективный инструмент по обмену информацией. Через портал они имеют доступ к маркетинговой и технической документации, планам выпуска продукции и проч.
Заказчики получают информацию о выпускаемой компанией продукции, новые инструменты работы с ней. Анализ действий заказчиков при работе с порталом может помочь компании принимать более взвешенные маркетинговые решения.
Три последние группы являются по отношению к компании внешними, поэтому внедрение корпоративного портала можно рассматривать как построение фундамента для ведения электронного бизнеса.

Для чего нужен корпоративный портал

Внедрение и использование корпоративного информационного портала дает компании целый ряд преимуществ:

- Повышение производительности. Легкий доступ к персонализированной информации из корпоративных и внешних источников повышает производительность труда сотрудников. На базе портала возможна организация документооборота — получение данных из приложений и их ввод, создание и редактирование документов в едином Web-интерфейсе.
- Повышение конкурентоспособности. Поскольку портал представляет собой единую точку входа в корпоративную информационную систему и позволяет получать весь объем необходимой информации, он дает возможность снизить затраты и повысить эффективность работы всех систем организации. Web-интеграция открывает новые возможности анализа деловой информации, сегментирования рынка и позиционирования, планирования и прогнозирования, выполнения ряда иных функций.
- Улучшение корпоративного взаимодействия. Портал играет роль центрального информационного ресурса для сотрудников компании, заказчиков, руководства, поставщиков, дистрибьюторов, партнеров и акционеров. Своевременный обмен необходимой информацией обеспечивает более тесную связь между всеми группами сотрудников и подразделениями. В то время как большинство существующих на рынке систем документооборота позволяет только отслеживать процесс прохождения документов внутри организации, корпоративный портал обеспечивает и ряд других функций: хранение всех версий документа, определение прав доступа ему, оповещение пользователей об изменении существующего документа или появлении нового, публикация документов.
- Повышение отдачи от инвестиций. Портал — это интегрированное приложение, которое можно достаточно быстро внедрить, легко поддерживать, затрачивая при этом сравнительно скромные ресурсы по сравнению с системами со сходными функциями, но построенными на основе других концепций. Все это снижает издержки и повышает отдачу от вложений в корпоративную информационную систему. Использование «тонкого клиента» — браузера — для визуализации информации позволяет, с одной стороны, экономить на обучении персонала, а с

другой, что особенно важно, дает возможность не устанавливать клиентские приложения на множестве компьютеров. Сокращение затрат на приобретение и обслуживание клиентского ПО и оборудования — один из основных ресурсов снижения издержек при использовании корпоративного портала. Важно упомянуть и о минимизации затрат на аренду Internet-канала за счет того, что большая часть информации уже размещена в портале.

Единая платформа для ведения электронного бизнеса. Внедрение корпоративного портала и обеспечение доступа к нему внешних пользователей способствует укреплению деловых связей с заказчиками, партнерами, поставщиками и повышает качество обслуживания заказчиков и партнеров за счет предоставления им дополнительных возможностей и услуг.

Web-служба

Как любое сложное явление, Web-службы не имеют одной точки зарождения; они появились в разных местах и под влиянием разных фактов. Поэтому происхождение Web-служб можно искать в нескольких источниках, например, идя от WWW. Сегодня, говоря о Web-службах, почти всегда имеют в виду только те решения, которые основаны на стеке протоколов SOAP, UDDI и WSDL. Строго говоря, это неверно, поскольку службы, работающие через Web, появились практически одновременно с самой Паутиной и прошли определенный эволюционный путь. Группа авторов из исследовательской лаборатории Hewlett-Packard предлагает деление служб на поколения [3].

Первое поколение: CGI и Perl. С момента своего появления World Wide Web дала возможность просматривать статические страницы средствами браузеров, но, кроме того, с самого начала обеспечивалось пассивное обслуживание. Язык HTML поддерживает механизм FORMS как средство для работы с меню и передачи информации посредством интерфейса Common Gateway Interface небольшим скриптам, написанным большей частью на языках Perl или Shell. Скрипты, в свою очередь, могут воспринимать введенные пользователем данные как запросы и выводить необходимые пользователю страницы, что расширяет возможности работы за рамки навигации.

Второе поколение: Java. С ростом Web стали востребованными такие виды служб, как онлайновые продажи, заказы билетов и т.д. Java-апплеты добавили браузерам возможности для графического интерфейса, благодаря чему Java стал первым настоящим языком программирования для Web-служб. Следующим шагом внутри этого поколения служб стали сервлеты, позволяющие в динамическом режиме генерировать HTML-страницы. Появились близкие по смыслу технологии JSP (Java Server Pages) от Sun Microsystems, ASP (Active Server Pages) от Microsoft, PHP для ОС Linux и др. Они позволили отделить представление страниц от их содержимого, обеспечили простейшую форму аутентификации с использованием имени и пароля пользователя и еще некоторые возможности обслуживания.

Третье поколение: J2EE. Для переноса служб на корпоративный уровень потребовалось развитие языка Java и Java-библиотек. Решение пришло в форме J2EE (Java 2 Platform, Enterprise Edition). Предложенная Sun Microsystems платформа стала развиваться и другими компаниями, в том числе IBM, превратившись в стандарт де-факто для всей отрасли. Появились серверы приложений; прежде же разработчик должен был самостоятельно собирать разрозненные приложения, подключать их к Web-серверу, управлять конфигурацией. Серверы приложений — это, по сути, предварительно собранные пакеты, предназначенные для разделения служб и бизнес-логики и позволяющие разработчику сосредоточить свое внимание на функциональной части.

Наибольшую известность получили построенные на основе J2EE серверы HP AS, IBM WebSphere, BEA WebLogic, Sun iPlanet и Oracle 9i AS.

Четвертое поколение: платформы для Web-служб. Первые три поколения предназначались в основном для пользователей, их возможности ограничивались возможностями HTML. Появление языка XML с его простым синтаксисом и возможностью описывать синтаксис средствами собственной нотации XML Schema стало поворотным моментом и повлекло за собой возникновение служб нового поколения. Сегодня они реализуются на платформах Sun ONE (Open Net Environment) и Microsoft .Net. Принципиальное отличие служб четвертого поколения заключается в том, что они обеспечивают взаимодействие между приложениями. Они никак не ориентированы на человека, поэтому называть их службами не совсем верно.

Уже сейчас просматриваются черты пятого поколения, где будет использоваться динамическая сборка Web-служб, агентские технологии и т.д.

Появление четвертого поколения Web-служб можно воспринимать еще и как логический итог развития распределенных вычислений. На корпоративном уровне потребность в распределенных вычислительных системах возникла давно, вслед за тем, как в качестве альтернативы мэйнфреймам стали использовать сетевые конфигурации, объединяющие рабочие станции и мини-ЭВМ. Одной из первых возникла задача обеспечения электронного обмена данными (electronic data interchange, EDI). Уже тогда были сделаны первые шаги по направлению к нынешним Web-службам. Многие технические решения, датируемые началом 80-х годов, остаются актуальными и сегодня. В свое время они были воплощены в операционных системах DEC VMS, в различных версиях Unix, поставлявшихся компаниями Tandem, Hewlett-Packard и Sun, а IBM со своим продуктом MQSeries тогда от этой группы приотстала.

Если до сих пор Web-службы играли роль консолидатора гетерогенных систем сильно децентрализованных предприятий, то на следующей стадии они станут неким подобием супермаркета по продаже информационных услуг. Web-службы будут строиться в этом случае из программных компонентов, которые должны быть идентифицированы, скомпонованы и динамически собраны в единое приложение. Правда, на сегодняшний день это еще невозможно. Отсутствуют четкие спецификации создания программных компонентов, позволяющие «положить» их на полку супермаркета; бизнес-пользователям придется изменить свои представления о ценности и открытости интеллектуального потенциала, применяемого в компонентах; необходимо еще создать общепризнанные стандарты по семантике безопасных строительных блоков для вновь создаваемых приложений.

Подводные камни Web-служб

Технологиям Web-служб уделяется сегодня особое внимание. Многие игроки ИТ-рынка заявили о поддержке этой технологии и предлагают свои реализации, часто преподнося их как панацею от всех «бед» интеграции приложений. Однако мало кто отмечает проблемы новой технологии.

Основное назначение Web-служб — обеспечение совместного доступа к распределенным компонентам. Этого можно достичь и простым выполнением вызова процедуры на удаленной машине; **особенность современной технологии Web-служб состоит в том, что она предполагает интеграцию путем обмена XML-сообщениями при помощи протокола SOAP. При этом интерфейсы определяются с помощью языка WSDL (Web Services Description Language), а UDDI (Universal Description, Discovery and Integration) предоставляет службу динамического каталога для компонентов, предоставляющих свои услуги, и для компонентов, которые эти услуги используют.**

Основные поставщики серверов приложений, серверов баз данных и промежуточного программного обеспечения уже заявили о своей поддержке технологии Web-служб. Более того, некоторые уже предлагают ее реализацию, зачастую преподнося ее как панацею от всех бед, способную решить накопившийся ком интеграционных проблем. Между тем, необходимо более критическое осмысление многообещающей технологии.

3.4.3 Интеграция АСУП и АСУТП

Для решения актуальных сегодня проблем интеграции АСУП и АСУТП рассматриваются возможные пути, использующие уже имеющееся программное обеспечение. Основой объединения информационных потоков служат базы данных, широко используемые в обеих системах, специальные программные продукты, имеющие целью объединять различные подсистемы, работая с разнородными протоколами в конкретных подсистемах АСУП и АСУТП. И, наконец, недавно было заявлено о появлении готовых интегрированных решений, ориентированных на автоматизацию предприятия как единого целого.
Еще до недавнего времени две подсистемы автоматизации промышленных предприятий: АСУП (системы автоматизации управленческой и финансово-хозяйственной деятельностью, планирования ресурсов предприятия) и АСУТП (системы автоматизации технологических и производственных процессов) развивались обособленно и независимо друг от друга (рис. 1).

Рис. 1. Основные подсистемы автоматизации на предприятии

Исторически сложилось так, что каналы обмена (особенно оперативные) между подсистемами оказались достаточно слабыми. Возможно, так и продолжалось бы дальше, но необходимость в АСУП технологических данных, в том числе и оперативных, стала очевидной. Несмотря на то, что до сих пор управленческие решения строятся главным образом на интуиции и опыте, что, конечно, крайне важно, заметное присутствие субъективного фактора на процессе принятия решения не гарантирует взвешенного, проверенного решения. Сегодня практически все службы предприятия заинтересованы в получении объективных технологических данных.

Технологические данные как исходная информация позволяют принять качественные стратегические управленческие решения многих задач:

- повышение качества продукта;
- повышение объемов производства;
- повышение эффективности производства;
- снижение длительности простоев;
- снижение себестоимости;
- сохранение инвестиций.

Объем и степень доступа к технологической информации зависят от типа программного обеспечения, используемого в управленческих структурах предприятия и от категории сотрудников-потребителей данной информации. Ниже указаны цели и типовые вопросы различных служб предприятия, которым все это нужно:

- управление производством;
- службы контроля;
- технологические службы;

- ремонтные службы;
- службы контроля качества;
- операторы оборудования;
- плановые службы;
- бухгалтерия;
- прочие.

Ситуация усугубляется еще и тем, что программное обеспечение, используемое в АСУП и АСУТП, достаточно долго развивалось независимо и не предусматривалась возможность стандартизации каналов обмена между двумя системами. Поэтому до детального обсуждения вопросов интеграции двух подсистем отметим общие свойства и различия в организации программного обеспечения для них.

Сходство и различие систем

Рассматриваемые подсистемы являются распределенными, поэтому протоколы локальных сетей и протоколы Internet позволяют интегрировать информационные и управляющие потоки в узлах каждой подсистемы. Объединение узлов возможно как в режиме n- tier (равные с равными), так и клиент – сервер. Кроме этого, на сегодняшний день определились категории программных средств, используемых в подсистемах АСУП и АСУТП, причем каждая категория в АСУП зависит от степени интеграции систем.

В рамках каждой категории можно обсуждать устоявшиеся протоколы обмена между компонентами. Основным типом программного обеспечения, используемым для автоматизации технологических процессов, являются SCADA-системы, решающие следующие основные задачи:

- визуализация технологического процесса;
- сбор данных от различных источников информации по DDE (Dynamic Data Exchange), OPC (OLE for Process Control) и частным протоколам;
- поддержка языка SQL для создания, удаления, чтения, записи и модификации информации в таблицы баз данных.

Теперь о различиях. Отношение к реальному времени в подсистемах АСУТП принципиально важно – негарантированное время реакции на событие в технологическом процессе недопустимо. Различные каналы обмена (а соответственно, и протоколы) характеризуются соответствующими приоритетами, и определяются степенью критичности выполняемых задач.

Из всего этого следует объективная необходимость интеграции – сегодня созданы для этого необходимые предпосылки. Рассмотрим теперь возможные пути интеграции подсистем уровней АСУП и АСУТП:

- использование баз данных в качестве буфера для обмена и бизнес- приложения для организации передачи данных между двумя подсистемами. Причем базы данных могут быть как основой функционирования самих подсистем, так и средством, используемым для хранения функциональных данных. Именно базы данных, скорее всего, могут стать основным средством интеграции двух подсистем;
- применение класса продуктов, импортирующих и экспортирующих объекты из одной подсистемы в другую (например, ПО VisualFlow);
- использование готовых решений, образуемых при объединении компаний-разработчиков продуктов АСУП и АСУТП (например, Wonderware и Marcom).

Базы данных
Важный компонент обоих типов систем – это СУБД. Именно они позволяют предоставить пользователю нужную информацию в нужном месте и в нужное время. Сегодня предприятия с помощью СУБД преодолели проблемы, связанные с дублированием информации и исключением в ней противоречий, однако использование традиционных реляционных баз данных, ориентированных на АСУП-решения, не всегда возможно в системах АСУТП.

Производственные процессы генерируют данные очень быстро. Чтобы хранить производственный архив системы, например, с 7500 рабочими переменными, в базу данных каждую секунду необходимо включать 7500 строк. Обычные базы данных не могут выдержать подобную нагрузку.
Производственная информация имеет большой объем. Многомесячный архив завода с 7500 рабочими переменными требует под базу данных около 1 Тбайт дисковой памяти. Сегодняшние технологии такими объемами манипулировать не могут.
SQL как язык не подходит для обработки временных или периодических данных, типичных для производственных систем. В частности, чрезвычайно трудно указать в запросе периодичность выборки возвращаемых данных.
Для преодоления этих ограничений был предложен новый класс продуктов – базы данных реального времени (БДРВ), созданные независимо, либо разработанные на основе существующих реляционных СУБД. Более перспективным представляется второй подход, поскольку, во-первых, в стоимостном отношении он дешевле, во-вторых, технологичнее. В качестве примера реализации базы данных реального времени отметим, например, IndustrialSQL Server (компания Wonderware) и Plant2SQL (Ci Technologies). Основные функции баз данных реального времени, построенной на основе Microsoft SQL Server, заключаются в следующем.

1. Сохранение некритичной ко времени информации в Microsoft SQL Server. В то время как вся технологическая информация сохраняется в специальном формате.

2. Поддержка высокой пропускной способности для обеспечения сохранения огромных потоков информации с высокой разрешающей способностью.

3. Поддержка целостности данных для обеспечения записи больших объемов информации без потерь.

4. Добавление в Microsoft SQL Server свойств сервера реального времени.

Сегодня базы данных реального времени ориентированы на хранение технологической информации, обеспечение связи с управленческими данными, на использование уже ставших стандартными в подсистемах АСУП компонентов OLE DB, а также Internet-интерфейсов (рис. 2). С одной стороны, им приходится иметь дело с данными, которые поступают в базы данных реального времени из различных технологических источников, с другой – с данными, запрашиваемыми потребителями через интерфейс SQL-сервера.

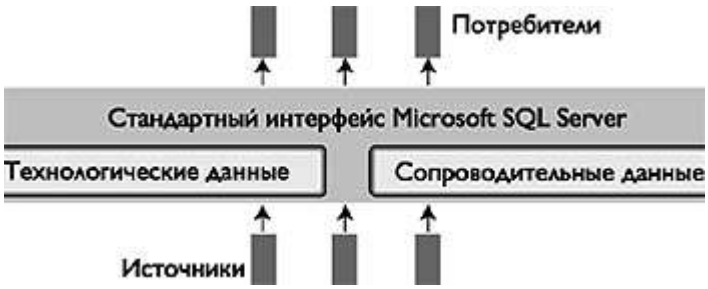


Рис. 2. База данных реального времени на основе Microsoft SQL Server

Стандартным механизмом поиска информации в серверах баз данных реального времени является SQL, что гарантирует доступность данных самому широкому кругу приложений. В подмножество языка SQL входят расширения, служащие для получения динамических производственных данных и позволяющие строить запросы на базе временных отметок.

Используемая в базах данных реального времени архитектура клиент-сервер позволяет заполнить промежуток между промышленными системами контроля и управления реального времени, характеризующимися большими объемами информации и открытыми гибкими управленческими информационными системами. Благодаря наличию мощного и гибкого процессора запросов пользователи имеют возможность осуществлять поиск любой степени сложности для выявления зависимостей и связей между физическими характеристиками, оперативными условиями и технологическими событиями.

Следует подчеркнуть, что в зависимости от требований создаваемой системы возможны следующие варианты решений:

использование только реляционных баз данных, в таблицы которых подсистема АСУТП по SQL-запросам записывает технологические данные, которые в дальнейшем могут быть использованы обеими подсистемами;
использование баз данных реального времени, которые обеспечивают более высокие характеристики регистрации данных и упрощают (без использования SQL) процесс внесения данных в таблицы;
построение комбинированного решения, предполагающего использование баз данных реального времени для технологических первичных данных и таблиц реляционных баз для вторичных.
Специальные технологии
Как уже было отмечено, существует устоявшийся набор стандартных протоколов, позволяющий назвать практически любую SCADA-систему открытой. Большинство SCADA-систем «живет» сегодня на платформе Microsoft Windows и поддерживая соответствующие технологии межпрограммного взаимодействия внутри подсистем АСУТП.

ПО подсистем АСУП, разработанное для платформы Windows, не могло избежать влияния технологий Microsoft – построенные на их основе каналы связи позволяют обеспечить обмен, а стандартные программные протоколы DDE, OLE и OPC могут стать основой интеграции подсистем (рис. 3).

Рис. 3. Обобщенная схема взаимодействия двух типов служб

Новый класс продуктов
Для организации информационного потока технологических данных в системы АСУП ряд крупных разработчиков инструментальных систем (прежде всего, SCADA) предложили использовать специальный тип программных продуктов, например, VisualFlow компании Envisionlt.

Рис. 4. Возможности интеграции VisualFlow

Ключевое назначение VisualFlow – объединять (рис. 4). Графический объектно-ориентированный инструментарий позволяет через объекты и промежуточные мосты организовывать каналы связи с приложениями, которые могут формировать специальные объекты, передаваемые в VisualFlow где с помощью таблиц и методов на объекте, переданном из источника, выполняются необходимые преобразования. Далее объект передается целевому приложению. Сейчас VisualFlow в состоянии обеспечить интерфейс для 120 различных приложений и баз данных.

Так, для интеграции с системами SAP R/3 определены в VisualFlow следующие интерфейсы:

- интерфейс PPPI (PI-PCS BAPI);
- интерфейсы, сертифицированные компанией SAP (R/3 RFC, Access 10,000+RFC, CALL_TRANSACTION RFC, IDOC, ABAP/4);
- интерфейс баз данных (доступ к таблице/полю и поддержка всех баз данных R/3).

Однако VisualFlow — это только продукт, способный интегрировать, а что интегрировать определяется конкретной задачей. Разработчик приложения VisualFlow определяет из каких подсистем принимаются объекты (например, из SCADA-приложений), каким образом полученная через объекты информация преобразуется и передается в целевые подсистемы (например, АСУП). Формирование SCADA-объектов или объектов баз данных реального времени позволяет на базе VisualFlow транспортировать объект с технологическими данными управленцам.

Заключение
Рассмотренные способы интеграции позволяют объединить бизнес-информацию с технологической, просчитывать стоимость инвестиций и материальных издержек и открывают новые возможности для построения интегрированных систем управления. Для управления инфраструктурой предприятия некоторые компании уже сейчас предлагают комплексные решения, представляющие собой набор базовых модулей платформы управления. Свойство некоторых модулей состоит в совершенствовании возможностей управления ресурсами корпоративной системы в нормальных и критических ситуациях:

- бизнес-срез предприятия с предоставлением карты услуг с распределением информационных служб по инфраструктуре и мониторингом работы приложений;
- накопление опыта технического персонала по управлению конкретным ресурсом в конкретной ситуации;
- возможность адаптирующего управления при возникновении проблем с работоспособностью системы в целом.

Таким образом, традиционные системы АСУП имеют сегодня тенденцию превращаться из систем управления сетевыми и системными ресурсами в интеллектуальную платформу управления предприятием. И в этих условиях объективная информация, поступающая с технологического уровня, позволит принимать более качественные управленческие решения.

Литература
1.Н Куцевич. Интеграция АСУП и АСУТП. М:, ж."Открытые системы", №9,2000

Управление производством
Цель: Обеспечить выполнение производственного плана в соответствии с запланированным объемом производства, затратами и качеством продукции.

Типичные вопросы:
- Соответствует ли выпуск плановым показателям? Где возникают узкие места?
- Что является причиной задержек?
- Соответствует ли реальная себестоимость расчетной?
- Каковы отклонения производственных показателей?

Службы контроля
Цель: Оптимизация процесса, соблюдение правил техники безопасности.

Типичные вопросы:
- Стабилен ли данный контур управления?
- Почему сработал данный предохранительный механизм?

- Каким образом вчерашние изменения повлияли на сегодняшнюю производительность?
- Почему стан № 5 не запускается в автоматическом режиме?

Технологические службы
Цели: Повышение эффективности процесса, поддержание работоспособности оборудования.

Типичные вопросы:
- Почему перегревается данный насос?
- Что явилось причиной подъем температуры в регуляторе давления?
- Каковы параметры процесса при высоком или низком выходе продукции?
- Какова связь между производственными характеристиками оборудования и стабильностью?

Плановые службы
Цели: Разработка графиков выпуска продукции.

Типичные вопросы:
- Как соотносится текущий коэффициент загрузки оборудования со средним?
- Соблюдается ли график производства?
- Каковы минимальные и максимальные показатели выпуска продукции в час?
- Каков, по сравнению с расчетным, фактический объем сегодняшнего выпуска?

Бухгалтерия
Цели: Контроль и минимизация затрат производства.

Типичные вопросы:
- Есть ли доход от выпуска данного вида продукции?
- Каков текущий уровень потребления сырья и материалов по сравнению с предыдущим месяцем?
- Какова структура себестоимости продукции в этом месяце?

3.4 Уровни интеграции в современных корпоративных сетях

«Традиционная» интеграция приложений и данных vs. Web-интеграция

В 60-е годы число приложений было невелико, каждое из них решало, как правило, одну узкую задачу. Использование вычислительной техники сводилось, в основном, к автоматизации рутинных операций. На этом этапе потребности в интеграции данных и приложений практически не существовало.

С начала 70-х стали появляться более развитые приложения: программы учета складских запасов, управление производством и финансами. Это были более сложные и многофункциональные, по сравнению с первыми примитивными приложениями, системы.

К середине 70-х стали появляться базы данных; при поиске информации уже не требовалось просматривать множество файлов, достаточно было обратиться к единому хранилищу. Появился новый тип приложений, служивших для управления поиском и доступом к данным. Компьютер стал выступать в новой роли инструмента для проведения транзакций. К сожалению, приложения разрабатывались только с учетом текущих потребностей, в них не закладывалась возможность развития. Очень немногие из них могли взаимодействовать с другими и обращаться к различным базам данных.

В начале 80-х компании столкнулись с серьезной проблемой роста затрат на поддержку корпоративных приложений. По некоторым данным, ИТ-менеджеры 95% времени тратили на поддержку существующей информационной инфраструктуры и только 5% — на развитие и создание новых приложений. Делались попытки расширить функциональные возможности и обеспечить взаимодействие различных систем. К сожалению, добиться этого было сложно из-за необходимости обеспечивать непрерывность работы компании.

Уже к середине 80-х возникла острая необходимость модернизации имевшейся у компаний информационной инфраструктуры; стали разрабатываться новые подходы построения корпоративных информационных систем. В этот период была выдвинута гипотеза о том, что единый рубрикатор (словарь данных) решит проблему интеграции данных. Затем появились теории репозитария данных, являющиеся развитием предыдущего подхода применительно к архитектуре клиент-сервер. Возникла также теория «моделирования данных», согласно которой для интеграции необходимо и достаточно построить правильную модель корпоративных данных. К сожалению, для реализации этих моделей необходимо проведение серьезной реорганизации, переписывания значительной части корпоративных приложений. Однако работающие приложения не поддаются значительным изменениям без того, чтобы это не привело к необходимости переписывать и другие связанные с ними программы. Это очень трудоемкий, длительный и дорогостоящий процесс, который, к тому же, может вызвать нарушения в работе организации.

В начале 90-х появились два новых подхода к интеграции существующих данных и приложений: технология хранилищ данных и системы планирования ресурсов предприятий (ERP — enterprise resource planning). Каждый из них решал проблему интеграции, но лишь частично. Технология хранилищ данных была ориентирована на интеграцию данных, их анализ, не обеспечивая при этом интеграции приложений при проведении транзакций. Очевидное преимущество хранилищ данных в том, что они могут быть построены «вокруг» унаследованных приложений. Иными словами, эта технология не требует переписывания старых приложений, а позволяет пользоваться данными, полученными уже существующими программами. Еще одно преимущество хранилищ данных — поэтапность их построения, повышающая отдачу от их внедрения. Следующим шагом после хранилищ данных стало создание механизмов обработки и анализа корпоративной информации. При этом работа может

вестись как со структурированными данными (например, производственные и финансовые показатели деятельности компании), так и с неструктурированными (ленты информационных агентств, организационные регламенты и т.д.).

Появившаяся на рубеже тысячелетий новый технология, которую называют Web-интеграция привела к новому витку в задачах интеграции информации, т.к. позволяет легко описывать и размещать в компьютерной среде новую разнородную информацию. Что же такое Web-интеграция?

Данный термин появился на Западе пару лет назад; по всей видимости, первой его использовала корпорация IBM. Сейчас в него вкладывается двоякий смысл. Во-первых, это возможность «выхода» за рамки одной организации в процессе интеграции, интеграция нескольких компаний, интеграция B2B. При этом противопоставляется интеграция внутри компании («традиционная» интеграция) и интеграция между компаниями (Web-интеграция). Во-вторых, термин используется для обозначения использования Web-технологий при построении информационных систем предприятия.

Мы будем понимать под Web-интеграцией определенную форму, методы обработки и представления внутренних и внешних информационных ресурсов организации при помощи Web-технологий [18]. Web-интеграция дает возможность более эффективно использовать информационную систему компании, повысить ее управляемость и снизить издержки, контролировать ее внутренние ресурсы, упростить взаимодействие между ее структурными подразделениями. Суть Web-интеграции в том, что с помощью Web-технологий пользователь получает доступ к разным корпоративным приложениям и ко всей информации организации, хранящейся в разных базах данных, а также к данным из других источников. Применение Web-технологий позволяет в короткие сроки и при относительно низкой стоимости повысить отдачу от использования имеющихся у организации корпоративных приложений: бухгалтерских и учетных программ, баз данных, систем управления отношениями с заказчиками (CRM — customer relationship management) и пр.

3.5. Web- клиент/серверные технологии

Число Web-серверов в мире возрастает очень стремительно, и запросы, адресуемые к ним, сохраняют такой же нарастающий темп. Первоначально, для Web-сервера было вполне достаточно возвратить простую Web-страницу многим окнам просмотра пользователей Web-браузеров, путешествующим в Internet и извлекающим для себя полезную информацию в виде текстов и графики. Затем, после того как Web-интерфейс стал основным средством доступа в сети, люди захотели выполнять все более сложные транзакции для доступа к самым разнообразным данным - иметь доступ к данным в реальном времени, о покупках и продаже товаров и т.д. Т.е. появилась функция интегрирования распределенной в сети информации.

Средства Web, помимо связывания распределенных данных, осуществляют еще одну очень важную функцию. Они позволяют рассматривать информацию с нужной степенью детализации, что существенно упрощает анализ больших объемов данных. Можно включать в просмотр только самое интересное, а все остальное маркировать специальными гипертекстовыми иконками или специальным образом выделенными словами, удар по которым мышкой пользователь персонального компьютера может получить новую порцию детализированной информации, которую можно изучить во всех подробностях.

Таким образом, Web-серверы и Web-навигаторы могут и должны использоваться не только в «мировом масштабе». Web – это сервис, необходимый каждой организации со сколько-нибудь заметными информационными потоками. Поэтому он сразу после своего возникновения стал использоваться не только в Internet, для которого первоначально был создан, но и в рамках небольших локальных сетей, работающих по протоколу TCP/IP, т.е. в режиме своеобразного внутреннего Internet, или, как его еще стали называть, Intranet.

В то же время Web-сервису присущи и определенные недостатки, вытекающие из отсутствия объектной ориентации и из природы HTTP-протокола.

Во-первых, клиент по существу лишен средств управления внешним представлением объектов на просматриваемой WWW-странице. Он получает страницу с маркированными ссылками в том виде, в каком ее создал публикатор.

Во-вторых, Web-страницы статичны. При использования протокола HTTP на клиентскую систему передаются только пассивные данные, полученные из различных объектов, но не сами объекты и образующие их методы.

В-третьих, Web-сервис обладает весьма ограниченными интерактивными возможностями, которые сводятся к заполнению пользователем чисто текстовых форм с последующей отправкой на сервер. Сервер анализирует полученные данные, после

чего формирует и возвращает клиенту новую WWW-страницу, которая нередко вновь оказывается формой. Такой стиль общения не всегда устраивает пользователей.

Java-технология позволяет устранить все отмеченные недостатки. В результате Web-сервис, и без того имевший огромную популярность в мире, получили новый импульс. Миллионы пользователей уже привыкших в Internet работать с Web-страничками, где информация организована по принципу раскрывающихся гнезд на домашней страничке, с появлением Java просто получили крылья. Теперь стало возможно творить все, не выходя из Web-браузера! Можно стало запускать приложения, находящиеся на сервере, и получать результаты прямо в HTML-страничку. Более того, не нужно стало заботится о клиентской части, о распределенной среде, о доступе к различным базам данных. Возможности открылись поистине безграничные.

Далее мы рассмотрим основные компоненты и направления развития Web-серверных технологий.

3.5.1. Архитетура Web-сервера с "браузером".

Достоинства этой архитектуры (рис 3.5.1) сводятся к достоинствам соответствующих частей системы клиент-сервер.

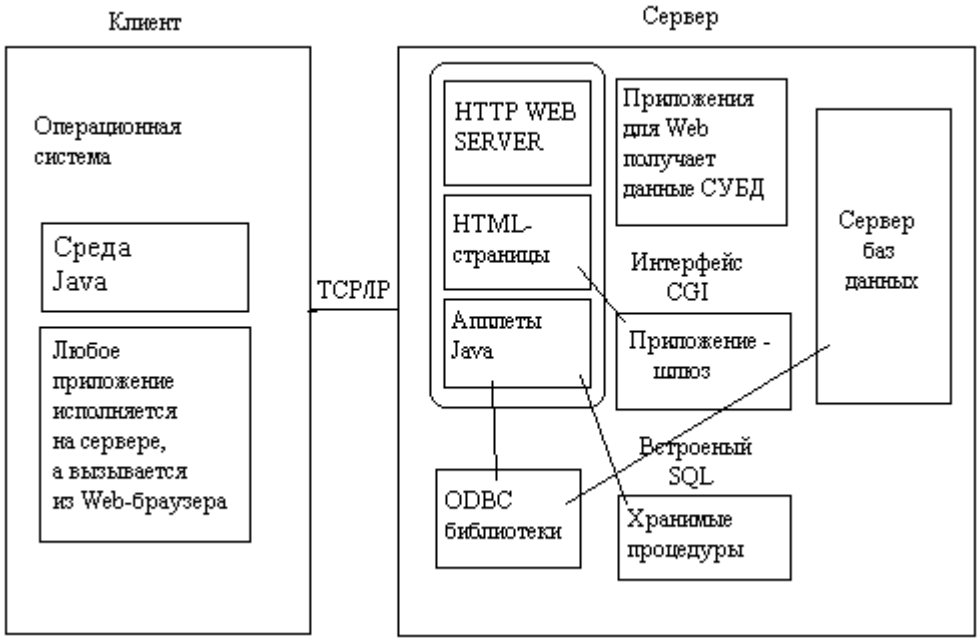


Рис. 3.5.1. Универсальная схема клиент-сервер для сетей Internet/Intranet

Клиентская часть.

Прикладная программа доступна с любого компьютера, на котором инсталлирован стандартный браузер. Пользователю нет необходимости изучать интерфейс прикладной программы, потому что он всегда преобразуется к стандарту HTML-странички. Это помогает снизить затраты на обучение. Кроме того пользователя совершенно не заботят особенности хардверной платформы и операционной системы, поскольку он имеет дело только с браузером, который умеет делать все.

Серверная часть.

Приложения доступны любому пользователю сети Internet/Intranet, имеющего право обращаться к ним. Поскольку все операции по сопровождению и усовершенствованию системы происходят на сервере, то отпадает необходимость сопровождать и модернизировать части приложений, находящихся на машинах-клиентах. Такая конфигурация способна обеспечить работу десятка тысяч или даже миллионов пользователей, являясь идеальной архитектурой для унаследованных программ.

Обратите внимание на то, что на рис 3.5.1 приложение на сервере выполняется в контексте Web, а это может быть любой сервер в Internet, который поддерживает протокол HTTP.

Что касается корпоративной сети, то Web-компоненты приложений серверной части могут обращаться к серверу БД за данными несколькими путями:

- Используя приложение-шлюз через CGI (Common Gateway Interface);
- Используя Java апплеты через хранимые процедуры встроенного SQL;
- Используя Java апплеты через ODBC- драйверы *.dll библиотек.

Причем, нужно помнить, что CGI жестко привязан к страницам Web, а другие способы более гибкие.

3.5.2 Брокеры запросов. Их функции и особенности.

Необходимая спецификация доступа к распределенным приложениям в гетерогенной среде была разработана OMG и получила название Object Management Architecture (OMA).

OMA состоит из четырех основных компонент, представляющих собой спецификации различных уровней поддержки приложений (рис. 5.2):

- архитектуры брокера запросов объектов (CORBA – Common Object Request Broker Architecture) – устанавливает базовые механизмы взаимодействия объектов в гетерогенной зоне;
- сервисов объектов (Object services) – являются основными системными службами, используемыми разработчиками для создания приложений;
- универсальных средств (Common Facilities) – ориентированы на поддержку пользовательских приложений, таких, как электронная почта, средства печати и т.д.;
- объектов приложений (Application Objects) – предназначены для решения конкретных прикладных задач.

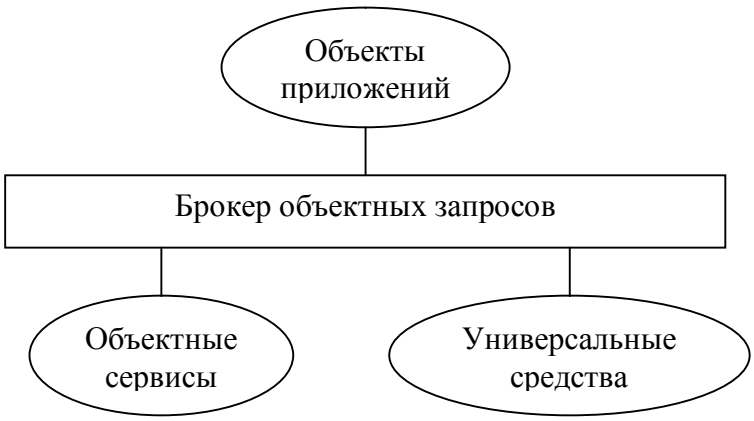


Рис. 5.2. Архитектура управления объектами (OMA)

COBRA определяет механизм, обеспечивающий взаимодействие приложений в распределенной среде.

Главными компонентами стандарта CORBA являются:

- объектный брокер запросов (Object Request Broker);
- язык определения интерфейсов (Interface Definition Language);
- объектный адаптер (Object Adapter);
- репозитарий интерфейсов (Interface Repository).

Место CORBA в семиуровневой модели OSI

Наличие большого количества сетевых протоколов в разных операционных системах потребовало разработки спецификации сетевых соединений. Семиуровневая модель Open System Interconnection (OSI) обеспечивает описание сервисов, которые каждый уровень должен представлять для реализации соединения. Низший уровень – физический – определяет доступ к физической линии. Уровень данных обеспечивает достоверную передачу данных по физической линии. Сетевой уровень имеет дело с установкой соединения и маршрутизацией. Транспортный уровень отвечает за достоверную передачу до точки назначения. Уровень сессий обеспечивает управление соединением. Уровень представлений описывает синтаксис данных и обеспечивает прозрачность для приложений. Последний уровень – уровень приложений – обеспечивает соответствующие сетевые функции для конечного пользователя.

Концептуально CORBA относится к уровням приложений и представлений. Она обеспечивает возможность построения распределенных систем и приложений на самом высоком уровне абстракции в рамках стандарта OSI. С ее помощью возможно изолировать клиентские программы от низкоуровневых, гетерогенных характеристик информационных систем. Характерные особенности разработки по технологии CORBA заключаются в следующем.

Язык описания интерфейсов OMG IDL позволяет определить интерфейс, независимый от языка программирования, используемого для реализации.

Высокий уровень абстракции CORBA в семиуровневой модели OSI позволяет программисту не работать с низкоуровневыми протоколами.

Программисту не требуется информация о реальном месте расположения сервера и способе его активизации.

Разработка клиентской программы не зависит от серверной операционной системы и аппаратной платформы.

3.5.3 Брокерная архитектура CORBA

Объектная модель CORBA

Объектная модель CORBA определяет взаимодействие между клиентами и серверами. Клиенты – это приложения, которые запрашивают сервисы. Серверы – это приложения, представляющие сервисы.

Объекты-серверы содержат набор сервисов, разделяемых между многими клиентами. Операция указывает запрашиваемый сервис объекта-сервера. Интерфейсы

объектов есть описание множества операций, которые могут быть вызваны клиентами данного объекта. Реализации объектов – это приложения, реально исполняющие сервисы, запрашиваемые клиентами.

Объектный брокер запросов (ORB)

Спецификация CORBA разработана для обеспечения возможности интеграции совершенно различных объектных систем.

Его задачей является представление механизма выполнения запроса, сделанного клиентом: поиск объекта, к которому относится данных запрос, передача необходимых данных, подготовка объекта к обработке. Интерфейс, с помощью которого клиент может запрашивать выполнение необходимых операций, не зависит от местонахождения объекта и языка программирования, с помощью которого он реализован. Клиент может запрашивать выполнение операций с помощью ORB несколькими способами. На рис. 5.3 показаны способы возможного взаимодействия объектов.

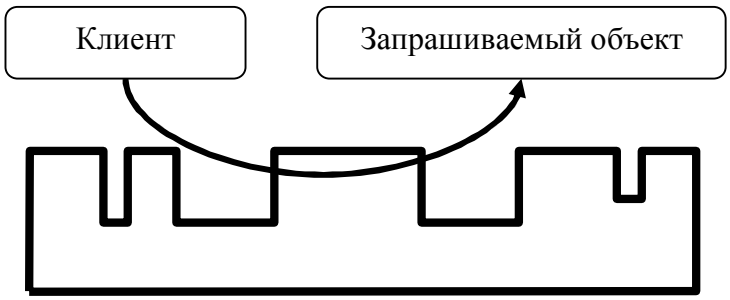


Рис. 5.3. Схема работы объектного брокера (ORB)

Структура ORB

Рис. 5.3. показывает запрос, посылаемый пользователем к системе объектной реализации. Клиент – это пользователь, который желает выполнить операцию над объектом в системе объектной реализации.

Система объектной реализации – это фактически код и данные, которые физически содержатся в объекте. ORB ответствен за все механизмы, требуемые, чтобы найти объектную реализацию для запроса, подготовить ее, реализовать объект и содержащиеся в нем метод и отправить данные, составляющие запрос, пользователю. Интерфейс, с которым имеет дело пользователь, полностью независим от того, где объект размещен, какой язык программирования в нем используется, где это

выполнено, а также в отношении любого другого аспекта, который совершенно не влияет на интерфейс объекта.

Рис. 5.4 показывает структуру индивидуального ORB. Интерфейсы к ORB показываются закрашенными прямоугольниками, и стрелки указывают, вызывается ли ORB или выполняет вызов-запрос при помощи интерфейса.

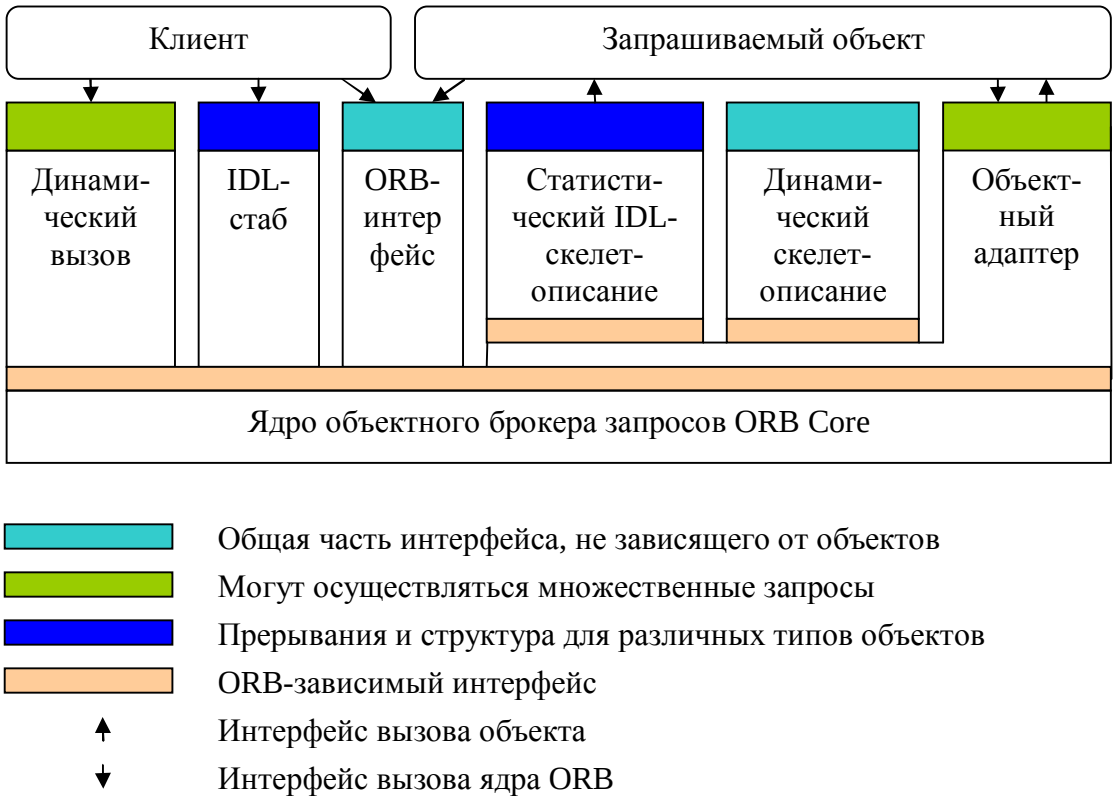


Рис. 5.4. Структура интерфейса объектного брокера запросов

Вызов операций разделяемого объекта-сервера может быть сделан статическим (IDL-стаб) и динамическим (Dynamic Invocation Interface) способом. Интерфейсы описываются, используя язык определения интерфейсов, получивший название OMG Interface Definition Language (IDL). В случае статического вызова эти описания интерфейсов отображаются в программный код на языках C, C++, Smalltalk. Информация об интерфейсах объектов может быть получена клиентом двумя способами: статически (compile time) и динамически (runtime). Интерфейсы могут быть также указаны с помощью службы репозитория интерфейсов (Interface Repository). Этот сервис представляет интерфейсы как объекты, обеспечивая доступ к ним во время работы приложения (в runtime-режиме). (Описание динамического вызова интерфейсов и репозитария интерфейсов см. ниже.)

Пользователь выполняет запрос при наличии доступа к объекту, когда знает и тип объекта и желаемую функцию выполнения. Пользователь инициализирует запрос, вызывая подпрограммы статической настройки, которые являются специфическими для объекта, или строит динамически оформляемый запрос.

Объектный адаптер (Object Adapter)

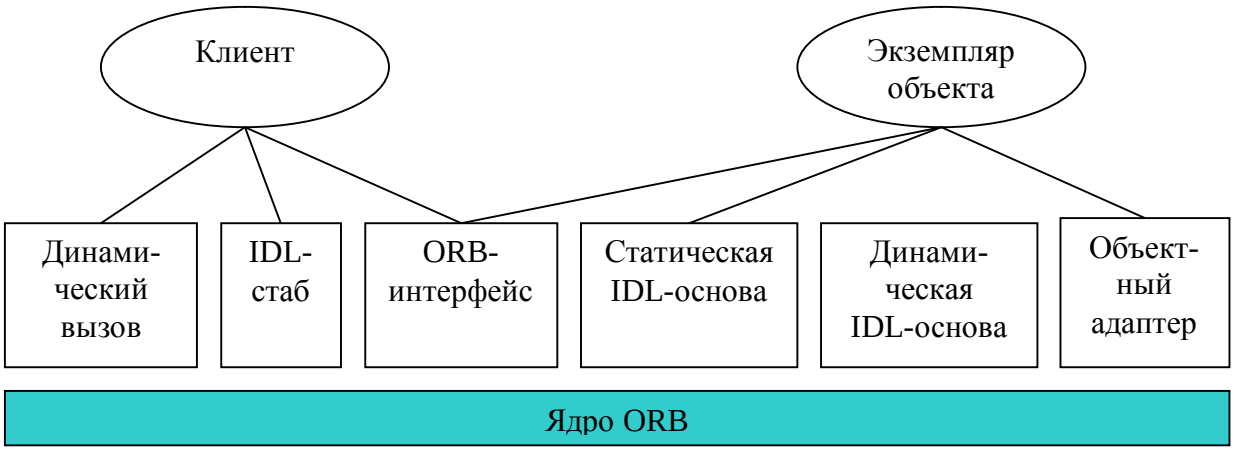


Рис. 5.8. Объектный адаптер как компонент в структуре интерфейса ORB

Главная функция объектного адаптера, используемого для реализации объекта, - предоставление клиентам доступа к сервисам объектного брокера запросов ORB. Объектный адаптер обеспечивает все низкоуровневые средства для связи объекта с его клиентами. Основными задачами объектного адаптера являются:

- генерация ссылок на удаленные объекты;
- вызов метода объекта, определенного в IDL;
- обеспечение безопасности взаимодействия;
- активизация и деактивизация объектов;
- установление соответствия между ссылками на удаленные объекты (proxy) и реальными экземплярами объектов;
- регистрация объектов.

Спецификация OMG CORBA определяет базовый объектный адаптер (Basic Object Adapter – BOA), который должен быть реализован во всех брокерах запросов.

3.5.3 Брокерные архитектуры (CORBA, DCOM)

На сегодняшний день в инженерии программного обеспечения существует 2 основных подхода к разработке программных систем. Первый подход называют функционально-модульным или структурным, в основу которого положен принцип алгоритмической декомпозиции, когда выделяются функциональные элементы системы и устанавливается строгий порядок выполнения действий. Второй, объектно-ориетированный подход использует объектную декомпозицию. При этом поведение системы описывается в терминах взаимодействия объектов.

Можно отметить, что главным стимулом к созданию распределенных систем стало появление персональных компьютеров и коммерциализация Интернета. Одной из первых, объединенных в рамках отрасли попыток по созданию стандартов распределенных систем, была предпринята в конце 80-х, когда была образована организация Distributed Computer Environment (DCE), но она не получила широкой поддержки. Следующим шагом по созданию инфраструктуры распределенных вычислений стала инициатива Common Object Request Broker Architecture (CORBA), предложенная консорциумом Object Management Group (OMG), который в соответствии со своим названием был сосредоточен на объектном подходе к кросс-платформенным приложениям. Microsoft на появление CORBA ответила компонентной объектной моделью Component Object Model (COM) и ее распределенной версией DCOM. Впрочем, и то, и другое решения уступают инициативе DCE по возможности масштабирования. Архитектуры CORBA и DCOM создают иллюзию превращения сети однородных компьютеров в единый образ машины, в которой обмен между приложениями осуществляется на уровне объектов. Средствами CORBA и DCOM строится программное обеспечение промежуточного слоя, независимое от операционных систем и от языков программирования. Специфической чертой обеих технологий является наличие простой схемы обнаружения нужных для выполнения объектов; для этой цели объектам присваиваются идентификаторы, а CORBA еще дополнительно поддерживает описание служб, предоставляемых объектами. Примерно такой же функциональностью обладает Java RMI (Sun Microsystems), с поправкой на нейтральность по отношению к аппаратным платформам, обеспечиваемую виртуальной машиной Java; интерфейс Java Native Interface служит для расширения языковых возможностей.

Будучи распределенными, архитектуры CORBA и DCOM относятся к категории «жестко связанных» систем: каждая из архитектур имеет собственную систему двоичного кодирования сообщений. Объекты и методы действуют в рамках своей архитектуры, объект из CORBA не может вызвать метод из DCOM и наоборот.

Для того чтобы снять ограничения на масштабирование, приложения, являющиеся компонентами системы, должны быть «слабо связанными». Впервые идея объединения компонентов в систему посредством служб приобрела привлекательность на аппаратном уровне, когда стали появляться более интеллектуальные и во все большей степени самоуправляемые устройства, способные предоставлять услуги другим устройствам. Виртуализация устройств, реализуемая при таком подходе, позволяет перевести пользование физическими устройствами в обращение к виртуальным

устройствам, иначе говоря, в термины обращений к объектам. По мере того, как расширялись физические границы систем, формировалась «службо-центричная» (service-centric) архитектура, предполагающая коммуникации между процессами. Благодаря этому стал возможен поворот к ориентированному на службы программному обеспечению промежуточного слоя, т. е. к службам без префикса Web. Первыми ласточками были клиентская служба e-speak от компании HP, Jini от Sun Microsystems и T-Spaces от IBM.

Основополагающими стали спецификации, получившие название Object Managment Architecture (OMA).

OMA состоит из четырех основных компонент, представляющих собой спецификации различных уровней поддержки приложений (рис. 5.2):

- архитектуры брокера запросов объектов (CORBA – Common Object Request Broker Architecture) – устанавливает базовые механизмы взаимодействия объектов в гетерогенной зоне;
- сервисов объектов (Object services) – являются основными системными службами, используемыми разработчиками для создания приложений;
- универсальных средств (Common Facilities) – ориентированы на поддержку пользовательских приложений, таких, как электронная почта, средства печати и т.д.;
- объектов приложений (Application Objects) – предназначены для решения конкретных прикладных задач.

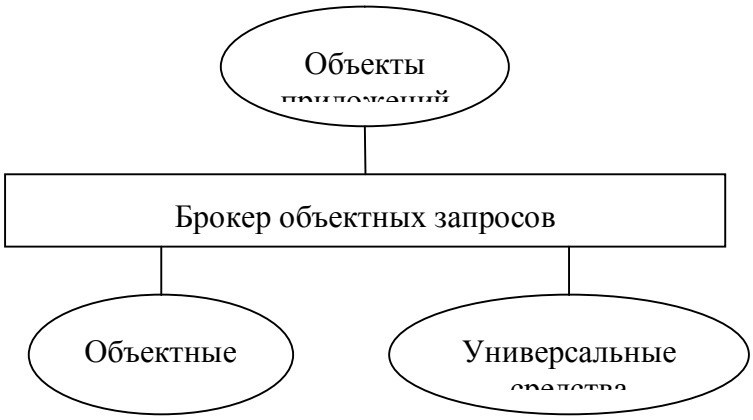


Рис. 5.2. Архитектура управления объектами (OMA)

COBRA определяет механизм, обеспечивающий взаимодействие приложений в распределенной среде.

Главными компонентами стандарта CORBA являются:

- объектный брокер запросов (Object Request Broker);
- язык определения интерфейсов (Interface Definition Language);
- объектный адаптер (Object Adapter);
- репозитарий интерфейсов (Interface Repository).

Место CORBA в семиуровневой модели OSI

Наличие большого количества сетевых протоколов в разных операционных системах потребовало разработки спецификации сетевых соединений. Семиуровневая модель Open System Interconnection (OSI) обеспечивает описание сервисов, которые каждый уровень должен представлять для реализации соединения. Низший уровень – физический – определяет доступ к физической линии. Уровень данных обеспечивает достоверную передачу данных по физической линии. Сетевой уровень имеет дело с установкой соединения и маршрутизацией. Транспортный уровень отвечает за достоверную передачу до точки назначения. Уровень сессий обеспечивает управление соединением. Уровень представлений описывает синтаксис данных и

обеспечивает прозрачность для приложений. Последний уровень – уровень приложений – обеспечивает соответствующие сетевые функции для конечного пользователя.

Концептуально CORBA относится к уровням приложений и представлений. Она обеспечивает возможность построения распределенных систем и приложений на самом высоком уровне абстракции в рамках стандарта OSI. С ее помощью возможно изолировать клиентские программы от низкоуровневых, гетерогенных характеристик информационных систем. Характерные особенности разработки по технологии CORBA заключаются в следующем.

Язык описания интерфейсов OMG IDL позволяет определить интерфейс, независимый от языка программирования, используемого для реализации.

Высокий уровень абстракции CORBA в семиуровневой модели OSI позволяет программисту не работать с низкоуровневыми протоколами.

Программисту не требуется информация о реальном месте расположения сервера и способе его активизации.

Разработка клиентской программы не зависит от серверной операционной системы и аппаратной платформы.

Объектная модель CORBA

Объектная модель CORBA определяет взаимодействие между клиентами и серверами. Клиенты – это приложения, которые запрашивают сервисы. Серверы – это приложения, представляющие сервисы.

Объекты-серверы содержат набор сервисов, разделяемых между многими клиентами. Операция указывает запрашиваемый сервис объекта-сервера. Интерфейсы объектов есть описание множества операций, которые могут быть вызваны клиентами данного объекта. Реализации объектов – это приложения, реально исполняющие сервисы, запрашиваемые клиентами.

Объектный брокер запросов (ORB)

Спецификация CORBA разработана для обеспечения возможности интеграции совершенно различных объектных систем.

Его задачей является представление механизма выполнения запроса, сделанного клиентом: поиск объекта, к которому относится данных запрос, передача необходимых данных, подготовка объекта к обработке. Интерфейс, с помощью которого клиент может запрашивать выполнение необходимых операций, не зависит от местонахождения объекта и языка программирования, с помощью которого он реализован. Клиент может запрашивать выполнение операций с помощью ORB несколькими способами. На рис. 5.3 показаны способы возможного взаимодействия объектов.

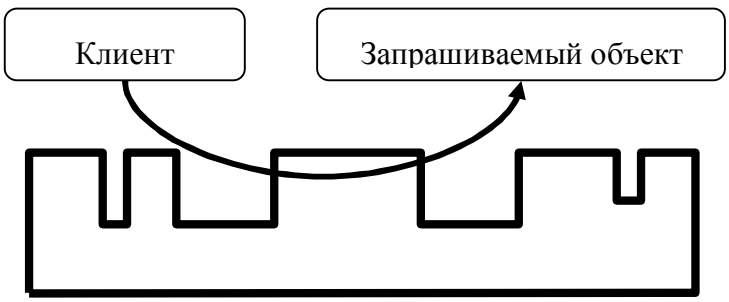


Рис. 5.3. Схема работы объектного брокера (ORB)

Структура ORB

Рис. 5.3. показывает запрос, посылаемый пользователем к системе объектной реализации. Клиент – это пользователь, который желает выполнить операцию над объектом в системе объектной реализации.

Система объектной реализации – это фактически код и данные, которые физически содержатся в объекте. ORB ответствен за все механизмы, требуемые, чтобы найти объектную реализацию для запроса, подготовить ее, реализовать объект и содержащиеся в нем метод и отправить данные, составляющие запрос, пользователю. Интерфейс, с которым имеет дело пользователь, полностью независим от того, где объект размещен, какой язык программирования в нем

используется, где это выполнено, а также в отношении любого другого аспекта, который совершенно не влияет на интерфейс объекта.

Рис. 5.4 показывает структуру индивидуального ORB. Интерфейсы к ORB показываются закрашенными прямоугольниками, и стрелки указывают, вызывается ли ORB или выполняет вызов-запрос при помощи интерфейса.

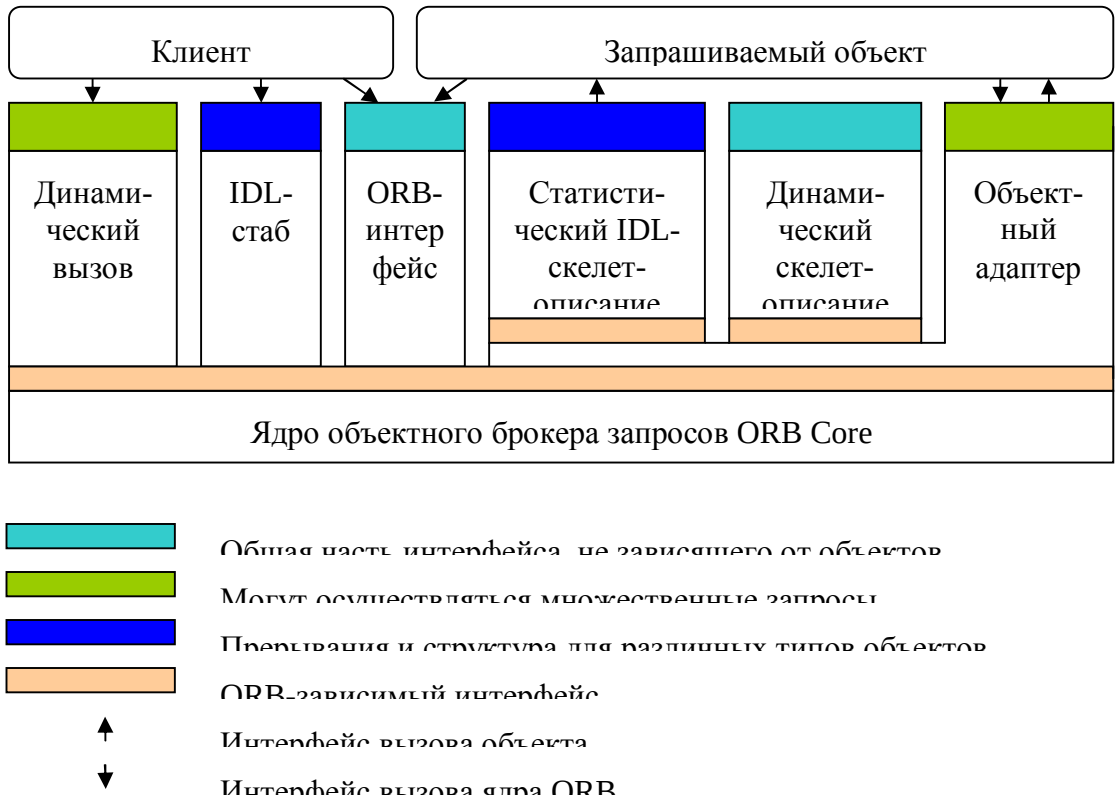


Рис. 5.4. Структура интерфейса объектного брокера запросов

Вызов операций разделяемого объекта-сервера может быть сделан статическим (IDL-стаб) и динамическим (Dynamic Invocation Interface) способом. Интерфейсы описываются, используя язык определения интерфейсов, получивший название OMG Interface Definition Language (IDL). В случае статического вызова эти описания интерфейсов отображаются в программный код на языках C, C++, Smalltalk. Информация об интерфейсах объектов может быть получена клиентом двумя способами: статически (compile time) и динамически (runtime). Интерфейсы могут быть также указаны с помощью службы репозитория интерфейсов (Interface Repository). Этот сервис

представляет интерфейсы как объекты, обеспечивая доступ к ним во время работы приложения (в runtime-режиме). (Описание динамического вызова интерфейсов и репозитария интерфейсов см. ниже.)

Пользователь выполняет запрос при наличии доступа к объекту, когда знает и тип объекта и желаемую функцию выполнения. Пользователь инициализирует запрос, вызывая подпрограммы статической настройки, которые являются специфическими для объекта, или строит динамически оформляемый запрос.

Объектный адаптер (Object Adapter)

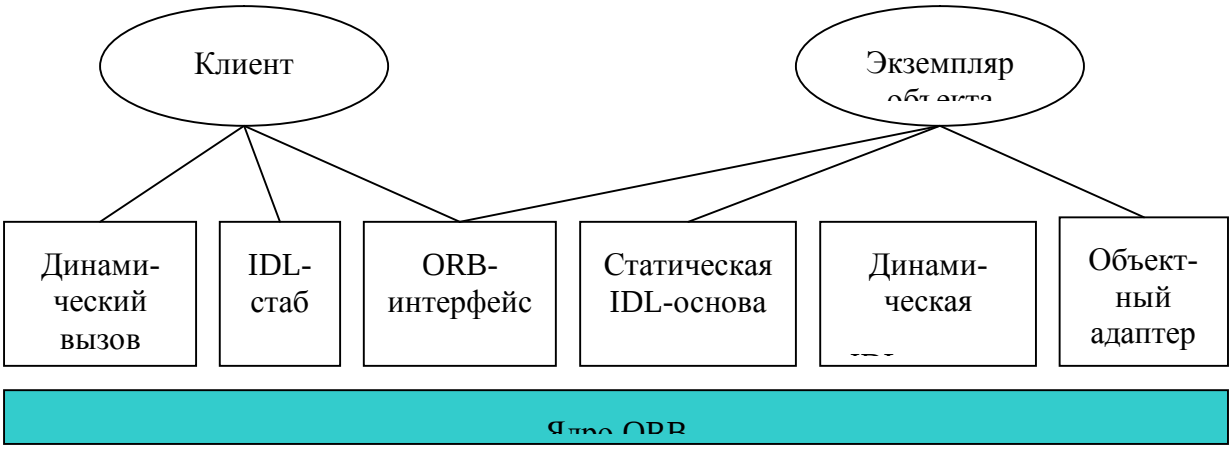


Рис. 5.8. Объектный адаптер как компонент в структуре интерфейса ORB

Главная функция объектного адаптера, используемого для реализации объекта, - предоставление клиентам доступа к сервисам объектного брокера запросов ORB. Объектный адаптер обеспечивает все низкоуровневые средства для связи объекта с его клиентами. Основными задачами объектного адаптера являются:

- генерация ссылок на удаленные объекты;
- вызов метода объекта, определенного в IDL;

- обеспечение безопасности взаимодействия;
- активизация и деактивизация объектов;
- установление соответствия между ссылками на удаленные объекты (proxy) и реальными экземплярами объектов;
- регистрация объектов.

Спецификация OMG CORBA определяет базовый объектный адаптер (Basic Object Adapter – BOA), который должен быть реализован во всех брокерах запросов.

Современные бизнес-компоненты создаются сегодня под лозунгом «Быстрее, легче и гибче» и . Преимущества подобного подхода можно выразить в длинном списке: масштабируемость, стабильность, управляемость, настраиваемость, гибкость, переносимость, возможность многократного использования. Но, к сожалению, чудес в жизни не бывает и, если сложностей и неурядиц убыло в одном месте, то в другом они как раз прибавились. Для того чтобы компоненты умели взаимодействовать, необходимо развитие «клея» – промежуточного программного слоя.

Если со стороны бизнеса все красиво, как на затейливом персидском ковре, то зато на изнанке – уйма всяких узелков и зацепок. Последние годы мировое программное сообщество усиленно пытается внести порядок в эту изнанку, в чем, надо сказать весьма преуспело. Основная возможность здесь – стандартизация компонентов, приведение их к единой природе. Блоки здания должны быть сопрягаемы. Нити ковра должны быть из близких материалов. Идея здесь достаточно проста – внешние интерфейсы компонентов должны быть описаны в едином стиле – на одном и том же языке. Поэтому два известных на сегодняшний день стандарта, CORBA и COM+ создали свои варианты IDL — Языка Описания Интерфейсов. CORBA, COM+ и технология Java, которая, естественно, использует для описания интерфейсов язык Java, предлагают близкие подходы к методу взаимодействия компонентов. На основе описания внешних интерфейсов создаются прокси (заместители) для клиента и сервера, которые позволяют им связываться друг с другом в реальном времени. Прокси для клиента принято называть стабом, а прокси для сервера – скелетоном. (Поэтому, в частности, мы не будем касаться возможностей динамического взаимодействия компонентов в технологии CORBA, когда внешние интерфейсы не определены на момент компиляции системы и появляются дополнительные динамические элементы.)

После стандартизации интерфейсов, мировое компьютерное сообщество перешло на следующий уровень стандартизации – самих компонентов. Из рис. 3 понятно, что имеется в виду под следующим уровнем стандартизации – все дальше от «железа», все ближе – к пользователю. В технологии CORBA это:

Рис. 3. Стандарты в программных приложениях

- CCM (CORBA Component Model)- компонентная объектная модель, компонентное развитие модели бизнеса BOM – Business Object Model;
- BOCA (Business Object Component Architecture) – принципы архитектуры компонентных систем, развитие OMA (Object Management Architecture) на вышележащий уровень стандартизации;
- CDL (Component Definition Language)- язык определения компонентов, развитие IDL.

Разработка этих стандартов продвигается, правда, не так быстро, как хотелось бы всем заинтересованным сторонам. Но признанным героем на поле стандартизации компонентов стала технология компании Sun – Enterprise Java Beans (EJB). Возможно причина ее успеха в том, что в «семейном кругу» одного языка программирования намного проще решить вопросы взаимодействия. Тем более языка молодого и резвого, который многие проблемы, такие как вызов удаленных методов (RMI – Remote Method Invocation), умеет решать сам. Не исключено, что универсальные цели консорциума OMG, развивающего технологию CORBA, – объединить компоненты, написанные на разных языках программирования, функционирующие на разных системах и разных компьютерах в разных точках земного шара, являются в данном случае некоторым тормозом. В дальнейшем мы увидим, возможно, как две технологии, сомкнувшись, отбросив амбиции, дают замечательный результат на радость всем заинтересованным сторонам.

Что же такое этот «бин» (bean — боб) и почему, стремительно выскочив как джин из бутылки в компьютерный мир, он уже завоевал такую популярность? Если перевести определение Sun как можно ближе к оригиналу – то «это модель для создания и развертывания серверных компонентов многократного использования, написанных на

языке Java.» Продолжим вместе с Sun разбирать это определения, – «компоненты – это заранее разработанные куски программного кода, которые могут быть установлены в работающие прикладные системы». Если классы Java образуют компонентную модель для проектирования приложений в технологии Java, то Java Beans логически развивает эту модель на следующем уровне интеграции создания автоматизированных систем и абстрагирования от процесса программирования – стадии внедрения. По сути, это переход от разработки приложения под заказ из готовых программных компонентов к сборке из готовых EJB действующих компонентов. Если раньше дома складывали из кирпичей, то теперь – из комнат-секций. Слово Enterprise в названии Enterprise Java Beans означает новую ступень технических задач, стоящих перед программными приложениями. Такие задачи привычны для приложений уровня Предприятия: поддержка распределенности, службы именования, транзакций, безопасности, уведомлений-сообщений, долговременного хранения, и т.д. Неудивительно, что этот список напоминает список Служб технологии CORBA.

С точки зрения разработки, EJB представляют собой Java-классы специального типа вместе с описателем-паспортом бина и параметрами среды функционирования, которые бин умеет обрабатывать. Описатель бина (Deployment Descriptor) в свою очередь представляет собой XML-файл, в котором содержатся правила, связанные с управлением бином, например, права доступа пользователей к бину. Несколько бинов могут объединяться, образуя приложения, Java-апплеты и новые бины.

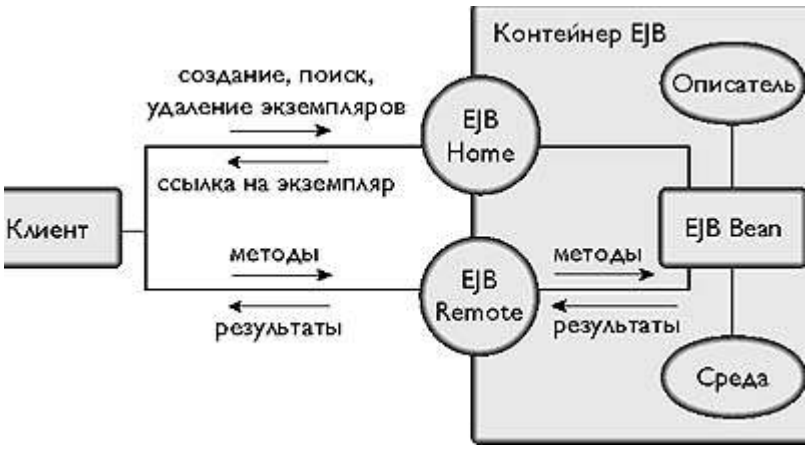


Рис. 4. Контейнер Enterprise Java Beans

По технологии EJB бобы-бины размещаются в контейнере (рис. 4) На контейнер возложены обязанности по защите бинов и поддержке их взаимоотношений с внешним миром: регистрация-прописка объектов, обеспечение для них внешних интерфейсов, создание и разрушение реализаций этих объектов, охрана безопасности, управление их состояниями и координация транзакций. В технологии не определено, как конкретно должен быть реализован контейнер. Это могут быть как многопоточные процессы на отдельном сервере, в которых выполняются бины, так и законченные программные приложения, которые можно переносить или распределять между серверами и/или процессами. Клиентские бины обрабатываются внутри виртуальных контейнеров, таких как Web-страницы, формы, составные документы, и т.д.

В технологии определены два основных типа бинов. Session Beans – сеансы клиент-серверного взаимодействия отрабатывают действия клиента, например, запись в базу данных, выполнение вычислений, и т.д. Это окна клиента в мир программных приложений. Они, в свою очередь, также бывают двух типов:

- Stateless session – не умеют сохранять свое состояние и существуют только на протяжении текущего сеанса. В случае сбоя сеанс не может быть восстановлен;

- Stateful session – сохраняют свое состояние; сеанс может быть восстановлен.

Entity Bean – компоненты объектного представления данных, размещаемых в хранилище. Entity Bean транзакционны и восстанавливаемы. Каждая их реализация имеет уникальную метку, называемую «первичным ключом» (Primary Key) по аналогии с таблицами баз данных. В свою очередь эти бины делятся на две группы по способу определения где, в каком хранилище и как хранятся данные:

- Bean-Managed Persistence – самостоятельные бины, управление хранением осуществляется на уровне бинов;

- Container-Managed Persistence – «опекаемые» бины, чьим хранением заправляет контейнер.

Кроме класса, реализующего функциональность бина, в него входят два обязательных класса, реализующие два типа интерфейсов:

- Home Interface – доступ к «домашним» службам бина, таким как начало или отбой для Session Bean или поиск Entity Bean. Этот интерфейс реализует все службы жизненного цикла компонента и позволяет контейнеру управлять и руководить его поведением;

- Remote Interface – доступ к бизнес-службам бина.

Клиентские приложения общаются с бинами через контейнер, который открывает наружу оба интерфейса бина: Home и Remote. С помощью первого открывается сеанс или находится нужный бин, а с помощью второго – осуществляется отработка действий клиента в Session или транзакционная механика обработки Entity.

Итак, повторим идею технологии с прикладной точки зрения. Прикладная бизнес-логика приложения делится на изолированные бизнес-объекты, каждый из которых реализуется в виде EJB. Они устанавливаются на Сервере Приложений или EJB Сервере и реализуют запрашиваемую логику для клиента (локального или удаленного). Давайте прямо сейчас, при первом появлении понятия Сервер Приложений в технологии Java, развеем непонимание, о котором уже говорилось в начале раздела. Написав оба слова, составляющие понятие, с заглавных букв, мы магическим образом преобразовали сервер приложений, реальную машину из корпуса, начинки и кабелей, в программное приложение.

Обратимся вновь к интерпретации Sun. Под Сервером Приложений в технологии Java понимается программное приложение, обеспечивающее оптимальную среду для выполнения EJB. Сервер Приложений занимается «коммунальным хозяйством» дома – программной системы. На его руках надзор за системными ресурсами, такими как процессы, потоки, память, связь с базами данных, сетевые отношения, выравнивание загрузки распределенных узлов. Кроме этого важнейшая обязанность Сервера Приложений заключается в предоставлении cle;, компонентам.

Еще раз можно подчеркнуть, что технология EJB, впрочем, как и технология CORBA не предоставляет готовые программные решения – контейнеры, Сервера Приложений, а только стандарты на такой тип программного обеспечения.

Стандарты – единственная возможность обеспечить некоторый порядок в бурно развивающемся компьютерном мире. Стандарты необходимы всем трем заинтересованным сторонам в области информационных технологий: разработчикам, потребителям-заказчикам и промежуточному звену – консультантам, интеграторам, продавцам. Первым – для поставки на рынок конкурентоспособных, жизнестойких решений, вторым – как средство борьбы с «зоопарком» программных систем, третьим – как возможность строить гибкие, интегрированные программные среды для своих заказчиков.

Стандарты продолжают захватывать все новые области информационных технологий. Но, если в сетевых технологиях стандарты полностью прижились, то в области программных приложений они еще только набирают силу. Понятны опасения специалистов использовать в собственных решениях «неправильный» стандарт, который зачастую, невзирая на грамотные технические решения, заложенные в его основу, не идет дальше прекрасных инициатив и не получает развития. Выбирая дорогу, всегда есть опасность оказаться в тупике. Именно поэтому стоит внимательно читать указатели (в смысле дорожные знаки) и прислушиваться к мнению мирового сообщества. Серверы Приложений уже сейчас имеют реализации от большинства крупных производителей программных систем (компании Inprise, Oracle, Sybase, Sun, BEA, Iona, IBM). Кроме того, в данной области архитектуры программных приложений пока не появилось ни одного достойного конкурента. Так что стиль пока единственный – Сервер Приложений.

Если посмотреть на Серверы Приложений глазами конечного пользователя, то они представляют собой основные несущие конструкции сооружения под названием многозвенная распределенная компонентная программная система. Именно Серверы Приложений поставляют бизнес-компоненты и обеспечивают необходимый уровень системных служб для этих компонентов, т.е. всю коммунальную систему жизнеобеспечения. Приложения, которые наш Сервер предоставляет клиентам, могут располагаться где угодно, причем эти приложения могут даже не знать, где хранятся данные, с которыми они работают – на каком сервере и в какой базе данных. Для клиентов Сервер Приложений открывает и закрывает сеансы. Для приложений – позволяет настраивать системные службы, из которых важнейшими являются служба долговременного хранения, политика хранения компонентоа Entity Beans, служба транзакций и служба безопасности.

«А надо ли городить огород?, – спросит недоверчивый читатель. – Зачем мне такое дополнительное ПО для моего сугубо конкретного программного приложения?»

Огород, может быть, городить и не надо, а обеспечить газом, электричеством и горячей водой наш блочный дом, программное приложение – необходимо. Да и надежный кодовый замок на входной двери не помешает. Конечно, можно предоставить решать эти проблемы самим жильцам. Пусть каждый в одиночку кипятит воду и укрепляет входную дверь. Тогда уж лучше просто влезть на дерево (я никоим образом не имею в виду службу каталогов NDS) и кушать бананы (не путайте с Baan).

Чтобы постройка программного приложения не напоминала возведение Вавилонской башни, в технологии EJB расписано, кто именно должен принимать участие в проекте. Комплексная бригада строителей здания состоит, по представлению компании Sun, из следующих профессионалов.

Архитектор

Поставщик серверной платформы (EJB Server Provider) – специалист в области распределенной платформы и служб. Он предоставляет инфраструктуру разработки и среду выполнения для программных приложений.

Конструктор

Поставщик контейнеров (EJB Container Provider) – эксперт в области распределенных систем, системной безопасности и поддержки транзакций. Он обеспечивает системный уровень контейнеров, то есть системную оболочку для одного или более компонентов – Enterprise Bean.

Снабженец

Поставщик бизнес-компонентов (Enterprise Bean Provider) – аналитик в функциональной области, который реализует бизнес-компоненты как разработчик или подбирает готовые.

Монтажник

Компоновщик Приложений (Application Assembler) – также эксперт-функционал, собирающий программное приложение из строительных блоков – бизнес-компонентов. Его взгляд на соответствующий бизнес должен быть более общим и универсальным, чем у Снабженца – Поставщика бизнес-компонентов.

Прораб

Внедренец (Deployer) – специализируется в установке приложений. Он настраивает приложение на реальную среду.

Управдом

Администратор системы (System Administrator) – обеспечивает работоспособность системы на этапе эксплуатации.

Конечно, то, что компания Sun описала именно такие роли, не означает, что любой проект, связанный с Enterprise Java Beans обязательно требует одного и только одного указанного специалиста и именно такой состав комплексной бригады. Да и найти, скажем, готового поставщика контейнеров не так-то просто. Что совершенно необходимо – так это не упустить специфические для компонентного проекта моменты: выбор архитектуры и способы реализации системных служб, определение структуры и поведения контейнеров и проработка вопросов внедрения (развертывания).

Завершив экскурс в EJB, можно привести определение Сервера Приложений «с большой буквы». Сервером Приложений называется программный продукт, поставляющий среду выполнения для прикладных компонентов, расположенный между клиентом – с одной стороны и данными и прикладными программами – с другой, который может обеспечить интеграцию различных источников данных и прикладных ресурсов и предоставить компонентам необходимые для них сервисы.

Вам намеренно привено универсальное определение Сервера Приложений, которое относится ко всем типам компонентов (CORBA, EJB, COM+). Серверы Приложений, зародившись внутри технологии EJB, оказались столь удобны, что достаточно уверенно продвигаются как единое решение для всех компонентных технологий. Реализации Серверов Приложений уже умеют работать с разными

компонентами. В качестве примера можно привести Application Server компании Inprise, который можно успешно использовать в среде компонентов EJB и CORBA.

Другой наблюдаемый процесс – смыкание технологий Java и CORBA. С небольшой долей натяжки можно считать, что EJB могут иметь двойное гражданство, а именно представлять собой еще и CORBA-объекты. Поддержка CORBA является необходимым условием для конкурентности реализации Сервера Приложений. Ведь из всех перечисленных технологий только эта поддерживает Унаследованные Системы – функционально пригодные, но технически устаревшие. Если считать парк автоматизации современного предприятия или компании некоторым поселением – деревушкой или городком, создаваемая интеллектуальная компьютерная среда должна включать уже существующие постройки. Одинаково неправильно требовать сноса существующих зданий или не обращать на них внимания.

Другой аспект выбора реализации Серверов Приложений заключается в функциональности, которая наиболее приоритетна. Попробуем разделить Серверы приложений на группы.

1. Интеграционный Сервер Приложений. (Application -integration-centric application server)

Основная задача такого Сервера – интеграция Бизнес-приложений в единую интеллектуальную среду. Такие Серверы особенно актуальны для организации приложений, связанных с задачами типа Supply Chain (Цепочки Поставщиков) и электронной коммерции. К таким серверам относятся реализации крупнейших поставщиков брокеров объектных запросов – компаний Inprise и Iona.

2. Информационный Сервер Приложений (Data-centric application server)

Деятельность такого сервера сконцентрирована на манипулировании данными. В качестве хранилищ могут выступать произвольные приложения, которые, в частности, не обязательно представляют собой реляционные базы данных и, уж тем, более объектные. Такие серверы берут на себя организацию представления распределенных данных, хранящихся в различных источниках, в объектной парадигме. Естественно, Oracle поставляет серверы именно такого типа.

3. Обрабатывающий Сервер Приложений (Processing-centric application server)

Центральной заботой таких серверов является обеспечение специфической бизнес-логики. Например, сервер, реализующий распределенные вычисления, или сервер ERP-системы. К таким реализациям относится IBM Websphere.

4. Управляющий Сервер Приложений (Rules-based application server)

Такие серверы являются подмножеством Обрабатывающих Серверов Приложений, но в отличие от них сконцентрированы на выполнении определенных бизнес-правил. Они ориентированы прежде всего на область электронной коммерции. Примером таких серверов является реализация от компании Vision Software.

5. Сервер XML (XML Server)

Этот сервер представляет собой подмножество Информационного сервера, с той лишь разницей, что в качестве хранилищ данных выступают XML – документы. С другой стороны, такой сервер может представлять собой подмножество Интеграционного сервера, в котором в качестве механизма интеграции выбран XML. Известна реализация такого сервера от компании Bluestone Software.

Не следует думать, что разделение Серверов Приложений по группам такое четкое. Наиболее интересные экземпляры Серверов Приложений гармонично сочетают в себе достоинства различных групп. Кроме того компании-разработчики активно развивают свои продукты. А каждая новая версия, естественно, богаче предыдущей.

Можно предвидеть следующий вопрос читателей: – « А это уже где-нибудь работает?» Ответ – «Да!». Особенно востребованы Сервера Приложений во всем, что касается использования Internet. И за счет поддержки распределенности, возможностей выравнивания загрузки, отслеживания распределенных неоднородных транзакций, управления безопасностью и многих других своих возможностей. Именно эти интеллектуальные приложения находят в Серверах Приложений могучую опору.

Рис. 5. Java 2 Enterprise Edition

Вскользь упомянем о следующих усилиях по созданию технологии компонентного программного обеспечения, снова связанных с Java. Это J2EE (Java 2 Enterprise Edition) – стандарт для многозвенных систем уровня предприятия, прикладная платформа, обеспечивающая взаимодействие Enterprise Java Beans, Java Server Pages, апплетов и сервлетов. рис. 5 в точности иллюстрирует то, как это выглядит у компании Sun.

Средства взаимодействия компонентов располагаются в нижней части рисунка. J2EE впервые стандартизовала выполнение родного для Java протокола удаленных вызовов методов через Internet — IIOP (Internet InterOrb Protocol). Таким образом, к сотрудничеству между CORBA и Java добавился еще один пункт. Серверные компоненты заключены в контейнеры, взаимодействующие с помощью специальных коннекторов. На уровне контейнеров задаются системные сервисы: Транзакций, Сообщений и Почты. На верхнем уровне находится API-интерфейс и Средства управления. Сказать об этой технологии в контексте Серверов Приложений меня вынудило не только то, что этот стандарт является логическим развитием технологии EJB, в которой впервые определился наш герой, но и то, что уже появляются Серверы Приложений, удовлетворяющие новому стандарту. Имеется в виду уже упоминавшийся Application Server (Inprise).

Но, стоп, что Вы уже устали от обилия технологий и нагромождения стандартов? Если да – то тогда обратитесь к рис. 6, где сделана попытка свести все воедино.

Рис. 6. Стандарты распределенных программных систем

Возможно мы будем свидетелями рождения новых технологий интеграции, например, на стыке объектных подходов в проектировании БД, программирования и создания интерфейсов.

3.5.4 Корпоративный информационный портал

Компания Merrill Lynch одной из первых воспользовалась термином «корпоративный информационный портал» (enterprise information portal — EIP). Она так определяет это понятие: «Корпоративные порталы –программные приложения, позволяющие компаниям «расконсервировать» информацию, сохраняемую как внутри, так и вне их границ, а также предоставить каждому пользователю единую точку доступа к предназначенной для него информации, необходимой для принятия обоснованных управленческих решений».

Портал и управляемость компании

Исследование, проведенное компанией RewardsPlus, показало, что организации начали инвестировать большие средства в создание корпоративных порталов для улучшения внутри корпоративных коммуникаций, аутсорсинга трудовых ресурсов, а также для социализации сотрудников компании и укрепления у них чувства самоидентификации с ней. По мнению аналитиков, портал должен не только обеспечивать выполнение корпоративных функций, но и помочь сотрудникам сбалансировать свои производственные обязанности и личные потребности. Руководители должны иметь в виду, что, поскольку рабочее место становится все более виртуальным, корпоративный портал будет основным средством общения внутри компании. Для руководителей портал — это канал взаимодействия с сотрудниками, ресурс укрепления организационной морали; для сотрудников — лучший способ решать повседневные проблемы.

В чем состоят резервы повышения управляемости организации при внедрении корпоративного информационного портала? Как известно, деятельность менеджеров любого уровня отличает необходимость принимать решения в условиях недостатка информации или при наличии противоречивых данных. Умение работать в условиях неопределенности, принимать правильные решения всегда может считаться одним из главных критериев эффективности деятельности управленца. С точки зрения налаживания коммуникаций корпоративный портал решает ряд задач:

- прохождение информационных потоков между подразделениями;
- устранение различия в «языке» между отдельными подразделениями и отдельными сотрудниками благодаря использованию единых форматов распространения информации;
- фильтрация потоков информации благодаря возможности настраивать профили пользователя;
- минимизация искажений информации при ее прохождении по многочисленным вертикальным и горизонтальным каналам внутри организации;
- упрощение процесса обучения сотрудников;
- обеспечение своевременного доступа к корпоративной информации.

Все это позволяет рассматривать корпоративный портал как систему управления знаниями, охватывающую большую часть корпоративной информации.

Технология серверов Web-порталов

Технология серверов порталов — еще довольно новая и потому фрагментарная область, но сейчас уже разрабатываются необходимые стандарты. Тем не менее, на

рынке уже есть несколько достаточно зрелых продуктов, которые определяют состояние дел в этой области.

Большинство современных серверов порталов базируются на Java. Компании Epicentric и Plumtree, пожалуй, первыми предложили серверы порталов как отдельные продукты. С приобретением компании TopTier корпорация SAP смогла присоединиться к ним имеющемуся в ее системе уровню интеграции приложений iViews. Корпорация IBM предлагает WebSphere Portal Server, который базируется на технологиях свободно распространяемой портальной платформы Jetspeed, реализованной в рамках проекта Apache. Со своей стороны, Apache принимает участие в разработке спецификаций для API портлетов в стандарте J2EE, благодаря чему Jetspeed стал серьезным кандидатом на роль эталонной реализации нового стандарта. Еще одна свободно распространяемая портальная платформа Zope реализована компанией Python. Помимо функций портала Zope предлагает некоторые возможности управления информационным наполнением и общие службы Web-приложений. (Этот список, конечно, не полон.)

В ближайшем будущем мы станем свидетелями стабилизации и более надежной интероперабельности между серверами порталов различных производителей по мере того, как они будут реализовывать ориентированные на порталы стандарты. Включение в состав спецификации J2EE интерфейса API портлетов также будет способствовать значительному расширению использования технологии портальных серверов, учитывая известность спецификации J2EE.Что касается других элементов спецификации J2EE, популярность свободно распространяемых реализаций будет расти, вынуждая к тому производителей, стремящихся сохранить свои конкурентные преимущества. В ближайшем будущем скорость внедрения технологии будет определять жизнеспособность рынка служб портлетов. Наконец, в перспективе, технология серверов порталов может распространяться на новые области. Представьте, к примеру, принтер, который предлагает свой пользовательский интерфейс управления как Web-службу удаленного портлета в рамках портала системного управления, обратиться к которому можно по телефону.

Попытки стандартизации серверов порталов

Попытки стандартизации предпринимаются в двух важных областях: API-интерфейсы портлетов и Web-службы для удаленных порталов.

API-интерфейсы портлетов

Эксперты Java Community Process работают над расширением стандарта J2EE с целью унифицировать свободно распространяемое программное обеспечение и коммерческие продукты различных производителей. Помимо унификации необходимо гарантировать, что API-интерфейсы портлетов будут основаны на открытых стандартах.

Эта спецификация будет состоять из самих портлетов, API-интерфейсов портлетов и формата дескриптора развертывания. Сегодня, однако, не существует открытой для публики предварительной спецификации; нет эталонной реализации.

Учитывая, что в работе этой группы участвуют разработчики Apache, их проект Jetspeed, вероятно, будет выбран в качестве эталонной реализации. Однако ряд производителей порталов, также принимающих участие в разработке спецификации, намерены опираться на стандарты, а не на свободно распространяемую реализацию.

Web-службы для удаленных порталов

Технический комитет организации Organization for the Advancement of Structured Information Standards сейчас разрабатывает стандарт на Web-службы для удаленных порталов. Данный стандарт, WSRP, позволит организовать взаимодействие ориентированных на пользователей Web-служб с порталами и другими Web-приложениями промежуточного слоя. Это также даст возможность реализовывать удаленные портлеты на любой платформе, будь то J2EE, .NET или некоторая служба, поддерживающая SOAP.

3.5.5 Web-службы и средства интеграции приложений предприятия (EAI)

Общественное мнение сегодня практически целиком сосредоточено на тех Web-службах, которые строятся на стеке стандартов SOAP, UDDI и WSDL. Сказанных за пару лет их существования слов более чем достаточно для того, чтобы сложилось впечатление вполне законченной картины, а сами Web-службы представляются некоей подлинной реальностью, существующей давным-давно [1].

На самом деле ситуация куда сложнее, она лишена однозначности и окрашена определенной драматичностью. Во-первых, данный стек протоколов — далеко не единственный подход к созданию распределенных вычислений средствами Интернета. Во-вторых, внутри лагеря приверженцев именно этой категории Web-служб пока еще нет полного единства взглядов. И, наконец, в-третьих, среди лидеров закрепились, скажем так, некоторые явно «антисетевые» компании, а сделавшие для развития Сети больше других, отнюдь не доминируют. Первые вынуждены менять свое лицо, а вторые явно чувствуют себя ущемленными. К сегодняшнему дню материализация Web-служб сильно преувеличена: они еще слишком далеки от практической реализации, и, соответственно, не сложились стабильные решения, связанные с интеграцией приложений предприятия (enterprise application integration — EAI) на основе именно этого типа Web-служб. И все же, несмотря на преждевременность, говорить об этом приходится, повинуясь требованиям времени.

Начать стоит с одного важного замечания: несмотря на то, что в названиях двух технологий — World Wide Web и Web-службы — присутствует объединяющее их слово, их пути к потребителю отличаются, как небо и земля. Всемирная паутина — гениальное открытие небольшой группы ученых, возглавляемых Тимом Бернерсом-Ли. Оно было построено на ограниченном количестве стандартов, не потребовало слишком больших инвестиций и дошло до каждого пользователя Интернета естественным образом и без каких-либо маркетинговых усилий. Напротив, Web-службы, явление в своей отдаленной перспективе столь же глобальное, требуют для своей реализации консолидации практически всех серьезных производителей программного обеспечения, колоссальных средств и огромного, еще не вполне оцененного количества стандартов. Сегодня трудно найти американскую ИТ-компанию, которая бы не причисляла себя к производителям Web-служб. Но не менее трудно найти у представителей этих компаний внятное объяснение того, что они делают.

Выработкой стандартов занимается множество организаций и комитетов. На ведущую роль в этом движении к стандартизации претендует учрежденная Microsoft и IBM организация WS-I (Web Services Interoperability Organization). Две эти корпорации являются ее членами-учредителями и пользуются, наряду с еще несколькими ведущими производителями определенными привилегиями. WS-I появилась позже других аналогичных образований, но именно она координирует вопросы стандартизации Web-служб, поэтому работает в тесном сотрудничестве с такими доминирующими в этой области организациями, как World Wide Web Consortium и Internet Engineering Task Force. В WS-I вошли также компании Oracle, BEA Systems и Hewlett-Packard. Однако среди ее членов пока нет Sun Microsystems, которая ранее продвигала альтернативный подход к службам, основанный на стандарте ebXML, и, учитывая свои заслуги по части Java, требует теперь для себя в случае вступления исключительного положения среди участников WS-I, равного статусу Microsoft и IBM. Даже по неполному перечню комитетов и организаций несложно сделать вывод о том, что процесс стандартизации идет полным ходом. (Содержательный обзор последних событий в области стандартизации Web-служб можно найти в [2].)

Беспрецедентная активность крупных фирм-производителей ПО объясняется тем, что Web-службы оцениваются всеми ими как безусловно перспективное начинание. **Однако Web-службы относятся к той категории новаций, которые создаются и продвигаются компаниями-разработчиками, а пользователями могут быть востребованы только тогда, когда достигнут определенного уровня зрелости**. Не то, что бы Web-службы навязываются этими фирмами, но потребность в них не осознана пользователями и не вызрела в полной мере.

Потребителей нужно разбудить, а потребность воспитать. Между хронологически первыми разработками такого типа и массовым принятием технологии проходит длительное время, в течение которого процесс ее становления подчиняется закономерности, которая с подачи самого известного аналитика ИТ-рынка, компании GartnerGroup, получила название «кривой хайпа». Поскольку английское слово hype имеет очевидный негативный оттенок — а оно действительно переводится как «активная реклама», «пускание пыли в глаза» и даже «очковтирательство» — кривую хайпа чаще рассматривают не как объективную закономерность, а как сознательную деформацию естественных процессов, вызванную чрезмерными маркетинговыми действиями заинтересованных сторон.
Скорее всего, подобные утверждения ошибочны; более вероятно, что каждому типу новаций соответствует определенная динамика перехода технологии или изделия от начальной фазы до массового принятия потребителями. Сравните: это могут быть мгновенные взрывы, как это произошло с простейшими изделиями, например, кубик Рубика или томагочи, а может быть вообще неопределенное время внедрения, как в случае с управляемой термоядерной реакцией или нанотехнологиями (понятно, что они когда-нибудь будут, но когда — неизвестно).

Enterprise Application Integration, это название по своей нелогичности может составить конкуренцию Web-service или knowledge management. Речь ведь идет о системе управления предприятием, а в таком случае механическая интеграция отдельных подсистем находится в противоречии не только с положениями общей теории систем, но и со здравым смыслом. Трудно представить себе, что систему управления летательным аппаратом будут интерпретировать как интеграцию отдельных модулей. Рисуется страшная картина: стоящий на поле самолет, а рядом ворох оборудования, во много раз превосходящий его по размерам. Рожденный летать в таком случае даже ползать не сможет.

Сравним отношение к предмету или объекту управления в технических системах и корпоративных информационных системах. Любой вузовский курс по специальности «Системы автоматического управления» включает в себя предмет «Теория автоматического регулирования». Почти полностью этот предмет строится на основе математических моделей объектов управления; изучающий его студент, быть может, еще и в глаза не видел реальных регулирующих устройств и объектов управления, на которых они работают. И это только начало: базис технических систем автоматического регулирования образован целым комплексом математических дисциплин. Именно они являются квинтэссенцией знаний, а инженерными методами создаются конкретные технические устройства; соответственным образом распределены роли ученых и практиков. Если же мы обратимся к корпоративным информационным системам, то здесь нет ни нормальных моделей, ни специалистов по управлению, а есть какие-то странные словосочетания, как, например, «интеграция приложений»...

Впрочем, в этом комментарии, как в большинстве многих эмоциональных высказываний, есть небольшое преувеличение — нельзя грести все информационные системы под одну гребенку. Для корпоративных информационных систем, как для

любых других, характерно волнообразное развитие. На первой волне (70-е годы) создавались в основном финансовые приложения, на второй (80-е годы) — системы планирования ресурсов предприятия (enterprise resource planning — ERP), на третьей (90-е годы) — системы управления отношениями с клиентами (customer relationship management — CRM). **Нынешняя, четвертая, обладает новым качеством. В информационных системах появляются самообслуживание и самоуправление, которые и могут быть реализованы средствами слабосвязанных приложений — и Web-службами в том числе. В результате эволюционного развития корпоративные системы превращаются в новый «зоопарк» — зоопарк приложений (см. раздел 3.4).**

Раньше подобным образом характеризовали комплекс разнородных аппаратных средств; со временем сетевые стандарты и инфраструктуры позволили связать их вместе. Однако корпоративный зоопарк не исчез, изменились его «обитатели», и ныне большинство компаний располагают набором приложений, который надо как-то упорядочить [4]. Это могут быть старые унаследованные и плохо документированные приложения, а могут быть и современные коробочные продукты классов ERP, CRM и SCM (supply chain management — «управление цепочками поставок»), а также порталы. Обычно они представляют собой «черные ящики», которые работают каждый на собственном наборе данных, выполняют возложенные на них функции, но не могут обмениваться между собой данными в режиме реального времени и не образуют единую систему.

Развивающиеся системы категорий B2B, B2C и аналогичные им требуют представления полной информации в приемлемое время. Необходимость в коммуникации с внешним миром является первостепенным по значимости требованием для создания интегрированной системы приложений предприятия (enterprise application integration — EAI). Процесс создания EAI — это процесс объединения систем ERP, CRM, SCM, баз данных, хранилищ данных и других внутренних подсистем предприятия.

Подчеркнем, что Web-службы не следует путать с собственно EAI. **Web-службы — это всего лишь еще одна технология, обеспечивающая создание интегрированной платформы, но в отличие от более традиционных подходов использующая для этой цели слабо связанные приложения.** При этом не стоит принижать возможности действующих средств, предназначенных для интеграции приложений, которые сложились за последние годы: они обладают вполне достаточными возможностями и сохранят применимость еще на годы.

Очень интересное сравнение возможностей Web-служб и архитектуры J2EE Connector Architecture можно найти на сайте dev2dev.com, поддерживаемом компанией BEA Systems специально для разработчиков. Технологии Web-служб, во всяком случае, по их состоянию на сегодняшний день и на ближайшее будущее, не перекрывают все возможности перечисленных выше четырех подходов к интеграции. Сейчас они привлекательны такими возможностями, как быстрая публикация, обнаружение и включение служб в приложение, но **в этом виде они могут быть использованы только на уровне функциональной интеграции.** Будущие поколения Web-служб обретут способность инкапсулировать и пользовательские интерфейсы, и средства для обеспечения безопасности, и многое другое, что образует потребительские качества корпоративной информационной системы. Для полноценного использования этих возможностей потребуются значительные изменения в самих интегрируемых компонентах, которые должны будут обладать способностью предоставления своей функциональности в виде служб, чтобы можно было использовать в качестве связующих средств XML, SOAP и UDDI. **На этом уровне задачи EAI сведутся не к**

интеграции приложений, а к интеграции служб. Несмотря на такую острожную оценку возможностей Web-служб, в их нынешнем состоянии создание интегрированных приложений описанными средствами обладает значительными преимуществами, которые проявятся в будущем.

3.5.6 Поисковые системы. Контекстный поиск. Проблемы

Традиционная технология информационного поиска в Web

В 1990 году в университете МакГилл в Монреале был разработан Archie – первый механизм поиска в Internet. Archie ищет файлы на FTP-серверах. В этот период были сделаны еще две поисковые машины для поиска на серверах службы gopher: Veronica, разработанная в 1992 году в Университете штата Невада и Jughead, созданная в 1993 году в Университете штата Юта.

Существующие службы поиска разделяются на поисковые машины и каталоги.

Поисковые машины (такие как AltaVista и HotBot) традиционно включают три компонента: программу сканирования (crawler), индекс и программное обеспечение поиска. Crawler, или spider (паук) – это программа, которая автоматически просматривает различные web-сайты и создает индексы ресурсов URL, ключевые слова, ссылки и текст. Программа-crawler может также переходить по расположенным на сайте ссылкам на другие, близкие по содержанию, страницы. При этом она периодически возвращается к исходным сайтам, чтобы проверить, не произошли ли какие-нибудь изменения. Когда пользователь делает поисковой машине запрос, ее программное обеспечение проходит по созданному индексу в поиске Web-страниц с заданными ключевыми словами и классифицирует эти страницы по степени близости к предмету поиска.

Каталоги (например, LookSmart и Yahoo) работают не с индексами, а с описателями Web-страниц, составленными либо Web-мастерами, либо специальными редакторами, которые просматривают Web-страницы. В ответ на запрос каталоги выполняют поиск по этим описателям. Некоторые поисковые машины, например, Microsoft MSN и Netscape Search, на самом деле являются гибридными технологиями, поскольку тоже используют каталоги. В каталогах не применяется программа-crawler, поэтому они не могут автоматически обнаружить изменения Web-страниц. Однако сторонники этого подхода настаивают на том, что для ряда запросов результаты, полученные по описаниям подготовленным человеком, могут оказаться более осмысленными.

Обычно поисковая машина получает от пользователя одно или несколько ключевых слов вместе с булевыми операторами «И», «ИЛИ», «НЕ» и просматривает индексированные Web-страницы в поиске этих ключевых слов. Для определения порядка вывода результирующих страниц поисковая машина использует алгоритм классификации сайтов, которые содержат заданное ключевое слово.
Поисковый механизм может, например, подсчитать, сколько раз ключевое слово встречается на странице. Он также может искать ключевое слово в **метатэгах (matatag)**, то есть тэгах HTML, которые предоставляют информацию о Web-странице. В отличие от большинства HTML-тэгов метатэги никак не влияют на внешний вид документа. Они содержат сведения об информационном наполнении Web-страницы и некоторые его ключевые слова.
Раньше некоторые администраторы сайтов нарушали нормальную работу машины поиска по ключевым словам, забив ключевыми словами всю свою Web-страницу или поместив в метатэги ключевые слова, не имеющие никакого отношения к содержимому их узла. Но разработчики поисковых служб нашли способ противодействия: либо службы вообще не сканируют метатэги, либо ставят на нижние позиции в своей классификации те узлы, которые используют ключевые слова, не имеющие отношения к их реальному содержимому.

Разработка новых технологий информационного поиска

Мы ежедневно путешествуем по Всемирной паутине в поисках нужной информации на Web-узлах, но, как правило, редко достигаем желаемого результата. Это связано с тем, что технологии поиска, к сожалению, не развиваются в соответствии с темпами роста масштабов Web.
Традиционная поисковая технология работает следующим образом: **пользователи вводят ключевые слова, затем поисковые службы просматривают Web-страницы на предмет наличия этих слов.** Такой подход неизбежно порождает ряд всем известных проблем.

Прежде всего, для успешного завершения поиска пользователю необходимо подобрать адекватные ключевые слова. Если слова выбраны общие или чересчур многозначные, то на выходе мы рискуем получить слишком много информации, не имеющей никакого отношения к интересующему нас предмету. Например, если введены ключевые слова «history of rock», результат поиска может содержать ссылки на страницы о популярной музыке, геологии или исторических курсах университета. В том случае, если ключевые слова выбраны неверно, пользователь получит совершенно ненужные ему ссылки, если вообще что-либо получит. «В сущности, происходит вот что: булевы методы поиска, изобретенные в 60-х и 70-х годах, истощили свой ресурс», – считает Кевин Вербах, выпускающий редактор информационного бюллетеня Release 1.0, посвященного новым коммуникационным и компьютерным технологиям.

Для разработки адекватных технологий поиска есть ряд важных экономических стимулов. Многие поисковые службы, в том числе AltaVista, Excite, Lycos и Yahoo, превращают свои Web-страницы в информационные порталы, из которых можно получить доступ к разнообразным службам, включая поисковые машины, электронные магазины, информацию о котировке акций, прогноз погоды, чаты и др. Компании заинтересованы в том, чтобы привлечь к своим порталам как можно больше пользователей, поскольку от этого напрямую зависит размер платы, которую они смогут назначить своим рекламодателям и партнерам за размещение информации на сайте.
В борьбе за пользователя службы поиска играют ключевую роль – для многих людей главной причиной первого обращения к порталу является необходимость найти ту или иную информацию. Более того, по данным Nielsen Media Research (компания, которая занимается измерением уровня используемости компьютеров и Internet), к поисковым службам обращается около 71% всех пользователей Internet.

Однако сейчас, как отмечается в [19], потребители информации считают современные поисковые машины примерно равными по своим возможностям. Так что, как правило, пользователи при выборе того или иного портала руководствуются иными причинами, нежели наличием определенной поисковой службы. Но если компания предлагает технологию поиска, которая существенно отличается от остальных, то вполне вероятно, что ее портал действительно привлечет большее количество пользователей. Поэтому сегодня так активно ведутся исследования в области новых технологий и методов поиска.

Проблемы

Размеры Web – это большое препятствие на пути к совершенствованию технологии поиска. Всего насчитывается около 350 млн. Web-страниц, а поисковая служба Alta Vista, которая является одной из самых информационно насыщенных, индексирует только 140 млн. страниц. При этом Web постоянно меняется – появляются новые

ресурсы, ликвидируются устаревшие страницы. По оценкам Research Institute компании NEC, более 5% результатов поиска с помощью одной известной поисковой службы составили неверные или уже несуществующие, «мертвые», ссылки.

На стадии изучения находятся несколько новых, усовершенствованных методов поиска в Web. В некоторых поисковых службах, например, увеличивают размеры индексов, пытаясь тем самым обеспечить пользователя более широким спектром ссылок на нужную ему информацию.

Среди проблем, которые отмечаются в литературе, можно выделить пять основных:

1) Человеческий фактор

Метод использования человеческого фактора (human annotation) предполагает поиск не по ключевым словам, а по реакции пользователей на результаты, полученные ими во время прежних обращений к механизму поиска. Сторонники этого подхода считают, что на основании результатов предыдущих сеансов поиска, а также информации о том, ссылки на какие страницы Web-мастер решает включить в свой сайт, можно лучше понять, какие именно сайты удовлетворят требованиям нового поиска. Они отмечают также, что этот метод не позволяет администратору того или иного Web-узла свободно манипулировать поисковыми службами путем подбора ключевых слов.

Однако, по мнению Вербаха из Release 1.0, такой подход снижает эффективность работы поисковой службы, поскольку он заставляет ее всегда реагировать на результаты прежних обращений и оставляет мало свободы для удовлетворения потребностей новых пользователей.

Служба поиска Direct Hit использует технологию под названием Popularity Engine, применяющую специальный алгоритм отслеживания пользователей в ходе выполнения поисковых запросов. Как отмечает Гари Галлисс, один из создателей Direct Hit, это отслеживание проводится анонимно и не позволяет установить соответствие между определенными IP-адресами и Web-страницами. Popularity Engine следит за тем, какие Web-страницы посещает пользователь, сколько времени он проводит на каждом узле и какие гиперссылки выбирает. Direct Hit использует эту информацию, чтобы оценить, насколько данный Web-узел соответствует поисковому запросу. По мнению Галлисса, благодаря этому методу пользователь становится редактором поиска.

В поисковой службе Google, придуманной аспирантами Стенфордского университета Серджем Брином и Ларри Пейджем, объединены принципы поиска по ключевым словам и метод ***human annotation***. Google использует собственный механизм сканирования Web-страниц Googlebot. *Вместо ключевых слов* ***Googlebot*** *ищет гиперссылки*. Для заданного предмета ведется поиск Web-страниц с гиперссылками на другие страницы, которые, по предположению Googlebot, соответствуют этому предмету. В основе Googlebot лежит метод сопоставления текстов и ряд других механизмов. Поисковая служба Google присваивает таким страницам первые места в своей классификации, и, с большой долей вероятности, именно они будут возвращены пользователю в ответ на поисковый запрос.

2) Быстрее, еще быстрее

Разработчики компании Fast Search&Transfer (www.fast.no) стараются максимально увеличить скорость работы своей поисковой службы (www.alltheweb.com), используя для этого несколько методов.

Благодаря масштабируемости архитектуры в этой поисковой системе среднее время отклика на сложный запрос не превышает одной секунды, тогда как в большинстве поисковых служб оно составляет от 4 до 4,5 секунд. Такой высокой производительности удается достичь благодаря нескольким факторам: быстрым алгоритмам индексации, большим массивам серверов, системе хранения и средствам межсоединения, а также программному обеспечению, эффективно использующему серверные возможности.

В Fast Search&Transfer считают, что масштабируемая архитектура позволит поисковой службе компании эффективно справиться как с ростом количества поисковых запросов, так и с постоянным увеличением числа Web-страниц. Индекс этой поисковой службы включает 80 млн. страниц и в ближайшем будущем должен вырасти до 200 млн. Если это произойдет, служба поиска Fast Search&Transfer будет иметь один из самых больших индексов.

3) Фильтрация

Поисковые службы iAtlas и Northern Light используют технологию фильтрации, которая позволяет получить результаты, наиболее близкие к предмету запроса. При обращении к службе поиска по ключевым словам пользователь заполняет электронную форму, в которой сообщает, что ему нужна, например, информация только по определенным отраслям промышленности или по конкретным регионам.

«При поиске нужно учитывать род занятий пользователя и его задачи, – считает Курт Монаш, исполнительный директор и один из основателей компании Elucidate Technologies, которая занимается разработкой различных программных продуктов, в том числе связанных с поисковыми службами. Это сделает поиск более точным.». Правда, если пользователь задаст слишком общие критерии фильтрации, он рискует потерять потенциально нужные результаты.

4) Естественный язык

Некоторые службы поиска уже имеют возможность обрабатывать запросы на естественном языке. Например, в Ask Jeeves пользователь, который собирается продать свой автомобиль, вместо одного или нескольких ключевых слов может ввести такой вопрос: «Сколько баксов я могу выручить за свою тачку?». Поисковая служба отошлет его на сайт, предоставляющий информацию о рыночных ценах на подержанные автомобили.

Механизмы поиска по запросам на естественном языке анализируют грамматическую структуру запроса и определяют его смысл, а затем используют результаты данного анализа для поиска по ключевым словам. Однако, как отмечает Джон Лафферти, профессор факультета компьютерных наук и Института языковых технологий университета Карнеги Меллон, применение естественного языка пока не имело большого успеха. По его мнению, технология поиска по запросам на естественном языке еще не в состоянии провести эффективный грамматический разбор запроса из-за недостаточной зрелости применяемых для этого алгоритмов.

5) Каталоги

Некоторые поисковые узлы представляют собой каталоги. Недавно компания Netscape Communications приобрела два проекта: NewHoo Community Directory Project и Open Directory (http://dmoz.org). В последнем на некоммерческой основе участвуют эксперты по различным областям знаний. В их задачу входит создание и сопровождение каталогов Web-сайтов, которые предоставляют информацию по интересующей данного эксперта предметной области. Ряд поисковых служб, в том числе HotBot и Lycos, приобрели патент на использование Open Directory.

Сторонники такого подхода заявляют, что благодаря привлечению экспертов-волонтеров проект сможет расширяться по мере роста Internet. Есть, правда, определенные сомнения в том, будет ли труд добровольных экспертов столь же эффективным, как создание каталогов специалистами, нанятыми и обученными компаниями-разработчиками поисковых служб.

Усовершенствованная технология поиска нужна и военным. Центр космических и военно-морских систем (Space and Naval Wafare Center) агентства DARPA вложил 2 млн. долл. в разработку **технологии классифицирующего поиска**, которая ведется в университете штата Миссисипи.

Что касается коммерческих поисковых служб, то они, скорее всего, будут стремиться занять каждая свою нишу и предоставлять информацию лишь на определенные темы и/или индексировать только определенное множество документов.

Будущее поисковых служб Web-порталов, по всей видимости, связано не с одним, а сразу с несколькими новыми методами поиска, которые сейчас осваивает передовой отряд поисковых машин.

4 Технологии электронного документооборота

4.1 Документы в СЭД. Понятие электронного документа

Документ — основная единица информации, и все существование системы документооборота посвящено хранению документа, его свойств и истории его жизни, а также собственно обеспечению его жизнедеятельности.

В свое время в Microsoft ввели для всех файлов, сохраняемых своими офисными приложениями, стандартную шапку, в которой задавались бы свойства, такие, как заголовок и автор. Однако на практике эта возможность не прижилась; редкий документ содержит что-то осмысленное в разделе свойств. Не получил широкого применения и механизм сохранения различных версий документа в одном файле, реализованный в MS Office 2000. Причина проста: без системных организационных мер и единства подхода наличие частных механизмов в конкретных продуктах не дает результата. Очевидно, что такие средства, как описание атрибутов документа, сохранение его версий и т.д., должны быть поддержаны в рамках информационной среды безотносительно к типу документа и приложению, с помощью которого этот документ создан.

Электронный документ — логическая единица СЭД. Способ его хранения зависит от того, как с ним удобнее работать. Ваш документ может состоять из текста, чертежей, рисунков и таблиц. Механизм COM позволяет организовать в одном файле подобие файловой системы, состоящей из аналогов папок и файлов. Этот механизм используется, например, в Word для того, чтобы обеспечить возможность вставки в текст объектов, созданных другими приложениями. Но это не всегда удобно; проще и практичнее хранить все части документа в отдельных файлах, каждый из которых редактируется своей программой. В большинстве СЭД отдельный документ может физически состоять из набора файлов.

4.2 Жизненный цикл документа в СЭД

Рождение

В один прекрасный день на свет появляется новый документ. Этот праздничный для документа день в современных системах хранения фиксируется и хранится, однако в стихии обычных файловых систем можно легко потерять предысторию файла. Достаточно, например, просто переписать на его место новый файл с тем же именем, и никто уже не вспомнит о предшественнике, не говоря уже о том, что файл, полученный из Сети, вообще не имеет «ни рода ни племени»: датой его создания всегда будет момент, когда он оказался на жестком диске вашего компьютера. В СЭД документ не может возникнуть, если у него нет «паспорта» — карточки учета, которая может быть разной для различных типов документов: документ нельзя просто так удалить, переименовать или переписать поверх. Все эти действия протоколируются, их следы останутся. В случае необходимости, система сохранит все предыдущие варианты и даже удаленные документы. Все действия, которые могут быть проделаны над документами, определяются правами, которые даны пользователям, что позволяет задать стратегию работы с документами.

Становление

Любой документ непременно проходит этап своей жизни, который называется «черновиком» — неокрепший документ в этот период переходит из рук в руки, его меняют и переделывают. Качество результирующего документа во многом зависит от

того, насколько успешно и организованно он прошел через эту «черную» полосу своей жизни. В СЭД для организации коллективной работы над документом применяется техника блокировки редактируемых документов («check-out, check-in»). Система берет на себя заботу о том, чтобы в каждый момент документ мог редактировать только один человек. Благодаря этому механизму исключается возможность того, что два сотрудника создадут у себя две локальные копии документа и одновременно внесут в него изменения. Когда в СЭД один из сотрудников забирает документ для редактирования, остальные увидят это и не смогут изменить документ до тех пор, пока первый не вернул его обратно. При этом возвращенному документу автоматически присваивается новый номер подверсии. Прежняя подверсия документа сохраняется, ее можно открыть, посмотреть и редактировать. Все действия всех участников процесса документируются, поэтому никакой путаницы не возникнет.

Публикация

Это день, ради которого документ создавался и доводился до ума. Публикация — это момент, после которого документ «умирает». Благодаря наличию механизма публикации вы можете быть уверены, что всегда будете иметь в электронном виде в точности то же самое, что было, например, подписано, или отправлено в печать, или выслано партнеру. А что если потребовалось что-то изменить в документе после публикации? Для этого на основе опубликованной версии создается новый вариант документа и начинается новый жизненный цикл. В различных системах эта функция поддержана по-разному. Где-то создается полностью новый документ, а где-то — просто новая версия.

Архивирование

После публикации документ отправляется в электронный архив, где ему предстоит пробыть столько времени, сколько это предусмотрено распорядком вашей организации. Есть документы, которые хранятся вечно. Есть документы, которые нужно хранить несколько дней. Создание архива — задача непростая, зависящая от потребностей организации. Например, документы, к которым часто обращаются, нужно хранить на быстрых носителях, а неактуальные документы, которые редко используются, можно положить на менее дорогие, но медленные носители. Для решения таких задач применяются технологии управления иерархическим хранением HSM (Hierarchical Storage Management), которые создают из всевозможных разнородных средств хранения «виртуальную файловую систему» сколь угодно большого размера, при этом управляя переносом информации с одного носителя на другой. Базовые средства HSM были встроены в Windows 2000, однако существуют и другие технологии, обеспечивающие более сложную и эффективную функциональность. Таковыми являются, например, продукты серии DiskXtender компании Legato Systems, Tivoli Storage Manager, Veritas Storage Migrator и др.

Поддержка жизненного цикла в различных СЭД

Практически все современные системы электронного документооборота поддерживают все этапы жизненного цикла документа. Вопрос только в том, насколько полна эта поддержка. Перечислять СЭД, в которых та или иная функциональность не поддержана, задача неблагодарная: к моменту появления этой статьи может выйти новая версия, где эта функция уже имеется. Часть систем не поддерживает механизм блокировки редактируемых документов, что делает коллективную работу с документами невозможной. Есть системы, ориентированные на делопроизводство, и в них не реализовано эффективное хранение документов, а актуально выполнение всех процедур работы с документами, регламентированных существующими нормами. А

сами документы могут лежать в папках в шкафу. Некоторые системы ориентированы на эффективную поддержку движения электронных документов внутри структуры, но при этом не имеют собственного электронного архива — хранилище, реализованное в этих системах, предназначено только для оперативного хранения документов в процессе их жизненного цикла. После опубликования документы выходят из системы и возвращаются в типовую для них среду хранения, например, в файловую систему. К такой системе можно «пристыковать» электронный архив, где сохраняется документ вместе с его историей и сопроводительной карточкой. Например, компания «Электронные Офисные Системы» предлагает состыковать свой продукт «Дело» с электронным архивом, созданным компанией на основе сервера «Кодекс-Intranet/Internet». Тот же самый сервер компании «Кодекс» применяет и компания «Гранит-Центр» в качестве электронного архива к своей системе «ГранДок». Прежние версии обеих систем поставлялись без электронного архива.

4.3 Компоненты СЭД. Место СЭД в информационной системе предприятия

Все СЭД содержат обязательные типовые компоненты: хранилище карточек (атрибутов) документов; хранилище документов; компоненты, осуществляющие бизнес-логику системы.

Хранилище атрибутов документов

Хранилище атрибутов документов предназначено для хранения «карточки» — набора полей, характеризующих документ. Обычно в СЭД имеется понятие типа документов (например, договор, спецификация, письмо и т.д.) и для каждого типа заводится своя собственная карточка. Карточки разных типов имеют обязательные поля, общие для всех документов, и специальные поля, относящиеся к документам данного типа. Например, общими полями может быть уникальный номер документа, его название, автор, дата создания. При этом документы типа «договор» могут содержать такие поля, как дата подписания, срок действия, сумма договора. Типы документов, в свою очередь, могут иметь подтипы, имеющие общий набор полей, который они наследуют от основного типа, и при этом дополнительные поля, уникальные для подтипа. Наиболее развитая система управления документами может поддерживать большую вложенность таких подтипов. Типизация документов, выстраивание их иерархии, и проектирование карточек для них является одним из наиболее важных этапов в процессе внедрения СЭД.

Кроме понятия типа документов, возможно присваивание документам категорий, причем один документ может принадлежать одновременно к нескольким категориям. Категории могут быть выстроены в дерево категорий. Например, можно иметь категорию «Юридические документы» с подкатегориями «Законы», «Договоры», «Приказы» и т.д. При этом можно иметь параллельную структуру по отделам, например, категорию «Документы отдела продаж», а в ней подкатегории «Договоры на продажу», «Счета» и т.д. Договор на продажу может быть одновременно отнесен к подкатегориям «Договоры» и «Договоры на продажу», относящимся к разным ветвям в иерархии категорий. Таким образом, появляется возможность поиска документа в таком дереве на основе его классификации, причем один и тот же физический документ может встречаться любое число раз в разных узлах этой иерархии.

Для организации хранилища карточек возможны три варианта решения: использование собственного хранилища, стандартной СУБД или средств среды, на основе которой построена СУБД.

Собственное хранилище атрибутов документов позволяет оптимизировать его под задачу хранения карточек, гибко реализовать функции создания сложных карточек (имеющих, например, большую вложенность типов), а также использовать эффективные алгоритмы поиска информации в карточках. К системам, имеющим собственное хранилище, относятся, например, Documentum, «Евфрат» компании Cognitive Technologies и «Гарант-Офис» компании «Гарант Интернейшнл». Очевидным недостатком такого подхода является невозможность использовать стандартные ресурсы имеющейся информационной среды, а также зависимость критически важной информации от поставщика СЭД. В случае, если вы используете стандартную СУБД, всегда есть возможность миграции данных на СУБД от другого поставщика. Здесь же выбор жестче — придется отказаться от использования конкретной СЭД вообще, а миграция данных из одной СЭД в другую на порядок сложнее, чем в случае СУБД.

При использовании стандартных СУБД для хранения документов данная проблема решается. К такого рода системам относятся, например, системы «Дело» от ЭОС, «1С:Архив» и DocsFusion компании Hummingbird. Однако такой подход имеет свои слабые стороны — реляционная модель, реализованная в большинстве СУБД, не удобна для модели данных, используемой в СЭД. Достаточно сложно обеспечить необходимую гибкость при создании карточек документов, особенно, если нужна сложная структура. Разработчики СЭД при этом оказываются перед дилеммой: разработать простую, но эффективную структуру хранения данных, при этом отказаться от гибкости при создании карточек, либо иметь громоздкую структуру, которая обеспечивает необходимую гибкость за счет эффективности, прозрачности и надежности работы системы. Вторая неприятная проблема состоит в том, что при использовании внешней СУБД возникают некоторые трудности как при миграции с одной версии СЭД на другую, так и при переходе с одной версии СУБД на другую. Чаще всего такая ситуация приводит к определенному консерватизму пользователей в вопросе перехода на новые версии.

Если СЭД построена на основе какой-либо информационной среды, то грех не воспользоваться ее ресурсами. Большинство систем такого типа, популярных в России, построено на основе Lotus Notes/Domino. Это позволяет использовать все механизмы, заложенные в эту среду, в том числе средства резервного копирования, репликации, поиска и т.д. Проблемы такого подхода лежат в самой необходимости наличия определенной среды для работы системы управления документами, а также в тех ограничениях, которые накладывает конкретная среда на структуру ее баз данных.

Хранилище самих документов

Для реализации хранилища документов, опять же, существует два подхода: хранение в файловой системе или в специализированном хранилище СЭД. С точки зрения прагматичного пользователя между этими подходами, если оценивать их в целом, большой разницы нет. Но некоторые особенности все же имеются.

Хранение в файловой системе понижает степень безопасности при разграничении доступа, так как файловая система может не поддерживать ту модель безопасности, которая реализована в самой СЭД. Поэтому приходится наделять СЭД своими правами доступа, так что файлы, сохраненные ею, будут недоступны ни одному из пользователей напрямую. А СЭД поддерживает свою систему списка пользователей с правами доступа, организуя доступ к файлам через эти права. Система доступа при

этом становится сложной в сопровождении и не вполне безукоризненной с точки зрения информационной безопасности. Для обеспечения дополнительной надежности часто используется шифрование файлов при хранении. Кроме того, практически все СЭД используют случайное именование файлов, что сильно усложняет поиск нужного файла при попытке доступа в обход системы. Надо сказать, что большинство СЭД осуществляют хранение файлов в файловой системе.

При работе с файловой системой большинство СЭД требуют перемещения файлов в специально организованные каталоги. Но есть и исключения. Например, системы «Евфрат» и Microsoft SharePoint позволяют регистрировать в системе файлы, не требуя их физического перемещения в хранилище. Понятно, что такой подход опасен с точки зрения целостности данных, но зато очень удобен в «переходный период» внедрения СЭД.

Системы, имеющие свое собственное хранилище файлов или использующие хранилище среды, на основе которой построены (например, Lotus Notes/Domino или Microsoft Exchange), могут гарантировать более эффективное управление доступом к документам и более надежное решение проблемы разграничения доступа. Так устроены, например, Documentum и системы на основе Lotus Notes («БОСС-Референт», CompanyMedia). Но при этом возникают вопросы, связанные с целостностью данных, наличием эффективных средств резервного копирования и интеграцией со средствами архивного хранения на медленных носителях. В большинстве систем они так или иначе решены, однако можно пользоваться только инструментами, доступными в самой системе, в то время как в случае файлового хранения вы всегда имеете выбор.

Бизнес-уровень

На уровне бизнес-логики обнаруживаются существенные различия между разными СЭД. Собственно, все описанные компоненты, хотя и могут быть устроены по-разному, отличаться степенью сложности, но при этом функционально аналогичны. Бизнес-логика же различных систем может отличаться кардинально, и это как раз то, что должно интересовать более всего при ознакомлении с системой электронного документооборота. Можно выделить ряд фундаментальных компонентов, из которых, как из кубиков, складывается функциональность любой СЭД.

Управление документами в хранилище. Включает процедуры добавления и изъятия документов, сохранения версий, передачи на хранение в архив, поддержания архива и т.д.
Поиск документов. Состоит из поиска по атрибутам, визуального поиска по различным деревьям, в которые уложены документы, поиска по полному тексту, смыслового поиска и т.д.
Маршрутизация и контроль исполнения. Обеспечивает доставку документов в рамках бизнес-процедур в организации. Собственно, от этой функциональности и пошел термин "электронный документооборот". Маршруты документов могут быть гибкими и жесткими. В случае гибкой маршрутизации следующий получатель документа определяется сотрудником, в ведении которого документ находится в данный момент. В случае жесткой маршрутизации путь прохождения документов определяется заранее на основе некоторой логики. В реальной жизни применяется "смесь" из этих двух подходов: для одних документов и структур в организации уместнее жесткая маршрутизация, для других гибкая. Функция маршрутизации присутствует не во всех СЭД. Обычно, чтобы не путаться, системы без средств маршрутизации называют электронными архивами. Контроль исполнения является неотъемлемой частью маршрутизации. Если у документа "появились ноги", то нужен контроль того, куда он идет и где сейчас находится. Фактически, маршрут определяется в терминах пути

прохождения и временных интервалов на исполнение документа каждым из участников процесса прохождения. Под исполнением документа подразумевается выполнение действия, связанного с документом, каждым из участников в рамках его должностных полномочий. Проще говоря, кому-то нужно его просто прочитать, а кто-то, возможно, будет уволен.
Отчеты. Служат аналогом конторских журналов учета документов. Используя различные отчеты, можно посмотреть, например, общее время, потраченное сотрудниками на работу над конкретным документом, скорость прохождения документов по подразделениям и т.д. Отчеты - отличный материал для принятия управленческих решений.
Администрирование.Поддержика работы самой системы,настройки ее параметров и т.д.

«Правильная» СЭД

С точки зрения технологий системы электронного документооборота мало отличаются от любых других распределенных информационных систем. По принципу построения архитектуры они пытаются по возможности следовать современным тенденциям и требованиям рынка. Сейчас наиболее популярна концепция открытой среды, максимально подверженной адаптации под конкретные нужды, но при этом несложной в установке и сопровождении, с «тонким» клиентом и выделенным сервером приложений, по возможности многоплатформным. Все существующие системы в той или иной мере приближаются к этому идеалу. Однако еще достаточно распространены системы, основанные на полнофункциональном клиенте, привязанном к конкретной платформе. Иногда в этих случаях для удаленного доступа предлагается отдельный Web-клиент с ограниченной функциональностью. Например, в системе «ГранДок» компании «Гранит-центр», полная функциональность доступна только при использовании клиентского приложения, но при этом пользователь может осуществлять поиск и просмотр документов, находящихся в архиве, с помощью обычного браузера.

Важно отметить, что описанный «идеал» не является еще тем ориентиром, который может стать приоритетом при выборе системы. Это только один из факторов. Если какая-то система вас устраивает по соотношению цены и функциональности, то вовсе не обязательно, чтобы она полностью соответствовала последним веяниям в области построения информационных систем. На переднем крае все быстро меняется, и все новое завтра станет старым. Проверенные надежные решения зачастую ничем не уступают системам, построенным по последнему слову технологии.

Место СЭД в информационной системе предприятия

Основные функции СЭД: обеспечение управляемости и прозрачности деятельности предприятия, а также накопление знаний и управление знаниями. В современном мире эти две задачи становятся все более критическими. Например, в себестоимости автомобиля «Мерседес» лишь 30% — непосредственные издержки производства, а остальное — компенсация стоимости разработки автомобиля, т. е. стоимости деятельности инженеров и управленцев, поэтому в оптимизации их деятельности и лежит основной ресурс снижения себестоимости.

СЭД и другие

Степень эффективности использования СЭД определяется тем, насколько документы (неструктурированная информация) определяют информационное наполнение деятельности предприятия. Очевидно, например, что для чисто торговой организации основное информационное наполнение — это структурированные данные,

заключающиеся в базах данных. Возможно, такой организации и нужно хранить договоры, но вряд ли дело дойдет до внедрения СЭД. Однако если торговая организация дорастет до торгового монстра с сетью магазинов в десятках городов и сложной логистикой, собственным производством полуфабрикатов, то рано или поздно придется подумать о внедрении системы ERP. На следующем этапе количество оптовых покупателей и крупных заказчиков может вырасти до таких масштабов, что придется подумать о внедрении CRM. И только если при этом аппарат управления разрастется до сотни человек, появятся параллельные непрофильные проекты, возникнут задачи диверсификации, встанет задача внедрения СЭД. При этом какие-то системы, возможно, придется интегрировать, чтобы система CRM имела ссылки на письма, договоры и на копии входящих заказов, которые хранятся в СЭД.

В некоторых случаях интеграция этих систем еще более тесная — СЭД может служить интегрирующим транспортом для передачи документов между системами, которые их порождают, и системами, которые их потребляют, в случае, когда прямая связь на уровне структурированных данных между этими системами не нужна. Предположим, предприятие имеет системы CRM и ERP, причем требуется, чтобы в CRM фиксировались ежеквартальные отчеты из ERP о поставках товара конкретному клиенту, дополненные, возможно, комментариями экспертов. Понятно, что такие отчеты удобнее всего хранить в СЭД. Благодаря интеграции ERP и СЭД документ будет автоматически создан и сохранен. Благодаря интеграции СЭД и CRM возможно автоматическое прикрепление документа к карточке конкретного клиента. И все эти операции могут происходить автоматически. (Подчеркнем, что приведенный пример является чисто умозрительным и на самом деле может не иметь практического смысла; интеграция любых информационных систем имеет смысл только тогда, когда четко понятна ее цель.)

СЭД и управление знаниями

Задачу управления знаниями на сегодняшний день в полной мере решенной назвать нельзя. Утверждение, что СЭД эффективно решают эту задачу, является некой натяжкой: СЭД лишь позволяют хранить информацию и представлять ее в виде, удобном для анализа. К сожалению, имеющиеся средства управления знаниями малоэффективны. Проблема в том, что применяемые сегодня алгоритмы работы с текстовыми данными, основанные на статистических методах, являются слишком грубым инструментом. Будущее за системами, которые смогут содержательно анализировать смысл текста. Пока таких систем нет, об управлении знаниями в системах СЭД можно говорить только условно.

Тенденции

Очевидно, что функциональность систем управления документами в части решения вопросов управления практически полностью удовлетворяет сегодняшние запросы, и здесь особого развития в ближайшие годы не будет. Основное направление развития систем документооборота — повышение эффективности поиска информации, интеграция со средствами публикации информации в сетях, автоматическая сортировка и рубрикация документов.

Развитие систем управления документами получит второе дыхание с появлением средств, позволяющих осуществлять смысловой поиск информации и интеллектуальное автоматическое реферирование текстов на основе смысла. К сожалению, пока никто не может сказать, как быстро такие технологии станут коммерчески доступными.

Как утверждают эксперты, в плане внедрения систем электронного документооборота мы отстаем от стран Западной Европы примерно на 5 лет. Западный опыт показывает, что при массовом внедрении возникает спрос на весь спектр продуктов — от самых простых до сложных, распределенных и интегрированных решений. Поэтому у нас, похоже, в ближайшее время будет в большей степени превалировать тема внедрения уже имеющихся систем, нежели их дальнейшее развитие.

4.4 Типовые требования к СЭД

Если следовать букве стандарта на составление технического задания, требования, которые типовой пользователь может предъявить к типовой системе электронного документооборота, можно описать следующим образом.

Система электронного документооборота должна:
- обеспечивать надежное хранение документов и их описаний;
- обеспечивать жизненный цикл документа (его создание, хранение версий, публикация, блокировка доступа к изъятому документу, передача документа для хранения в архиве);
- допускать задание пользователем различных типов документов, создания и редактирования карточек для них;
- поддерживать иерархию категорий для эффективного поиска документа;
- осуществлять поиск документов на основе информации из карточки, а также полного текста;
- обеспечивать разделение доступа к документам на уровне отдельных пользователей, по ролевому принципу, и на основе иерархической структуры организации;
- поддерживать технологию HSM;
- протоколировать все события, связанные с работой пользователей и самой системы; необходимо наличие развитых средств администрирования;
- поддерживать удаленный доступ к информации.

Продвинутые системы должны поддерживать:
- кластерные технологии для обеспечения бесперебойной работы;
- территориально распределенные организации;
- алгоритмы шифрования при хранении и передаче данных;
- цифровую подпись.

Требования к архитектуре:
- наличие выделенного сервера приложений;
- наличие тонкого клиента; поддержка доступа к документам с использованием браузера.
- многоплатформность для обеспечения масштабируемости.

Требования к открытости и интеграции с другими системами:
- интеграция со средствами потокового ввода документов;
- интеграция с офисными приложениями;
- интеграция с электронной почтой;
- наличие развитого программного интерфейса (API);
- интеграция со стандартными службами каталогов (к примеру, LDAP) для ведения и синхронизации списка пользователей системы;
- возможность адаптации пользовательского интерфейса под конкретные задачи;

- возможность дополнения системы собственными специализированными компонентами.

В случае использования внешней базы данных для хранения атрибутов документов необходимо наличие подробного описания структуры данных и средств работы с разными СУБД.

Стандарты для систем управления документами

Стандартизация в области систем управления документами в основном заключается в выработке спецификаций взаимодействия систем от различных производителей, а также внешних приложений. Стандартизируются как сами протоколы, так и форматы данных, передаваемых между системами. На данный момент наиболее популярным универсальным стандартом взаимодействия с внешними приложениями (офисными приложениями, средствами потокового ввода) стал ODMA (http://odma.info). Этот стандарт существует с 1994 года, и его поддерживают многие производители ПО. Однако, как все универсальное, ODMA содержит определенные ограничения и всегда оказывается «запасным» вариантом взаимодействия с СЭД, когда, по какой-то причине, нет возможности реализовать более полноценную интеграцию. К примеру, несмотря на то, что начиная с версии Office 97 в продуктах Word и PowerPoint поддерживается ODMA, практически все производители СЭД поставляют специальные макросы для интеграции с MS Office.

Правда, есть случаи, когда протокол ODMA оказывается вполне эффективным. К примеру, система ABBYY FineReader, начиная с версии 4.0, поддерживает ODMA, что позволяет пользователю, не обращаясь к производителю СЭД или к услугам интегратора, вводить бумажные документы в хранилище систем, поддерживающих этот протокол. К сожалению, ODMA не затрагивает вопросов взаимодействия между различными СЭД, однако другие стандарты имеют существенно меньшее применение.

Литература

1. А. Пахчанян. Технологии электронного документооборота.Открытые системы, 2002, №10.

4.5 Технический документооборот и его особенности

Любой документооборот является неотъемлемой частью системы управления документацией. Создание такой системы является одной из сложнейших задач автоматизации, поскольку документооборот, и особенно технический, является глобальной системой, объединяющей бизнес- процессы и структурные подразделения как внутри предприятий, так и при совместной деятельности предприятий. Реклама выгод от внедрения систем электронного документооборота (ЭТДО) редко отличается разнообразием, и это лишний раз служит доказательством того, что документооборот является общесистемной дисциплиной, объединяющей различные области создания, обработки и управления техническими документами и данными.

Напротив, реклама самих систем электронного документооборота и особенно систем технического документооборота всегда пытается показать их некую уникальность, которая обычно заключается в том, что программное обеспечение управления ЭТДО интегрируется с теми или иными средствами управления или приложениями обработки прикладных данных, в учете специфики российского производителя, российских стандартов и т.п.

Однако любое конкурентное преимущество хорошо лишь тогда, когда оно может быть воплощено, а в идеале — гарантированно окупится и приносит прибыль. Чем «круче» изменения, предлагаемые информационными технологиями, тем большие изменения должны произойти в самой системе организации и управления бизнес-процессами, тем больше проблем, связанных с человеческим фактором, необходимо решить. Ведь системы управления документооборотом охватывают сотни и тысячи рабочих мест из разных предприятий и их подразделений. Кроме того, любой программный комплекс должен органично вписаться в программно-технический комплекс существующей информационной системы. Попробуем рассмотреть некоторые популярные мифы о выгодах, получаемых от внедрения ПО управления ЭТДО.

Выгоды от внедрения системы управления ЭТДО: мифы и реальность

Миф 1. Быстрый поиск документов — не надо затрачивать часы на поиск необходимого документа.

Этот миф порожден огромной скоростью обработки электронной информации вычислительной техникой, легкостью ее создания, тиражирования и перемещения. Однако для эффективного поиска электронных документов базисом являются организационные и административные технологии, игнорирование или недооценка которых (обычная российская практика), сводит на нет преимущества автоматизированного поиска. Управление наименованием документов, размещением, правильная идентификация документа и его местоположения, знание того, как правильно это сделать — вот главные условия для быстрого поиска документа. Например, при формировании запроса к компьютерной системе достаточно ошибиться в одном символе имени документа (или в маске имени), или указать неправильную область поиска — и результат поиска будет нулевым. Еще хуже, если само наименование документа содержит орфографическую или синтаксическую ошибку. Технологии нечеткого поиска могут частично помочь, но они далеко не всегда обеспечивают приемлемые результаты поиска и не так просты в эксплуатации. Кроме того, они нередко порождают иллюзию, что правильно знать язык теперь вовсе необязательно. Так же влияет на результаты поиска морфология и структура наименования. Это случай, когда от перемены мест слагаемых результат меняется. Например, «Программное обеспечение системы ЭТДО» и «Система ЭТДО.

Программное обеспечение» — для компьютерной системы являются совершенно разными.

Легкость модификации, тиражирования и перемещения электронных документов создают опасность другого рода — избыточность версий документа, особенно черновых. Статистика показывает, что на практике электронные документы «плодятся» в семь раз больше, чем их бумажные собратья. Важность размещения документов и сегодня очень редко воспринимается серьезно даже администраторами информационных систем. ПО управления ЭТДО не может автоматически управлять классификацией, каталогизацией и размещением документов (эти функции часто подменяют функциями присвоения класса, каталога документу и помещения документа в требуемое место хранилища).

Миф 2. Надежное хранение документов — значительно снижается вероятность потери документа или доступа к нему лиц, не имеющих на это права.
Хранение имеет смысл только в контексте того, что пользователь, передавший документ на хранение, получит его по запросу. Если вы точно знаете, что документ находится и надежно хранится в хранилище, а найти или получить его не можете, то какая вам от этого польза? Однако при неправильной организации системы хранения электронных документов, последние намного уязвимее, чем системы хранения традиционной документации. Вот несколько причин этого.

1. Создание инфраструктуры для электронных хранилищ и работы с ними требует больших капиталовложений и сильно зависит от топологии инфраструктуры зданий и помещений.
2. Электронные хранилища более климатически и энергозависимы.
3. Электронные системы намного более подвержены возможности несанкционированного доступа к данным, искажению и уничтожению этих данных; скорость такого разрушительного воздействия может быть куда быстрее, чем скорость распространения огня (наиболее страшная угроза для любых архивов).
4. Надежное хранение документов также предполагает их сохранность в случае возникновения чрезвычайных ситуаций (пожар, стихийные бедствия и т.д.).

Система мероприятий по восстановлению электронного архива в случае его существенного повреждения или гибели при катастрофах намного более сложна, чем архивов традиционной документации, - и более уязвима, так как очень немногие предприятия задумываются о такой угрозе и мероприятиях по восстановлению архивов.

5. Электронные хранилища в критических ситуациях более человекозависимы, так как "пожарниками" часто выступают один-два системных администратора, которых в отличие от настоящих пожарных не заменят их коллеги с другого предприятия. Причина банальна - подавляющее число решений и специфика создания, организации, конфигурирования и настройки хранилищ не документируются надлежащим образом или не документируются вообще.

Еще больше требований возникает при создании долговременных хранилищ ЭТДО, например, электронных архивов. Получение документа из электронного хранилища еще не гарантирует, что пользователь может с ним работать. Ведь в отличие от бумажных электронные документы требуют различных программных средств их обработки и визуализации. Эти средства должны быть правильно сконфигурированы, причем эта конфигурация может быть связана с конфигурацией системного программного обеспечения, которое в свою очередь нередко зависит от особенностей «железа». Это в значительной степени относится к техническим документам, которые, в отличие от офисных, имеют теснейшую связь с системами PDM и САПР.

Хранилище электронной технической документации должно быть рассчитано на:

- электронные технические документы (ЭТД);

- связанные с ЭТД электронные технические данные;
- двухмерные и трехмерные модели, с учетом особенностей хранения части данных в базах данных управляющих программ (САПР, PDM), обеспечивающих работу с этими моделями;
- ПО, на котором велась обработка документов и данных, хранящихся в архиве;
- данные о конфигурации программного обеспечения и вычислительной технике, на которых велась обработка.

Таким образом, надежное хранение электронной документации без создания соответствующей системы управления хранением, учитывающей вышесказанное — не более чем очередной миф.

Миф 3. Внедрение ПО управления ЭТДО резко увеличивает эффективность управления деловыми процессами.

Зарубежная статистика приводит цифры об увеличении производительности при автоматизации управления деловыми процессами (УДП) на 30-40%. Но причины этого кроются не только в возможностях, предоставляемых компьютерными технологиями, но и в изменении технологии выполнения этих процедур и в дисциплине труда.

Во-первых, переход к АСУ ЭТДО требует организационной и управленческой зрелости предприятия, как исходного состояния для эффективной автоматизации УДП. Нашим предприятиям до такого состояния пока далеко: даже не прибегая к автоматизации УДП, только за счет организационных изменений можно поднять эффективность УДП не на 30-40%, а на все 100-200% и более.

Во-вторых, проблема сосуществования электронного и бумажного документооборота требует разработки и внедрения соответствующей системы правил УДП, доработки стандартов ЕСКД, ЕСТД и других серий. Система правил УДП в техническом документообороте намного разнообразнее и сложнее, чем в других видах (организационно-распорядительный, банковский и др.).

В-третьих, эффективное автоматизированное УДП в ЭТДО возможно только при сквозной интеграции потоков работ и документов между всеми приложениями, осуществляющими УДП. Это сама по себе чрезвычайно сложная задача, процесс стандартизации которой начался совсем недавно (первые стандарты появились в 1999 году) и активно продолжается коалицией Workflow Management Coalition. Эти стандарты пока реализованы в весьма небольшом количестве продуктов таких известных разработчиков, как SAP AG, BAAN, IBM и некоторых других.

К сожалению, наши разработчики средств управления документооборотом этим похвастаться не могут, но зато многие декларируют «соответствие требованиям стандартов и принципов создания открытых систем». Максимум, что предлагается нашими разработчиками — точечная интеграция между отдельными средствами УДП, основанная на частных нестандартных спецификациях обмена. Отличительная особенность этих спецификаций — опора на текущие программные реализации продуктов де-факто (т.е. в лучшем случае — на внутрифирменные стандарты производителей, в худшем — на логику разработчиков программного обеспечения). Решения на базе таких интеграционных решений обычно «сыпятся» при переходе на новую версию продукта или при изменении опорных технологий, и на все сто процентов зависят от разработчика.

Миф 4. Содержание бумажных архивов обходится предприятиям на 80% дороже, нежели архивов электронных.

Может быть, если дело касается архивов нетехнических документов. Что же касается архивов технических документов, то говорить о стоимости их содержания пока что рано. Сначала неплохо было бы просчитать, во что выльются вложения в разовые работы: обследование предприятия, проектирование и создание программно-технического комплекса, оцифровка бумажного архива, разработка системы

управления применением ЭЦП, разработка новой технологии работы и регламентов работы с ЭТД, гармонизация с системой традиционного документооборота, обучение персонала, переход на новую технологию. Добавьте сюда специфику электронных архивов, описанную во втором мифе, во сколько обходятся штатные и внештатные задержки при сопровождении программно-технического комплекса с его весьма частыми изменениями...

Миф 5. Внедрение ПО документооборота делает деятельность сотрудников более прозрачной, а значит и более управляемой.

Управляемость есть мера достижимости поставленных целей за счет влияния руководителей на исполнителей, на изменение бизнес-технологий и бизнес-процессов, и уж никак не является следствием прозрачности деятельности персонала и не зависит от электронных систем. Каждый уровень управления в идеале должен заниматься своими делами, а не разбираться, почему не работает более нижний уровень (хотя есть немало руководителей, с удовольствием подменяющих выполнение своих основных задач выполнением задач нижнего уровня).

Практика убедительно доказывает, что руководство все меньше и меньше имеет возможностей для обдумывания и принятия даже стратегических решений. Это еще раз доказывает, что управление людьми будет развиваться в сторону «черного ящика», когда важен результат, а не процесс. Что же касается бизнес-технологий, то грамотное внедрение ЭТДО в совокупности с созданием и внедрением автоматизированной системы управления бизнес-процессами действительно может дать эффект прозрачности бизнеса, т. е. возможность видеть бизнес-технологии и бизнес-процессы на разных уровнях их представления, эффективно управлять ими и изменять их. Это чрезвычайно важная задача, так как технологии проектирования и создания изделий в электронном виде только начинают создаваться.

Проблемы создания технического документооборота

Исторически сложилось так, что технический документооборот и его автоматизация никогда не рассматривался ни как самостоятельная дисциплина, ни как комплексная система управления. Были и есть стандарты ЕСКД, ЕСТД и других групп, отраслевые стандарты, в которых технологические, организационные и процедурные вопросы, связанные с технической документацией и документооборотом достаточно хорошо отработаны и проверены десятилетиями. Сама же система управления была «человеко-ориентированной» и что чрезвычайно важно — интеллектуальной (т.к. средством всевозможных операций над документами, средством информационного взаимодействия, средством разрешения коллизий была и есть единственная интеллектуальная унифицированная природой «вычислительная машина» — человек). Это позволяло решать задачи управления документооборотом фактически в фоновом режиме при сохранении слабо формализованного информационного взаимодействия его участников. Такое положение дел долгое время оставалось неизменным во время автоматизации систем создания и подготовки инженерных данных. Это подтверждает и тот факт, что категории Docflow и Workflow появились не в виде самостоятельных систем, а как вспомогательные модули к различным системам PDM, ERP и САПР и бизнес-приложениям.

Такая потеря самостоятельности ЭТДО и сегодня играет злую шутку с отечественными предприятиями: вместо создания современных систем управления документами и деловыми процессами предприятия по-прежнему концентрируют свое внимание на задачах создания и подготовки содержимого документов (технических данных). Методы и средства управления документооборотом фактически оказались без изменений. Таким образом, задача автоматизации управления ЭТДО фактически является придатком систем создания и обработки электронных технических данных. В результате не используется один из самых важных инструментов, способных повлиять

на бизнес-технологии, в том числе на переход к процессно-ориентированной организации бизнеса.

Стандартов на электронные технические документы (ЭТД) ни в России, ни в Украине пока не существует. Да и сама система стандартизации электронного документооборота строится весьма нелогично: разрабатываются частные системы стандартов, охватывающие некоторые частные области его применения (например, стандарты STEP, стандарты серии ГОСТ Р ИСО 10303 и др.), в то время, как нет основополагающих общесистемных стандартов на электронный документ, системы управления электронной документацией и системы управления электронными данными. Таким образом, даже в таком системном деле, как стандартизация, мы продолжаем «славное» дело лоскутных решений.

В настоящее время разработку стандартов на систему управления электронной технической документацией в России ведет научно-исследовательский центр CALS-технологий «Прикладная логистика» (www.cals.ru). При разработке проекта стандарта на систему управления электронной технической документацией немало сил было приложено к тому, чтобы решить, казалось бы, простую задачу: развести по разным углам электронный технический документ и электронные технические данные (ЭТДА). Уж слишком велик был соблазн сделать единую систему стандартов на систему управления ЭТДО и ЭТДА, и весьма непросто провести различие между их управлением и обработкой с точки зрения ИТ.

Почему это так важно? ЭТД является интегрирующим объектом, который позволяет провести ясную и обоснованную границу между собственно электронным документом и его контентом (ЭТДА). Лишь при таком подходе можно выработать унифицированные интерфейсы и протоколы взаимодействия между ПО управления ЭТДО и ПО управления ЭТДА, обеспечив не точечную, а системную интеграцию между ними. Пока что эта граница определяется разработчиками ПО самостоятельно для каждой программной системы. До сих пор программное обеспечение создания и обработки технической информации делает упор на работу с ЭТДА; вопросы представления ЭТДА в виде подписанных ЭТД разработчиков фактически не волновали, в частности по причине того, что применение ЭТД не является легитимным.

К тому же административное управление ролями и пользователями в приложениях, функционирующих в рамках одной информационной системы, частично заменяет использование электронной цифровой подписи (ЭЦП).

Кроме того, электронные данные в общем виде — та самая «глина», позволяющая разработчикам программных систем «лепить» электронные объекты самыми разнообразными способами, часто не учитывающими главного: документ должен свободно перемещаться между разными информационными системами, приложениями и обрабатываться в них, в то время как объекты электронных баз данных обычно привязаны к своим прикладным средствам и СУБД. Но работа с ЭТД связана, прежде всего, с его «организационными» свойствами. Документ — бумажный или электронный — интересует нас как механизм, прежде всего регулирующий взаимоотношения между людьми, организациями. С необходимостью применения ЭТДА формально, в виде документально оформленных объектов баз данных (например, трехмерных моделей, каталогов деталей) возникает масса технических и процедурных вопросов, связанных, прежде всего, с применением ЭЦП, переносимостью этих объектов между технологическими средами разных информационных систем и т.д.

Применение ЭТД требует управления применением ЭЦП, а последнее основывается на **специфике электронного представления ЭТД**. Между тем, например, в России Федеральный закон об электронной цифровой подписи определяет электронный документ как «документ, в котором информация представлена в электронно-цифровой форме» (в Украине только готовится аналогичный закон). К сожалению, такое определение говорит об информации, но не о самом документе, и уж тем более о специфике ЭД. Необходимость работы с ЭД и ЭТД заставляет предприятия

самим давать определения этим терминам, разрабатывать собственные системы правил и механизмы управления ЭЦП и ЭД, что порождает множество различий в реализациях, часто принципиальных. При интеграции с информационными системами других предприятий возникают неизбежные коллизии или несовместимость. И если люди с их интеллектуальной системой обработки данных и принятия решений могут договориться в таких случаях, то их компьютерные системы здесь бессильны — им нужны стандартные интерфейсы и протоколы обмена.

5 Технологии обеспечения надежности корпоративных информационных систем

Первыми открытыми системами, построенными в расчете на высокую готовность, были приложения баз данных и систем коммуникаций. Базы данных высокой готовности гарантируют непрерывных доступ к информации, которая является жизненно важной для функционирования многих корпораций и стержнем сегодняшней информационной экономики.

Программное обеспечение систем коммуникаций высокой готовности усиливает средства горячего резервирования систем и составляет основу для распределенных систем высокой готовности и систем, устойчивых к стихийным бедствиям. Несколько компаний, поставляющих базы данных, такие как Oracle, Sybase Informix, имеют в составе своих продуктов ПО, позволяющее выполнять быструю реконструкцию файлов в случае отказа системы. Это снижает время простоя для зеркальных серверов и кластерных решений.

Первоначально модель вычислительного кластера была разработана компанией Digital в конце 80-х годов. Она известна под названием *VMS-кластеров* (или VAX-кластеров), которые представляли собой объединенные в кластер мини-компьютеры VAX.

Несмотря на то что кластерные решения предлагают более десятка фирм, VMS-кластеры удерживают лидерство по числу установок (более 20 тыс.) и до сих пор являются наиболее функционально полной реализацией парадигмы кластерных вычислений.

Использование ОС UNIX в качестве платформы для кластеров баз данных началось совсем недавно (в 1993 году). До этого времени попытки реализации подобных систем, осуществленными такими компаниями, как IBM, Sequent, Pyramid и другими, коммерческого успеха не имели. В 1996 году во всем мире едва ли функционировало полсотни UNIX-кластеров баз данных – признать это массовым освоением рынка, конечно, нельзя.

Этап распределения ресурсов, *Escalante (1999-2000),* должен обеспечить пользователям взаимодействие с единой системой, точнее, с единым образом системы (Single System Image, SSI), полностью спрятав ее (возможно, гетерогенную) внутреннюю сущность. Клиент сможет запрашивать конкретные сервисы и ресурсы и получать их «вовремя и без потерь», не задумываясь о том, где конкретно в сети их искать.

По мнению Novell, этот этап должен сделать кластеры обычным явлением на всех стандартных аппаратных платформах. Серверы, данные и приложения должны образовывать одну «виртуальную сеть». То есть все ресурсы должны представать перед пользователем как составляющие одного объекта, администрируемого как единый образ системы (SSI).

Многие компании в настоящее время работают над задачей построения гетерогенных кластерных систем, позволяющих интегрировать ПК- и UNIX-серверы. В такой гетерогенной среде для непосредственного обслуживания клиентских систем будут использоваться серверы на Intel-платформе с NT, объединенные в кластер с UNIX-серверами. Последние могут быть предназначены для резервного копирования, быстрого тиражирования баз данных или приложений групповой работы.

Сегодня среди кластерных систем старшего класса преобладают UNIX на базе RISC-процессоров. Но по мере того как ПО и операционные системы для кластеризации будут совершенствоваться и предлагать все большие функциональные возможности, кластеринг, безусловно, станет основным способом построения надежных масштабированных систем с Windows NT и NetWarp, являющихся промежуточным звеном между SMP-машинами и компьютерами с массовой параллельной обработкой.

Надежность и отказоустойчивость серверов

Важнейшая характеристика любой вычислительной системы, каковой является и сервер, - надежность. Повышение надежности основано на принципе предотвращения неисправностей путем снижения интенсивности отказов и сбоев за счет применения специальных технологических решений и совершенствования методов сборки аппаратуры.

Отказоустойчивость – это такое свойство системы, которое обеспечивает ей возможность продолжения заданных действий после возникновения неисправностей. Введение отказоустойчивости требует избыточного программного и аппаратного обеспечения. Направления, связанные с предотвращением неисправностей и с отказоустойчивостью, - основные в проблеме надежности. Понятие надежности включает не только аппаратные средства, но и программное обеспечение. Главной целью повышения надежности систем является целостность хранимых в них данных. Один из важных показателей отказоустойчивости – наличие у сервера памяти с коррекцией ошибок (ECC – Error Checking and Correction). Применение ECC позволяет серверу обнаруживать и исправлять ошибки в памяти, которые способны вызвать сбой или зависание системы в целом. Если исправление ошибок не возможно, часть банка памяти или весь банк целиком будут заблокированы, что вызовет лишь уменьшение объема доступной памяти с простым контролем на четность.
Другой не менее важный показатель отказоустойчивости аппаратных средств – это поддержка работы с матрицей дисков RAID.

Избыточность систем питания и охлаждение сервера в сочетании с возможностью «горячей» замены и источников питания еще больше повышают надежность работы серверной системы.

Производительность и масштабируемость

Производительность сервера определяется его архитектурой, т.е. конкретной реализацией отдельных компонентов сервера и связей между ними. К основным элементам архитектуры сервера можно отнести следующие: процессоры, средства реализации многопроцессорных вычислений, средства организации совместной работы процессоров с кэш-памятью и оперативной памятью, средства организации подсистемы ввода/вывода и ее взаимодействия с процессором и оперативной памятью.

Увеличение производительности и быстродействия достигается благодаря распараллеливанию операций ввода/вывода между несколькими дисками, использованию достаточно большого объема кэш-памяти RAID-контроллера и снижению нагрузки на процессор сервера за счет обработки операций ввода/вывода процессором RAID-контроллера.

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 сервера, а также соответствующими средствами ПО.

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 сервера 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. Она зависит также и 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. Аппаратные средства (процессоры, шины и устройства ввода/вывода) являются только частью масштабируемой архитектуры, на которой ПО может дать предсказуемых 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.

Управляемость

Управляемость подразумевает возможность конфигурирования серверов, мониторинга их состояния и оперативного устранения возникающих проблем.

Для обеспечения управляемости серверы комплектуются как аппаратными, так и программными средствами. К аппаратным средствам относятся встроенные датчики температурного режима и напряжения питания таких компонентов сервера как жесткие диски, источник питания и вентиляторы, а также датчик, отвечающий за автоматическую перезагрузку сервера при «зависании» сетевой ОС. Автоматическая перезагрузка позволяет существенно сократить время простоя сервера. Информация, получаемая встроенными датчиками, может собираться и анализироваться с помощью входящего в комплект поставки специального ПО.
Системы управления сервером должны самостоятельно решать незначительные проблемы и сообщать администратору о возникновении серьезных.

Совместимость с сетевым ПО

Концепция программной совместимости заключается в создании такой архитектуры, которая была бы одинакова с точки зрения пользователя для всех моделей системы, независимо от цены и производительности каждой из них. Огромные преимущества такого подхода, позволяющего сохранять существующий задел ПО при переходе на новые (как правило более производительные) модели, были быстро оценены как производителями компьютеров, так и пользователями.

5.1 Аппаратные средства повышения надежности

Некоторые термины
Важнейшая характеристика любой вычислительной системы, каковой является и сервер, - надежность. Поскольку эффективность работы сети зависит от функционирования сервера, к его надежности предъявляются особые требования. Имеет смысл рассматривать две составляющие надежности. Первая – устойчивость к внешним воздействиям и сбоям аппаратуры (что, как правило, и вкладывается в понятие надежности), устойчивость и предсказуемость системы при увеличении трафика сетевой нагрузки. Компания IBM утверждает, что ПК-серверы при повышении нагрузки до 60% от максимальной начинают вести себя непредсказуемо. Обеспечение надежности сервера является самой главной задачей, поскольку только надежностью поддерживается полезность остальных функциональных характеристик сервера.

Высокая готовность **(High Availability)**

Настоящие конструкции с высоким коэффициентом готовности для времени простоя используют обычную компьютерную технологию. При это конфигурация системы обеспечивает ее быстрое восстановление после обнаружения неисправности, для чего в ряде мест используются избыточные программные и аппаратные средства. Длительность задержки, в течение которой программа, отдельный компонент или система простаивает, может находится в диапазоне от нескольких секунд до нескольких часов, но более часто в диапазоне от 2 до 20 минут. Обычно системы высокой готовности хорошо масштабируются, предлагая пользователям большую гибкость, чем другие типы избыточности.

Эластичность к отказам **(Fault Resiliency)**

Ряд поставщиков компьютерного оборудования делит весь диапазон систем высокой готовности на две части, при этом в верхней его части оказываются системы, эластичные к отказам. Ключевым моментом в определении эластичности к отказам является более короткое время восстановления, которое позволяет системе быстро «откатиться» назад при обнаружении неисправности.

Устойчивость к отказам **(Fault Tolerance)**

Отказоустойчивые системы имеют в своем составе избыточную аппаратуру для всех функциональных блоков, включая процессоры, источники питания, подсистемы ввода/вывода и подсистемы дисковой памяти. Если соответствующий функциональный блок неправильно работает, всегда имеется

горячий резерв. В наиболее продвинутых отказоустойчивых системах избыточные аппаратные средства можно использовать для распараллеливания обычных работ. Время восстановления после обнаружения неисправности для переключения отказавших компонентов на избыточные для таких систем обычно меньше одной секунды.

Непрерывная готовность **(Continuos Availability)**
Вершиной линии отказоустойчивых систем являются системы, обеспечивающие непрерывную готовность. Продукт с непрерывной готовность, если он работает корректно, устраняет любое время простоя как плановое, так и не плановое. Разработка такой системы охватывает как аппаратные средства, так и программное обеспечение и позволяет проводить модернизацию (upgrade) и обслуживание в режиме on-line. Дополнительным требованиям к таким системам является отсутствие деградации в случае отказа. Время восстановления после отказа не превышает одной секунды.

Устойчивость к стихийным бедствиям **(Disaster Tolerance)**

Широкий ряд продуктов и услуг связан с обеспечением устойчивости к стихийным бедствиям. Иногда устойчивость к стихийным бедствиям рассматривается в контексте систем высокой готовности. Смысл этого термина в действительности означает возможность рестарта или продолжения операций на другой площадке, если основное месторасположение системы оказывается в нерабочем состоянии из-за наводнения, пожара или землетрясения. В простейшем случае продукты, устойчивые к стихийным бедствиям, могут просто представлять собой резервные компьютеры, расположенные вне основного месторасположения системы, сконфигурированные по спецификациям пользователя и доступные для использования в случае стихийного бедствия на основной площадке. В более сложных случаях устойчивость к стихийным бедствиям может означать полное (зеркальное) дублирование системы вне основного месторасположения, позволяющее принять на себя работу немедленно после отказа системы на основной площадке.

Единицей измерения надежности является среднее время наработки на отказ (MTBF – Mean Time Between Failure). Повышение готовности – есть способ борьбы за снижение времени простоя системы. Единицей измерения здесь является коэффициент готовности, который определяет вероятность пребывания системы в работоспособном состоянии в любой произвольный момент времени. Статистически коэффициент готовности определяется как MTBF/(MTBF+MTTR), где MTTR (Mean Time To Repair) – среднее время восстановления (ремонта), то есть среднее время между моментом обнаружения неисправности и моментом возврата системы к полноценному функционированию.

«Болевые точки» серверов

Современные недорогие серверы имеют много встроенных возможностей, повышающих их надежность. Тем не менее, любой из перечисленных ниже компонентов может стать слабым звеном, способным вывести сервер из строя:

- Вентиляторы обычно отказывают первыми. Выбирайте такие вентиляторы, которые можно легко вынуть без отключения сервера;
- Двойные источники питания способны сохранить работоспособность сервера, если один из них даст сбой. Однако для замены неработающего источника все-таки может понадобиться выключить сервер;
- Сетевые платы могут при совместном использовании уменьшать сетевой трафик, или одна из них выполнять роль страховочной, если откажет другая;
- Жесткие диски с поддержкой RAID могут содержать копии данных друг друга, некоторые допускают свою замену при сбое.

По данным зарубежных аналитиков отказы серверов вызваны следующими причинами: сбой (зависание) сетевой ОС – 79%, отказ дисковых накопителей – 5%, отказ ПО – 4%, проблемы сети электропитания – 8%, отказ источника питания – 2%, отказ ЦП – 1%, другие причины – 1%.

Поэтому важным показателем надежности работы сервера является успешное прохождение сертификационных испытаний на совместимость той или иной ОС.

5.2 Программные средства обеспечения надежности
(на основе Контрактного программирования)

Теория Контрактного Проектирования является основой для решения многих проблем, критических для ОО подхода: какую ОО "методологию" применять, на какие понятия опираться на стадии анализа, как специфицировать компоненты, как документировать ОО ПО, чем руководствоваться при выполнении тестирования. Все это вместе обеспечивает систематический подход к построению ПО с меньшим количеством дефектов. И надежное ПО получается как итог надлежащим образом встроенных в процесс разработки действий.

Понятие "контрактного проектирования" (Design by Contract) – сердцевина "Метода Эйфеля", разработанного автором систематического подхода к созданию надежного объектно-ориентированного программного обеспечения. Это понятие столь же важно для ОО парадигмы, как и классы, объекты, наследование, полиморфизм и динамическое связывание. Для получения уверенности в надлежащей работе ОО ПО необходим систематический подход к специфицированию и реализации ОО программных сущностей и их взаимосвязей в программной системе. Эта статья содержит введение в подход "Контрактное Проектирование", предложенный компанией Interactive Software Engineering.
При оценке новых методов и средств разработки ПО обычно ориентируются на их производительность. Объектные технологии действительно могут существенно повысить производительность, при этом, однако, нельзя упускать из виду качество создаваемого ПО. Качественное ПО — это прежде всего надежное ПО. Надежность — это способность системы функционировать в соответствии со спецификацией ("корректность") и при этом успешно справляться с возникающими ненормальными ситуациями ("устойчивость" — robustness). Ну, а проще говоря, надежная программа не содержит ошибок.

Конечно, надежность — это желательное качество ПО безотносительно к методу его разработки. Однако, объектно-ориентированный метод предполагает повышенные требования к надежности — прежде всего из-за той особой роли, которую здесь играет повторное использование программных компонентов, в корректности которых не должно быть никаких сомнений.

Есть несколько составляющих подхода к построению надежного ОО ПО. Например, статическая типизация – это признанный механизм для выявления ошибок еще до того, как они проявляются в виде "багов" в исполняемой программе. Такой метод, как сбор мусора, весьма полезен при защите от целого спектра ошибок, связанных с управлением памятью (вот почему я не согласен с теми, кто считает наличие механизмов сбора мусора лишь желательной, но не обязательной деталью реализации). Да и принцип повторного использования при надлежащем его воплощении также можно рассматривать как инструмент более надежной разработки. Если применяемая технология отлажена до такой степени, что вместо разработки каждый раз своего собственного решения вы опираетесь на повторное использование компонентных библиотек, произведенных и проверенных сторонним производителем, имеющим хорошую репутацию, то это означает, что существенной части программного обеспечения можно доверять в той же степени, что и компьютеру, на котором оно исполняется. Другими словами, повторно используемые библиотеки по-существу становятся частью единой "программно-аппаратной машины", компонентами которой можно считать аппаратуру, операционную систему и компилятор.

Но и это еще не все. Если мы хотим быть уверенными в надлежащей работе ОО ПО, то мы нуждаемся в систематическом подходе к специфицированию и реализации

ОО программных сущностей и их взаимосвязей в программной системе. Этот подход, известен как "Контрактное Проектирование" ("Design by Contract") и в его рамках программная система рассматривается в виде множества взаимодействующих компонентов, чьи отношения строятся на основе точно определенной спецификации взаимных обязательств – контрактов. Контрактное Проектирование обеспечивает:

- лучшее понимание ОО метода и — в более широком плане — процесса конструирования ПО;
- систематический подход к построению не содержащих "багов" ОО систем;
- эффективную концептуальную схему для отладки, тестирования и — в конечном итоге — гарантии качества;
- метод для документирования программных компонентов;
- лучшее понимание и управление механизмом наследования;
- метод для обработки ненормальных ситуаций, обеспечивающий, в частности, безопасную и эффективную языковую конструкцию для обработки исключений.

Все идеи, подробно рассмотренные здесь, являются составной частью Эйфеля (Eiffel), который надо рассматривать не только (и не столько!) как конкретный язык программирования, а как метод разработки ПО [1,2, 3].

Специфицирование и отладка

Первая и возможно самая трудная проблема, возникающая при построении ПО повышенной надежности, связана с точным определением того, что, собственно, каждая программная единица должна делать. Иными словами, речь идет о необходимости точных и полных спецификаций функционирования программных сущностей. Конечно, можно сразу же возразить — наличие спецификации вовсе не гарантирует, что модуль будет работать в полном соответствии с ней. Это верно, однако:

- без спецификации, определяющей, что должен делать модуль, вероятность того, что он все-таки именно это и будет делать, очень мала ("закон исключенного чуда");
- на практике, бывает неожиданно обнаруживается, насколько простое и следующее некоторым правилам соглашение о том, что модуль должен делать, гарантирует, что он действительно будет делать именно это.

Наличие спецификации, даже если она не полностью гарантирует корректность модуля, является хорошей базой для систематического тестирования и отладки.

Теория Контрактного Проектирования предполагает приложение спецификации к каждому программному элементу. Эти спецификации (или "контракты") и управляют взаимодействием элемента с окружающим его миром.

Однако не имеется в виду использование законченных формальных спецификаций. Хотя формальные спецификации имеют свои — и немалые — достоинства, все же более перспективен подход, в котором спецификации не обязательно являются исчерпывающе определенными с формальной точки зрения. Это позволяет инкорпорировать язык специфицирования непосредственно в язык проектирования и программирования (в данном случае — Eiffel), в то время как языки формальных спецификаций обычно либо не исполняемы, либо, хотя и исполняемы, но могут использоваться только при создании прототипов.

Наши же критерии предъявляют более жесткие требования: язык используется в рамках практической технологии при разработке коммерческого ПО, что предполагает эффективность реализации. Это позволяет претворить в жизнь ключевое свойство надлежащим образом понимаемого ОО процесса: его бесшовность (seamlessness), что делает возможным использование одной единственной нотации и одного набора понятий на всем протяжении жизненного цикла программного продукта

— от анализа до реализации и сопровождения, причем с гарантиями установления корректного взаимного отображения проблемы и решения. Это, помимо прочих выгод, обеспечивает безболезненную эволюцию продукта.

Понятие Контракта

В человеческих взаимоотношениях под контрактами понимаются документы, фиксирующие условия соглашения между двумя сторонами, одна из которых (поставщик) выполняет некоторую задачу в пользу другой (клиента). Каждая сторона имеет свои выгоды от контракта, и принимает, в свою очередь, некоторые обязательства. Обычно, то, что для одной из сторон – обязательство для другой – выгода. Цель же контрактного документа – зафиксировать формально эти выгоды и обязательства.

Для выражения условий контракта (например, между авиаперевозчиком — поставщиком и пассажиром — клиентом) часто является удобной табличная форма. Контрактный документ защищает и клиента (специфицируя, что и как должно быть им сделано), и поставщика (устанавливая, что он не обязан выполнять задачу, выходящую за рамки оговоренных условий).

Те же идеи применимы и при разработке программ. Рассмотрим программный элемент E. Чтобы достичь своей цели (выполнить свой контракт), E использует определенную стратегию, которая включает ряд подзадач t1,..., tn. Если подзадача ti является нетривиальной, то она будет выполняться с помощью вызова некоторой процедуры R. Иными словами, E передает работу по контракту процедуре R. Такая ситуация должна управляться хорошо определенным реестром обязательств и выгод — контрактом.

Предположим для примера, что ti – это задача вставки некоторого элемента в словарь (таблицу, где каждый элемент идентифицируется определенным символом – ключом) ограниченного размера. Тогда можно предложить следующий контракт.

Этот контракт управляет отношениями между процедурой и тем, кто потенциально может ее вызвать. Он содержит наиболее важную информацию об этих отношениях: что каждая сторона должна гарантировать для корректного вызова и что она в конечном счете получит.

Что здесь действительно важно, так это то, что мы не можем удовлетвориться неформальной спецификацией контракта в представленном виде. Если следовать духу "бесшовности" (поощряющем включать в один программный текст всю относящуюся к делу информацию на всех уровнях), то мы должны дополнить текст процедуры записью необходимых условий. Предполагая, что имя процедры – put, можно записать (используя синтаксические соглашения Eiffel) в виде части generic класса DICTIONARY[ELEMENT]:

```
put (x: ELEMENT; key: STRING) is
// Вставить x так, чтобы его можно было найти по ключу
  require
     count <= capacity
     not key.empty
  do
// ...некоторый алгоритм вставки
  ensure
   has (x)
   item (key) = x
   count = old count + 1
```

```
end
```

Здесь раздел require содержит входное условие или предусловие; раздел ensure вводит выходное условие – или постусловие. Оба условия являются примерами утверждений (assertions) или логических условий (пунктов контракта), ассоциированных с программными сущностями. В предусловии, count – это текущее число элементов, а capacity – максимальное число элементов в таблице. В постусловии, has – это запрос, отвечющий, на вопрос – присутствует ли уже элемент в таблице, item – возвращает элемент, ассоциированный с определенным ключом. Наконец, old count ссылается на величину count на входе в процедуру.

Контракты в анализе

Приведенный пример иллюстрирует процедуру, описывающую реализацию (хотя понятие словаря является фактически независимым от любой специфики реализации). Однако, введенные понятия столь же интересны и полезны на уровне анализа. Представим, например, объектную модель химического завода, содержащую классы TANK, PIPE, VALVE, CONTROL_ROOM. Каждый из этих классов описывает некоторую абстракцию данных – некоторый тип объектов реального мира, характеризуемых свойствами (особенностями, операциями). Например, TANK как абстракция резервуара, имеет следующие особенности:

запросы типа Да/Нет: is_empty, is_fulls
другие запросы: in_valve, out_valve (оба типа VALVE), gauge_reading, capacitys
команды: fill, emptys
Затем, чтобы охарактеризовать команды (например, fill), мы можем использовать предусловия и постусловия:

```
fill is
// Заполнить резервуар жидкостью
 require
   in_valve.open
   out_valve.closed
 deferred  // т.е. без реализации
   ensure
   in_valve.closed
   out_valve.closed
   is_full
 end
```

Такой стиль анализа позволяет избежать классической дилеммы анализа и специфицирования: либо вы используете программистскую нотацию и, соответственно, принимаете риск преждевременных обязательств реализации; или вы рискуете увязнуть в высоко-уровневой нотации (с "облаками и стрелками"), которая неизбежно приводит к спецификациям с достаточно туманно (с точки зрения следующих фаз жизненного цикла) выраженными свойствами, отказываясь от установления и прояснения некоторых тонких свойств системы уже на этапе анализа. Представленная нотация является точной (благодаря механизму утверждений, способному выражать семантику различных операций), и при этом избегает каких-либо обязательств по реализации. (В приведенном примере такого рода опасности нет – ведь то, что он специфицирует, не является программой, предназначенной для исполнения на компьютере. В данном случае, эта нотация выступает как инструмент моделирования).

Единственной ОО методикой, которая в полной мере интегрирует эти идеи на уровне анализа и проектирования, является Business Object Notatoion (BON) [5], которая обеспечивает соответствующую графическую нотацию.

Обязательства. Выгоды

Клиент Должен гарантировать удовлетворение предусловий. Быть в аэропорту Санта Барбара по крайней мере за 5 минут до обозначенного в расписании времени отлета. Не иметь недозволенных вложений в багаже. Заплатить за билет. Может ожидать выгоды от постусловия. Попасть в Чикаго.

Поставщик Должен гарантировать выполнение постусловия. Доставить клиента в Чикаго. Может предполагать, что предусловие выполняется. Нет никакой нужды перевозить пассажира, если он опоздал, имеет недозволенные вложения в багаже или не заплатил за билет.

Контракт о включении в таблицу

Обязательства Выгоды

Клиент Должен гарантировать удовлетворение предусловий. Удостовериться, что таблица не заполнена полностью и что ключ не является пустой строкой. Может ожидать выгоды от постусловия. Получить обновленную таблицу, где присутствует заданный элемент, ассоциированный с задданным ключом.

Поставщик Должен гарантировать выполнение постусловия. Записать заданный элемент в таблицу и установить его связь с заданным ключом. Может предполагать, что предусловие выполняется. Нет необходимости выполнять какие-либо действия, если таблица заполнена до отказа или ключ является пустой строкой.

Инварианты

Предусловия и постусловия применимы к отдельным процедурам. Можно предложить и другие виды утверждений, которые будут характеризовать класс как целое, а не просто свойства его индивидуальных процедур. Утверждение, описывающее свойство, которое сохраняется для всех экземпляров класса, называется инвариантом класса. Например, инвариант класса DICTIONARY может установить:

```
invariant
  0 <= count
  count <= capacity
```

А инвариант класса TANK может установить, что is_full на самом деле означает "почти полный":

```
invariant
  is_full = (0.97 * capacity <= gauge)
    and gauge <= 1.03 * capacity)
          ...другие clauses
```

Инварианты класса – это ограничения, характеризующие его семантику с точки зрения непротиворечивости. Это понятие является особенно важным в контексте управления конфигурацией и регрессионного тестирования – оно описывает более глубокие свойства класса: здесь не просто устанавливается характеристика, верная для определенного момента эволюции класса, но и налагаются ограничения, приложимые ко всем последующим изменениям.

С точки зрения контрактной теории, инвариант является обобщенным утверждением, применимым ко всему множеству контрактов, определяющих класс.

Документирование
Еще одно важное применение контрактов – обеспечение стандартного способа документирования программных сущностей – классов. Программисты, разрабатывающие клиентские приложения нуждаются в надлежаще оформленном описании интерфейсных свойств класса; основанный на контрактах подход позволяет использовать версию класса, известную под названием "короткая форма" (short form). В ней опущены все детали реализации, но зато представлена исчерпывающая информация об использовании класса – контракт.

В среде EiffelBench, короткую форму класса можно получить в интерактивном режиме – по нажатию одной клавиши, причем в любом желаемом формате (RTF, HTML, MIF или MML для FrameMaker, TEX, troff, Postscript и т.д.) через один из предопределенных в среде фильтров, к которым можно добавить и любой другой фильтр – данный механизм носит открытый характер.

Короткая форма представляет собой заголовки и утверждения экспортированных особенностей, а также инварианты. Например:

```
class interface
DICTIONARY [ELEMENT] feature
put (x: ELEMENT; key: STRING) is
// Вставить x так, чтобы его можно было найти по ключу.
  require
      count <= capacity
not key.empty
  do
...некоторый алгоритм вставки...
  ensure
      has (x)
item (key) = x
count = old count + 1
  end
...спецификации итерфейсов других особенностей...
invariant
  0 <= count
count <= capacity
end
```

Короткие формы классов могут быть использованы в качестве базового средства при документировании библиотек и других программных сущностей. Они также весьма эффективны как средство коммуникации между разработчиками. Наш опыт показал, что акцент на систематическое использование коротких форм облегчает проектирование ПО и управление процессом его разработки, так как поощряет и разработчиков и менеджеров обсуждать действительно ключевые вопросы (интерфейс, спецификация, межмодульные протоколы), а не детали реализации.

Тестирование, отладка и гарантии качества
Для класса, в описание которого входят логические утверждения, в идеале можно было бы попытаться доказать математически, что реализации процедур с этими утверждениями согласуется. Однако, пока что, реалистические средства, позволяющие это сделать , отсутствуют; зато вполне можно получить ощутимые выгоды, используя утверждения при тестировании.

Доступные разработчику опции компиляции позволяют исследовать, какой эффект для каждого класса оказывают утверждения, если:

обработка утверждений отключена (в этом случае они представляют собой стандартизованные комментарии);
обрабатываются только предусловия (что и происходит по умолчанию);
обрабатываются предусловия и постусловия;
предусловия, постусловия и инварианты класса;
обрабатываются все утверждения.
Эти механизмы являются мощным средством обнаружения ошибок. Мониторинг утверждений – это способ проверить, действительно ли программа делает то, что разработчик от нее ожидает. В результате мы имеем весьма продуктивный подход к тестированию, отладке и повышению качества ПО, где поиск ошибок ведется не вслепую, а на базе условий непротиворечивости, сформулированных самими разработчиками.

Важно, что эти механизмы вполне доступны; опыт показывает, что их использование обеспечивает радикальное сокращение "багов" в программах и помогает формированию нового самосознания разработчиков, акцентированного на надежность ПО.

Контракты и наследование
Важное следствие теории Контрактного Проектирования – это лучшее понимание таких центральных для объектной парадигмы понятий, как наследование, полиморфизм, переопределение и динамическое связывание.

Класс B, который наследует от класса A, может обеспечить новую декларацию для некоторой унаследованной особенности r, до того определенной в A. Например, специализирующая реализация класса DICTIONARY может переопределить алгоритм для put. Такого рода переопределения несут в себе потенциальную опасность, так как переопределенная версия может, в принципе, иметь совершенно другую семантику. Это особенно настораживает в присутствии полиморфизма, который означает, что в вызове типа a.r — целевой объект вызова a, через статически декларированный тип A, может фактически во время исполнения быть "пришпилен" к объекту типа B. Затем, вступающее в действие динамическое связывание предполагает, что в этом случае будет вызываться именно "B – версия" r. По-существу, мы имеем здесь дело с производным контрактом – суб-контрактом ("subcontract"): A передает контракт ("суб-контрактирует") для r объекту B для цели соответствующего типа. Но суб-контрактор должен быть связан оригинальным контрактом. Если клиент осуществляет вызов в форме: if a.pre then a.r end, то ему должен быть гарантирован обещанный контрактом результат: сам же вызов будет выполняться корректно, так как удовлетворяется предусловие (предполагая, что pre означает предусловие для r); на выходе же a.post будет истинно, где post – постусловие для r.

Отсюда следует принцип производного контрактирования (subcontracting): переопределенная версия r может сохранять или ослаблять предусловие и сохранять или усиливать постусловие. Усиление предусловия или ослабление постусловия (что можно обозначить как "нечестный производный контракт") способно привести к катастрофе. Язык Eiffel содержит правила для переопределения утверждений в соответствии с принципом производного контрактирования.

Сделанные замечания не только обеспечивают полезные указания по корректному использованию наследования, но и проливают свет на истинную его значимость: это не просто механизм для повторного использования, введения подтипов и обеспечения классификации, но и способ гарантировать совместимую семантику.

Обработка исключений

Среди многих других приложений контрактной теории отметим, что она предлагает систематический подход к решению трудной проблемы обработки исключений – дает рецепты, как строить ПО, должным образом реагирующее на ненормальные ситуации.

По-существу, любая программная единица выполняет некоторый контракт, даже если в явной виде он не сформулирован. "Исключение" (exception) – это неспособность такой контракт выполнить – по любой причине: из-за сбоя в аппаратуре или программного дефекта.

В таких случаях имеют смысл три возможные реакции.

Повтор (Retrying): при условии, что возможна альтернативная стратегия. Процедура восстановит инвариант и сделает еще одну попытку, задействовав эту новую стратегию.
"Организованная паника" (Organized panic): никаких альтернатив не существует. В этом случае необходимо восстановить инвариант, завершить работу и рапортовать вызывавшей процедуре о неудаче, инициируя новое исключение.
"Ложная тревога" (False alarm): в этом случае на самом деле можно продолжить работу после принятия некоторых корректирующих мер. Впрочем, этот случай достаточно редок (к сожалению – ведь его легче всего реализовать).
Механизм исключений непосредственно следует из этого анализа. Он основан на понятии "раздел спасения" (rescue clause), ассоциированном с процедурой, и "команде повтора" (retry instruction), реализующей повторную попытку выполнения. Легко видеть здесь сходство с контрактами из обыденной жизни, которые обычно включают пункт или раздел, связанный с действиями в исключительных, заранее незапланированных обстоятельствах. Если такой "rescue clause" присутствует в контракте, то любое исключение, случившееся во время выполнения процедуры, прервет исполнение тела (раздел "do") и инициирует выполнение раздела "rescue". Этот раздел может содержать одну или более инструкций; одна из них – "retry" вызовет повторное выполнение тела процедуры (раздел "do"). Целочисленная локальная переменная failure всегда инициализируется при входе в процедуру нулевым значением (но, конечно, после retry).

Приведем пример, иллюстрирующий этот механизм [2,3]. Пусть низкоуровневая процедура unsafe_transmit передает сообщение по сети. Мы не имеем никакого контроля над этой процедурой, но знаем, что ее действия могут закончиться неудачей. В этом случае хотелось бы бы повторить все снова; после 100 таких неудачных попыток было бы целесообразно их прекратить, передав исключение вызывающей процедуре. Механизм Rescue/ Retry непосредственно поддерживает такую схему:

```
attempt_transmission
(message: STRING) is
// Попытка передачи сообщения по
// коммуникационной линии с использованием
// низкоуровневой (например, Си ) процедуры
// unsafe_transmit, которая может оказаться
// неудачной, порождая исключение. После 100
// безуспешных попыток, прекратить действие
// (инициируя исключение в вызывающей процедуре)
  local
     failures: INTEGER
  do
```

```
    unsafe_transmit (message)
  rescue
    failures := failures + 1
    if  failures < 100 then
     retry
    end
  end
```

Подведем итоги

Теория контрактного проектирования, основы который изложены в данной статье, продолжает развиваться. Отметим две перспективных области.

Параллельность и распределенность – принципы контрактного проектирования позволяют по новому взглянуть на фундаментальные проблемы параллельного (одновременного – concurrent) и распределенного ОО программирования (избегая при этом проблемы "аномальной наследственности" и некоторых других подобных псевдопроблем, проистекающих из непонимания существа ОО технологии). Принятый в Eiffel подход, основанный на контрактном проетировании и реализованный для ISE Eiffel 4.2 подробно разобран в [1,4].

Расширенный язык спецификаций – позволяет выражать еще более представительное множество утверждений.

Литература

1. B. Meyer, "Object-Oriented Software Construction", Prentice Hall, 1997
2. B. Meyer, "Applying Design by Contract", //Computer, Vol.25, No. 10, October 1992, pp. 40-51
3. B. Meyer, "Eiffel: The Language", //Prentive Hall, 1992
4. B. Meyer, "Systematic Concurrent Object-Oriented Programming", //Communicatons of the ACM, Vol. 36, No. 9, September 1993, p. 56-80
5. K. Walden and J.-M. Nerson, "Seamless Object-Oriented Software Architecture: Analysis and Design of Reliable Systems", //Prentice Hall, 1995 ?

5.3 Основные методы обеспечения надежности серверов и сравнительный анализ некоторых из них

Обслуживание без инструментов, резервные компоненты и «горячая» замена

Когда-нибудь вам непременно понадобится заменить отдельные компоненты сервера, будь то отказавшие или морально устаревшие и требующие обновления детали. Хорошо, если ваша машина позволяет выполнять эти операции быстро. Модели компаний IBM, Toshiba, Gateway могут похвастать «обслуживанием без инструментов». Обычно это означает, что для доступа внутрь корпуса и даже для полной замены узлов не потребуется ничего, кроме собственных пальцев. Для сравнения: чтобы открыть корпус компьютера Compaq ProLiant, нужна специальная торцевая отвертка под винты «звездочкой», что вроде бы способствует лучшей защите от посторонних. Однако согласно закону Мерфи можете быть уверены, что в нужный момент этой редкой отвертки не окажется именно у вас, а не у злоумышленника.
Как и ProLiant, сервер Crystal Group CS 900 также не предполагает моментальную замену компонентов, его корпус закреплен дюжиной обычных винтов. Но модель фирмы Crystal Group имеет одно существенное отличие для рабочих групп: в его относительно плотно заполненном корпусе нет места для установки резервных компонентов. По мнению поставщика, это объясняется тем, что сервер сам является одним резервным компонентом, так как при промышленном использовании рассчитан на совместную работу в связке с другими машинами CS 900. Если какая-либо деталь откажет, из стойки удаляется весь компьютер, в отключенном состоянии неполадка устраняется.
Другие поставщики этой философии не разделяют. Вместо этого они расхваливают резервные компоненты, хотя не все реализации хороши. Например, машина IBM Netfinity имеет избыточный источник питания, состоящий из двух различающихся частей, поэтому для замены придется держать в запасе два блока.
Наличие избыточных компонентов (таких как блоки питания, вентиляторы) делают возможным прием, называемый «горячей» заменой, т.е. замена отказавшего компонента без отключения компьютера.

Управляющее ПО
Сервер – это сердце сети. Он работает без выключения день за днем, порой на протяжении нескольких лет. Такой режим негативно влияет на компоненты с движущимися частями: чаще всего первыми отказывают источники питания, жесткие диски и охлаждающие вентиляторы.
Естественно было бы ожидать, что наиболее ценными (во всех смыслах этого слова) компонентами, обеспечивающими отказоустойчивость сервера, должны быть элементы, обладающие возможностью «горячей» замены. И все же не сами по себе компоненты с «горячей» заменой сохраняют работоспособность сервера.
Управляющее ПО – вот необходимый ингредиент, позволяющий администратору перезагружать сервер практически из любого места и предупреждающий его об отказе отдельных компонентов. Некоторые программные инструменты способны обнаруживать условия, ведущие к преждевременному выходу деталей из строя. Подобные утилиты сообщают, что такой-то компонент почти готов «закончить жизнь свою», даже если он продолжает нормально работать. Заметим, что при замене компонента это является основанием для возмещения его стоимости.

RAID – это не только надежность
В повседневной работе сетей задачи администрирования, распределения ресурсов и централизованного хранения данных возлагаются на сервер, например Windows NT. Когда на одной машине обрабатывается много важной информации, то сервер должен

сохранять данные с высокой степенью надежности. Для этого, кроме ежедневного создания резервных копий (Back up), может использоваться метод, который является оптимальным дополнением: RAID.
В 1987 году три специалиста по вычислительной технике из Калифорнийского университета в Беркли разработали концепцию, названную «избыточным массивом недорогих дисков» (Redundant Arrays of Inexpensive Disks, RAID). За прошедшие годы «I» в аббревиатуре RAID стала означать independent («независимых»), но идея, по сути, осталась прежней: обеспечение максимально быстрого доступа к данным и их защита посредством аппаратной избыточности.
Как правило, поток пользовательских интересов чрезвычайно равномерно распределен по всей дисковой памяти. Образуются так называемые «горячие пятна» - области, к которым происходит подавляющее большинство обращений, в то время как остальная часть памяти имеет очень низкую нагрузку. Наиболее общим решением этой проблемы остается распределение логического дискового пространства по набору (массиву) относительно небольших и сравнительно недорогих дисковых устройств со средним временем поиска. «Горячее пятно» располагается небольшими фрагментами на всех накопителях, чем достигается более равномерная их нагрузка. При очевидной привлекательности такого решения оно имеет и очень серьезный недостаток: с ростом числа дисков в массиве пропорционально возрастает и вероятность отказа. Увеличить надежность можно за счет введения избыточной емкости, обеспечивающей возможность восстановления разрушенных данных.

Технология RAID
Для реализации виртуальных дисков большой емкости из совокупности недорогих и небольших устройств были разработаны и стандартизованы алгоритмы объединения дисков, известные сейчас как RAID-алгоритмы. Основная идея этих алгоритмов заключается в том, что входной поток информации делится на блоки, которые записываются на диски. При считывании происходит обратный процесс – блоки информации собираются с накопителей и преобразуются в единый поток. На практике существует много применений таких алгоритмов, но самое большое распространение получили именно матрицы жестких дисков – RAID.
В зависимости от способа распределения блоков в дисковом массиве различают несколько уровней RAID, с 0 по 5. Именно они были изначально стандартизованы, хотя сейчас можно встретить контроллеры с более высоким уровнем RAID. Последние являются лишь продолжением развития RAID-технологии и к тому же могут отличаться у разных поставщиков. В сущности, они представляют собой комбинацию различных уровней RAID, принцип работы которой заключается в одновременном чтении (или записи) блоков данных из нескольких массивов дисков, каждый из которых, в свою очередь, представляет собой стандартный RAID.
Каждый из уровней RAID имеет свои сильные стороны, обеспечивая или более высокие скорости, или исключительную емкость памяти, или повышенную отказоустойчивость, но на текущий момент значительный интерес в сфере промышленности вызвали уровни RAID 1,3,5.
RAID-0 – алгоритм, при котором каждая порция информации входного потока делится на N фиксированных блоков (где N – число дисков в матрице), а затем каждый блок записывается на свой диск. В связке «контроллер-шина-диск» самым медленным является диск. Используя большое количество дисков, можно получить увеличение скорости записи/считывания системы до тех пор, пока позволяет пропускная способность интерфейсной шины. RAID-0 не создает никакой избыточности и не обеспечивает 100% сохранности данных при возникновении проблем с одним из дисков. Его назначение – сделать максимально быструю дисковую систему, скорость которой в идеале в N раз превышает быстродействие одного диска. Таким образом, RAID-0 – это еще не «настоящий» RAID, так как не обеспечивает отказоустойчивость: выход из строя одного диска приводит к потере данных во всем массиве.

RAID-1, или зеркализация (mirroring), требует четного числа дисков и осуществляет попарно дублирование информации. Данные записываются одновременно на два диска: если один из них откажет, информация обеспечивает удвоенную скорость считывания, однако время записи здесь такое же, как и при одном диске. Этот алгоритм уже дает стопроцентную гарантию сохранности данных при сбое любого из дисков и теоретическое увеличение скорости в N/2 раза, однако вдвое повышается и стоимость дискового пространства. Алгоритм применяется в системах, где необходима надежность сохранности данных, и объем дисковой матрицы незначителен, и, как правило, в таких системах, которые не поддерживают «горячую» замену отказавших дисков.
RAID-2 – в системах используется контроль кода Хэмминга с коррекцией ошибок (ECC, Error-Correcting Code) для их исправления «на ходу». Как и в системах с RAID-0, блок данных при записи делится на части, распределяемые по разным жестким дискам. Для записи кода коррекции ошибок требуется значительный объем дискового пространства, приобретение которого настолько удорожает RAID-2, что последние стали редкостью.
RAID-3 обеспечивает избыточность расщеплением данных на уровне байтов и сохранением информации о четности на выделенных ECC-дисках. Использование в качестве ECC обычного метода контроля четности значительно удешевляет данную систему по сравнению с RAID-2. RAID-3 обеспечивает высокую скорость считывания, особенно в случае больших последовательных файлов; при отказе одного из дисков производительность практически не снижается. А вот скорость записи RAID-3-систем в лучшем случае не выше чем у обычного диска, что определяется «узким местом» - использовании ECC при сохранении информации о четности.
RAID-4 построена аналогично предыдущему уровню, с тем исключением, что данные распределяются здесь поблочно, а не по байтам. Если объем обновляемых данных невелик и не требуется обращение ко всем дискам данных, то, оказывается, достаточно четырех запросов к дискам: двух запросов на считывание прежних данных и информации о четности, чтобы вычислить новое значение по контролю четности; и двух – на запись новых данных и информации о четности. Из-за высоких накладных расходов этот метод не нашел широкого применения.
RAID-5 – данный метод является самым распространенным при построении отказоустойчивых систем. В системе RAID-5 блок информации о четности в каждой полосе чередования попадает равномерно на различные диски, причем максимальное количество последних не превышает 32-х. Это полностью оправдывает себя при интенсивной обработки транзакций, для которой характерен одновременный перенос множества мелких фрагментов данных. Если случится отказ диска, это повлияет на производительность системы RAID-5, так как потребуется обработка отсутствующих данных «на ходу». Мало того, после замены отказавшего диска восстановление может занять значительное время, но система, разумеется, сохранит работоспособность.

Сравнительные характеристики наиболее распространенных RAID-систем
Сравнение конфигураций RAID необходимо проводить с учетом особенностей архитектуры и потребностей приложений, в которых эти массивы будут использованы. При этом следует учитывать не только чисто технические характеристики, но и другие требования, которые может предъявлять пользователь. В зависимости от требований к системе можно условно выделить три группы пользователей. Для первой группы наиболее важной характеристикой является стоимость. К следующей группе относятся пользователи, для которых более важна надежность. Для третьей группы основной характеристикой является производительность. Сравнительные характеристики RAID-систем, которые нашли широкое применение, приведены в таблице №1.

Технология Hot Swap (резервные диски)
Жесткие диски – один из немногих компонентов, в котором есть механическая часть, в первую очередь подверженная износу и долговременной работе. И какие бы хорошие и

надежные диски не использовались, всегда имеется вероятность выхода их из строя, увеличивающаяся со временем эксплуатации. Как правило, все компьютеры со статусом сервер не могут быть выключены в любой момент времени для замены жесткого диска. RAID-контроллеры позволяют решить эту проблему за счет поддержки функций «горячей» замены и технологии резервных дисков. В случае поломки диска или каких-либо других проблем, в следствие которых контроллер не может в дальнейшем его эксплуатировать, например превышение допустимого количества плохих блоков, диск помечается как мертвый (Kill Drive) и не участвует в дальнейшей работе. В этом случае можно вынуть этот диск из системы и заменить другим. После подключения и обнаружения RAID-контроллером нового диска произойдет его форматирование (если необходимо) и будет выполнена операция перестройки матрицы (Rebuild). Таким образом, система опять восстановит свой прежний вид. Так может продолжаться до тех пор, пока в запасе имеются резервные диски или пока при режиме работы без избыточности не выйдет из строя еще один диск.
Современные RAID-контроллеры позволяют решить эту проблему без участия человека. В матрицу заранее устанавливается диск или несколько дисков, которые при конфигурации RAID-матрицы помечаются как резервные (Standby). В этом случае на резервные диски не подается команда раскрутки шпинделя, они никак не эксплуатируются, а значит, не изнашиваются. При возникновении «мертвого» диска резервный из режима StandBy сразу же автоматически переводится в рабочий, и осуществляется Rebuild. В таком режиме сервер будет работать до тех пор, пока не кончатся резервные диски. Замена «мертвого» диска на новый, резервный, может быть произведена в любое удобное время.

Кластеризация
Повышение отказоустойчивости всегда было и остается главной задачей, стоящей перед разработчиками аппаратных средств вычислительной техники. Одним из эффективных способов решения этой проблемы является построение кластерных систем.
Термин «кластеризация» на сегодня компьютерной промышленности имеет много различных значений. Строгое определение могло бы звучать так: «реализация объединения независимых машин, представляющееся единым целым для ОС, системного ПО, прикладных программ и пользователей».
Машины, кластеризованные таким образом, могут при отказе одного процессора очень быстро перераспределить работу на другие процессоры внутри кластера.
Кластеринг может принимать разнообразные формы. Кластер может представлять из себя набор стандартных ПК, объединенных сетью Ethernet. На другом конце спектра аппаратная структура может состоять из высокопроизводительных SMP-систем, соединенных высокоскоростными коммуникациями и шиной ввода/вывода. По мере необходимости обработки более сложных или увеличивающихся в объеме запросов от клиентов, к кластеру могут быть добавлены дополнительные системы.
Существуют две модели кластеризации. В программной модели «shared nothing» каждая система, входящая в состав кластера, владеет частью его ресурсов. В каждый конкретный момент времени данным ресурсом кластера может владеть только одна система. Не смотря на это, в случае отказа другая, динамически определяемая система может принять на себя владение данным ресурсом. Тем самым обеспечивается постоянная доступность ресурса внутри кластера и, следовательно, для клиента. В модели «shared disk» ПО каждой системы получает доступ к любым ресурсам (например, дисковым), соединенным с любой системой или кластером в целом. Если две системы запрашивают одни и те же данные, они должны быть либо прочитаны дважды, либо скопированы с одной системы на другую.

Таблица №1

Уровень RAID	**Конфигурация накопителей**	**Уменьшение производительности**	**Повышение надежности системы**	**Эффективность по затратам**
0	Состоит из нескольких дисков для хранения данных; отсутствует диск для записи информации о четности	Обеспечивает самый быстрый доступ к дисковой подсистеме (лучшее решение для мощной графической станции, работающей с очень большими объемами мультимедиа файлов – в реальном времени, когда кэш уже не спасает)	нет	Наилучшая, так как вся доступная емкость используется для хранения данных
1	Представляет собой два зеркальных диска	Характеризуется ускорением чтения, но не записи (применяется при создании виртуальных дисков, главное требование к которым – надежность хранения данных)	Данные не теряются при отказе любого из дисков	Наихудшая, ибо оплачивается емкость, вдвое большая, чем фактически получается
3	Состоит из нескольких дисков для хранения данных и содержит один диск для записи информации о четности	Обычно производительнее, чем система RAID-1 или RAID-5, особенно при работе с крупными блоками данных (в случае малых блоков данных характеризуется незначительным ускорением записи)	Данные не будут потеряны при выходе из строя любого из дисков	Лучше, чем у RAID-1 и RAID-5, так как информация о четности занимает минимум места
5	Построен на нескольких дисках для хранения данных; информация о четности записывается равномерно на все диски	Наибольшее в случае небольших блоков данных, что типично для трафика локальной сети (применяется при создании виртуальных дисков, главное требование к которым – надежность хранения данных)	Данные не будут потеряны при выходе из строя любого из дисков	Лучше, чем у RAID-1, но немного хуже, чем у RAID-3

6. Современные и традиционные способы проектирования корпоративных информационных систем

6.1. Классификация CASE-средств

Все CASE-средства делятся на типы, категории и уровни. Классификация *по типам* отражает функциональную ориентацию CASE-средств в технологическом процессе.

I) *АНАЛИЗ И ПРОЕКТИРОВАНИЕ.* Средства данной группы используются для создания спецификаций системы и ее проектирования; они поддерживают широко известные методологии проектирования. К таким средствам относятся: *CASE.Аналитик (Эйтэкс), The Developer (ASYST Technologies), POSE (Computer Systems Advisers), ProKit* Workbench (McDonnell Douglas), Excelerator (Index Technology'), Design-Aid (,Nastec}, Design Macliine (Optima), MicroStep (Meta Systems), vsDesigner (Visual Software), Analist/Designer (Yourdon), Design/IDEF (Meta Software), BPWin (Logic Works), SELECT (Select Software Tools), System Architect (Popkin Software & Systems), Westmount /-CASE Yourdon (Westmount Technology B. V. & CADRE Technologies), CASE/4/0 (microTOOL GmbH).* Их целью является определение системных требований и свойств, которыми система должна обладать, а также создание проекта системы, удовлетворяющей этим требованиям и обладающей соответствующими свойствами. На выходе продуцируются спецификации компонент системы и интерфейсов, связывающих эти компоненты, а также "калька" архитектуры системы и детальная "калька" проекта, включающая алгоритмы и определения структур данных.

2) *ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ И ФАЙЛОВ* Средства данной группы обеспечивают логическое моделирование данных, автоматическое преобразование моделей данных в Третью Нормальную Форму, автоматическую генерацию схем БД и описаний форматов файлов на уровне программного кода: *ERWin (Logic Works), Chen Toolkit (Chen &Asssociates), S-Designor (SDP), Designer2000 (Oracle), Silverrun (Computer Systems Advisers).*

3) *ПРОГРАММИРОВАНИЕ.* Средства этой группы поддерживают этапы программирования и тестирования, а также автоматическую кодогенерацию из спецификаций, получая полностью документированную выполняемую программу: *COBOL 2/Workbench (Mikro Focus), DECASE (DEC), NETRON/CAP (Netron), APS (Sage Soft\vare).* Помимо диаграммеров различного назначения и средств поддержки работы с репозитарием, в эту группу средств включены и традиционные генераторы кодов,

анализаторы кодов (как в статике, так и в динамике), генераторы наборов тестов, анализаторы покрытия тестами, отладчики.

4) СОПРОВОЖДЕНИЕ И РЕИНЖИНИРИНГ. К таким средствам относятся документаторы, анализаторы программ, средства реструктурирования и реинжениринга : Adpac CASE Tools (Adpac), Scan/COBOL u Superstructure (Computer Data Systems), Inspector/Recoder (Language Technology). Их целью является корректировка, изменение, анализ, преобразование и реинжениринг существующей системы. Средства позволяют осуществлять поддержку всей системной документации, включая коды, спецификации, наборы тестов; контролировать покрытие тестами для оценки полноты тестируемости; управлять функционированием системы и т.п. Особый интерес представляют средства обеспечения мобильности (в CASE они получили название средств миграции) и реинжиниринга. К средствам миграции относятся трансляторы, конверторы, макрогенераторы и др., позволяющие обеспечить перенос существующей системы в новое операционное или аппаратурное окружение. Средства реинжиниринга включают:

• статические анализаторы для продуцирования схем системы ПО из ее кодов, оценки влияния модификаций (например, ” эффекта ряби " - внесение изменений с целью исправления ошибок порождает новые ошибки);

• динамические анализаторы (обычно, компиляторы и интерпретаторы с встроенными отладочными возможностями);

• документаторы, позволяющие автоматически получать обновленную документацию при изменении кода;

• редакторы кодов, автоматически изменяющие при редактировании все предшествующие коду структуры (например, спецификации);

• средства доступа к спецификациям, их модификации и генерации нового (модифицированного) кода;

• средства реверсного инжиниринга, транслирующие коды в спецификации.

5) *ОКРУЖЕНИЕ*. Средства поддержки платформ для интеграции создания и придания товарного вида CASE-средствам: *Multi/Cam (AGS Management Systems), Design/OA (Meta Software).*

6) *УПРАВЛЕНИЕ ПРОЕКТОМ*. Средства, поддерживающие планирование, контроль, руководство, взаимодействие, т.е. функции, необходимые в процессе разработки и сопровождения проектов: *Project Workbench (Applied Business Technology).*

Классификация *по категориям* определяет уровень интегрированности по выполняемым функциям и включает вспомогательные программы (tools), пакеты разработчика (toolkit) и инструментальные средств (workbench).

Категория tools обозначает вспомогательный пакет, решающий небольшую автономную задачу, принадлежащую проблеме боле широкого масштаба. Категория toolkit представляет совокупность интегрированных программных средств, обеспечивающих помощь для одного из классов программных задач; использует репозитарий для всей технической и управляющей информации о проекте, концентрируясь при этом на поддержке, как правило, одной фазы или одного этапа разработки ПО. Категория workbench представляет собой интеграцию программных средств, которые поддерживают системный анализ, проектирование и разработку ПО; используют репозитарий, содержащий всю техническую и управляющую информацию о проекте; обеспечивают автоматическую передачу системной информации между разработчиками и этапами разработки; организуют поддержку практически полного ЖЦ (от анализа требований и проектирования ПО до получения документированной выполняемой программы). Workbench, по сравнению с toolkit, обладает более высокой степенью интеграции выполняемых функций, большей самостоятельностью и автономностью использования, а также наличием тесной связи с системными и техническими средствами аппаратно- вычислительной среды, на которой workbench функционирует. По существу, workbench может рассматриваться как автоматизированная рабочая станция, используемая как инструментарий для автоматизации всех или отдельных совокупностей работ по созданию ПО.

Классификация *по уровням* связана с областью действия CASE в пределах жизненного цикла ПО. Однако четкие критерии определения границ между уровнями не установлены, поэтому данная классификация имеет, вообще говоря, качественный характер.

Верхние (Upper) CASE часто называют средствами компьютерного планирования. Они призваны повышать эффективность деятельности в руководителей фирмы и проекта путем сокращения затрат на определение политики фирмы и на создание общего плана проекта. Этот план включает цели и стратегии их достижения, основные действия в свете целей и задач фирмы, установление стандартов на различные виды взаимосвязей и т.д. Использование верхних CASE позволяет построить модель предметной области, отражающую всю существующую специфику. Она направлена на понимание общего и частного механизмов функционирования, имеющихся возможностей, ресурсов, целей проекта в соответствии с на- значением

фирмы. Эти средства позволяют проводить анализ различных сценариев (в том числе наилучших и наихудших), накапливая информацию для принятия оптимальных решений. Средние (Middle) CASE считаются средствами поддержки этапов анализа требований и проектирования спецификаций структуры ПО. Их использование существенно сокращает цикл разработки проекта; при этом важную роль играет возможность накопления и хранения знаний обычно имеющихся только в голове разработчика-аналитика, что позволит использовать накопленные решения при создании других проектов. Основная выгода от использования среднего CASE состоит в значительном облегчении проектирования систем, проектирование превращается в итеративный процесс, включающий следующие действия

- пользователь обсуждает с аналитиком требования к проектируемой системе;
- аналитик документирует эти требования, используя диаграммы и словари входных данных;
- пользователь проверяет эти диаграммы и словари, при необходимости модифицируя их;
- аналитик отвечает на эти модификации, изменяя соответствующие спецификации.

Кроме того, средние CASE обеспечивают возможности быстрого документирования требований и быстрого прототипирования.

Нижние (Lower) CASE являются средствами разработки ПО (при этом может использоваться до 30% спецификаций, созданных средствами среднего CASE). Они содержат системные словари и графические средства, исключающие необходимость разработки физических спецификации. Имеются системные спецификации, которые непосредственно переводятся в программные коды разрабатываемой системы (при этом автоматически генерируется до 80-90% кодов). На эти средства возложены также функции тестирования, управления конфигурацией, формирования документации. Главными преимуществами нижних CASE являются: значительное уменьшение времени на разработку, облегчение модификации. поддержка возможностей прототипирования (совместно со средними CASE).

6.2. Каскадная схема разработки ПО

Стандарт ISO/IEC 12207 не предлагает конкретную модель ЖЦ и методы разработки ПО. Его регламенты являются общими для любых моделей ЖЦ, методологий и технологий разработки. Стандарт ISO/IEC 12207 описывает структуру

процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.

Под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении ЖЦ. Модель ЖЦ зависит от специфики ИС и специфики условий, в которых система создается и функционирует. К настоящему времени наибольшее распространение получили следующие две основные модели ЖЦ: каскадная модель (1970-1985 гг.) и спиральная модель (1986-1990 гг.).

В изначально существовавших однородных ИС приложения представляли собой единое целое. Для разработки такого типа приложений применялся *каскадный способ*. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем (рис. 6.1).

Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Преимущества применения каскадного способа заключаются в следующем [5]:

- на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
- выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении ИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем чтобы предоставить разработчикам свободу реализовать их технически как можно лучше. В эту категорию попадают сложные расчетные системы, системы реального времени и др.

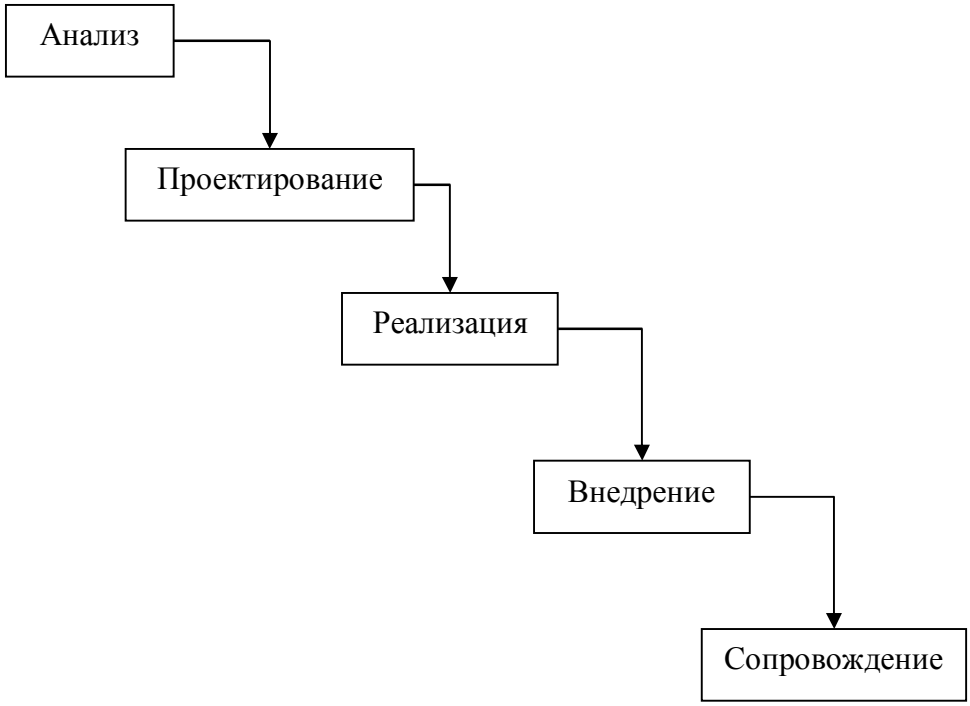


Рис. 6.1. Каскадная схема разработки ПО

В то же время этот подход обладает рядом недостатков, вызванных прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему, постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимал вид, представленный на рис. 6.2.

Изображенную на рис. 6.2 схему часто относят к отдельной модели, так называемой «модели с промежуточным контролем», которой межэтапные корректировки обеспечивают большую надежность по сравнению с каскадной моделью, хотя и увеличивают весь период разработки.

Основным недостатком каскадного подхода является существенное запаздывание с получением результатов. Согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к ИС «заморожены» в виде технического задания на все время ее создания. Таким образом, пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания ПО пользователи получают систему, не удовлетворяющую их потребностям. Модели (как функциональные, так и информационные) автоматизируемого объекта могут устареть одновременно с их утверждением.

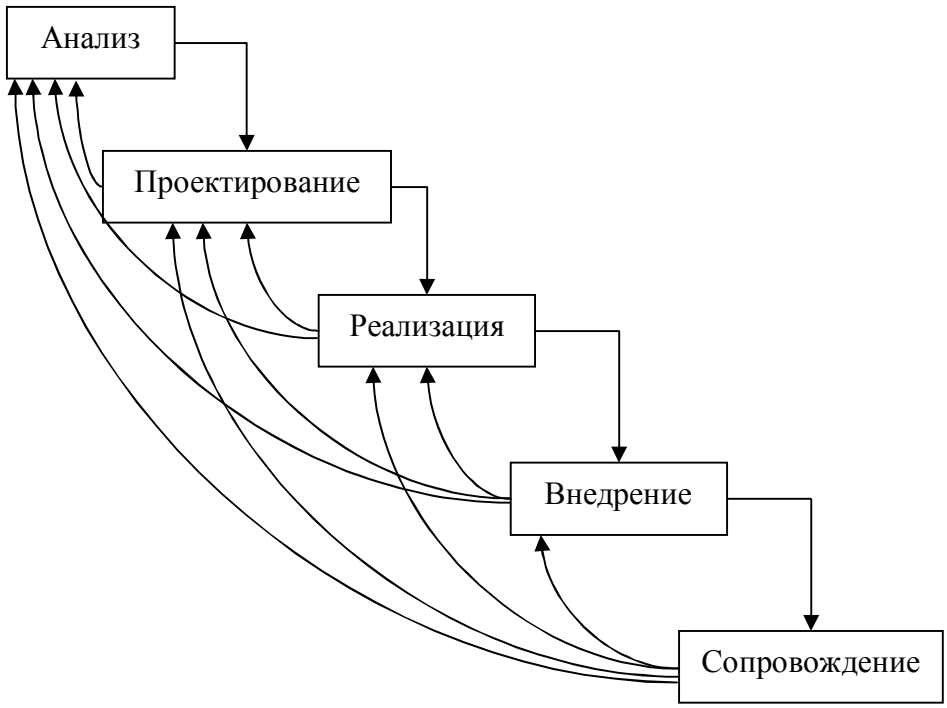


Рис. 6.2. Схема реального процесса разработки ПО

6.3. Спиральная модель проектирования

Для преодоления перечисленных проблем была предложена спиральная модель ЖЦ [15] (рис. 6.3), в которой делается упор на начальные этапы ЖЦ: анализ и проектирование. На этих этапах реализуемость технических решений проверяется путем создания прототипов. Каждый виток спирали соответствует созданию фрагмента проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача – как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

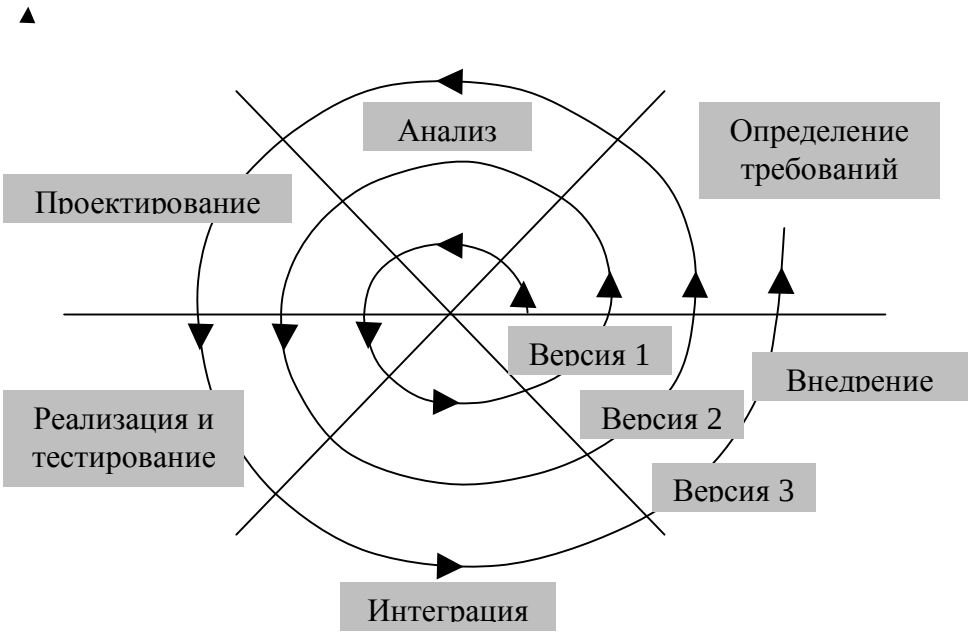


Рис. 6.3. Спиральная модель ЖЦ

Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

6.4. Методологии и технологии проектирования ИС

Общие требования к методологии и технологии

Методологии, технологии и инструментальные средства проектирования (CASE-средства) составляют основу проекта любой ИС. Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают выполнение процессов ЖЦ.

Технология проектирования определяется как совокупность трех составляющих:

- пошаговой процедуры, определяющей последовательность технологических операций проектирования (рис. 6.4);
- критериев и правил, используемых для оценки результатов выполнения технологических операций;

- нотаций (графических и текстовых средств), используемых для описания проектируемой системы.

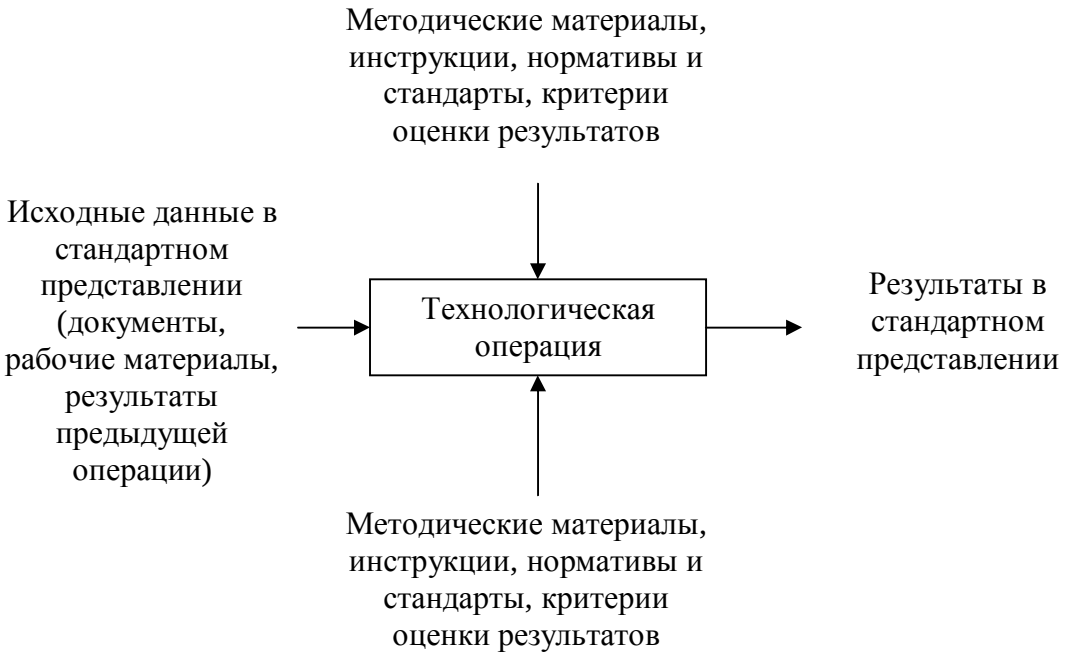


Рис. 6.4. Схема технологической операции проектирования

Технология проектирования, разработки и сопровождения ИС должна удовлетворять следующим требованиям:

- поддержка полного ЖЦ ПО;
- гарантированное достижение целей разработки ИС с заданным качеством и в установленное время;
- возможность выполнения крупных проектов в виде подсистем (возможность декомпозиции проекта на составные части, разрабатываемые группами исполнителей ограниченной численности с последующей интеграцией составных частей). Опыт разработки крупных ИС показывает, что для повышения эффективности работ необходимо разбить проект на отдельные слабо связанные по данным и функциям подсистемы. Реализация подсистем должна выполняться отдельными группами специалистов. При этом необходимо обеспечить координацию ведения общего проекта и исключить дублирование результатов работ каждой проектной группы, которое может возникнуть в силу наличия общих данных и функций;

- возможность ведения работ по проектированию отдельных подсистем небольшими группами (3-7 человек). Это обусловлено принципами управляемости коллектива и повышения производительности за счет минимизации числа внешних связей;
- минимальное время получения работоспособной ИС. Речь идет не о сроках готовности всей ИС, а о сроках реализации отдельных подсистем. Реализация ИС в целом в короткие сроки может потребовать привлечения большого числа разработчиков, при этом эффект может оказаться ниже, чем при реализации в более короткие сроки отдельных подсистем меньшим числом разработчиков. Практика показывает, что даже при наличии полностью завершенного проекта внедрение идет последовательно по отдельным подсистемам;
- возможность управления конфигурацией проекта, ведения версий проекта и его составляющих, возможность автоматического выпуска проектной документации и синхронизации ее версий с версиями проекта;
- независимость выполняемых проектных решений от средств реализации ИС (систем управления базами данных (СУБД), операционных систем, языков и систем программирования).

Реальное применение любой технологии проектирования, разработки и сопровождения ИС в конкретной организации и конкретном проекте невозможно без выработки ряда стандартов (правил, соглашений), которые должны соблюдаться всеми участниками проекта. К таким стандартам относятся следующие:

- стандарт проектирования;
- стандарт оформления проектной документации;
- стандарт интерфейса пользователя.

Перечисленные стандарты устанавливают определенные требования.

Стандарт проектирования:

- набор необходимых моделей (диаграмм) на каждой стадии проектирования и степень их детализации;
- правила фиксации проектных решений на диаграммах, в том числе правила именования объектов (включая соглашения по терминологии), набора атрибутов для всех объектов и правила их заполнения на каждой стадии, правила оформления диаграмм (включая требования к форме и размерам объектов) и т.д.;

- требования к конфигурации рабочих мест разработчиков, включая настройки операционной системы, настройки CASE-средств, общие настройки проекта и т.д.;
- механизм обеспечения совместной работы над проектом, в том числе правила интеграции подсистем проекта, правила поддержания проекта в одинаковом для всех разработчиков состоянии (регламент обмена проектной информацией, механизм фиксации общих объектов и т.д.), правила анализа проектных решений на непротиворечивость и т.д.

Стандарт оформления проектной документации:

- комплектность, состав и структура документации на каждой стадии проектирования;
- требования к ее оформлению (включая требования к содержанию разделов, подразделов, пунктов, таблиц и т.д.);
- правила подготовки, рассмотрения, согласования и утверждения документации с указанием предельных сроков для каждой стадии;
- требования к настройке издательской системы, используемой в качестве встроенного средства подготовки документации;
- требования к настройке CASE-средств для обеспечения подготовки документации в соответствии с установленными правилами.

Стандарт интерфейса пользователя:

- правила оформления экранов (шрифты и цветовая палитра), состав и расположение окон и элементов управления;
- правила использования клавиатуры и мыши;
- правила оформление текстов помощи;
- правила стандартных сообщений;
- правила обработки реакции пользователя.

Методология RAD

Один из подходов к разработки ПО в рамках спиральной модели ЖЦ – получившая широкое распространение методология быстрой разработки приложений RAD (Rapid Application Development), она включает в себя три составляющие:

- небольшую команду программистов (от 2 до 10 человек);

- короткий, но тщательно проработанный производственный график (от 2 до 6 мес.);
- повторяющийся цикл, при котором разработчики по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Команда разработчиков должна представлять собой группу профессионалов, имеющих опыт в анализе, проектировании, генерации кода и тестировании ПО с использованием CASE-средств, способных хорошо взаимодействовать с конечными пользователями и трансформировать их предложения в рабочие прототипы.

Жизненный цикл ПО по методологии RAD состоит из четырех фаз: анализа и планирования требований; проектирования; построения; внедрения.

На фазе *анализа и планирования требований* пользователи системы определяют функции, которые она должна выполнять, выделяют наиболее приоритетные из них, требующие проработки в первую очередь, описывают информационные потребности. Формулирование требований к системе осуществляется в основном силами пользователей под руководством специалистов-разработчиков. Ограничивается масштаб проекта, устанавливаются временные рамки для каждой из последующих фаз. Кроме того, определяется сама возможность реализации проекта в заданных размерах финансирования, на имеющихся аппаратных средствах и т.п. Результатом фазы должны быть список расставленных по приоритету функций будущей ИС, предварительные функциональные и информационные модели ИС.

На фазе *проектирования* часть пользователей принимает участие в техническом проектировании системы под руководством специалистов-разработчиков. CASE-средства используются для быстрого получения работающих прототипов приложений. Пользователи, непосредственно взаимодействуя с ними, уточняют и дополняют требования к системе, которые не были выявлены на предыдущей фазе. Более подробно рассматриваются процессы системы. Анализируется и при необходимости корректируется функциональная модель. Каждый процесс рассматривается детально. Если требуется, для каждого элементарного процесса создается частичный прототип: экран, диалог, отчет, устраняющий неясности или неоднозначности. Устанавливаются требования разграничения доступа к данным. На этой же фазе происходит определение необходимой документации.

После детального определения состава процессов оценивается количество функциональных элементов разрабатываемой системы и принимается решение о разделении ИС на подсистемы, поддающиеся реализации одной командой

разработчиков за приемлемое для RAD-проектов время (60-90 дней). С использованием CASE-средств проект распределяется между различными командами (делится функциональная модель). Результатом данной фазы должны быть:

- общая информационная модель системы;
- функциональные модели системы в целом и подсистем, реализуемых отдельными командами разработчиков;
- точно определенные с помощью CASE-средств интерфейсы между автономно разрабатываемыми подсистемами;
- построенные прототипы экранов, отчетов, диалогов.

На фазе *построения* выполняется непосредственно сама быстрая разработка приложения. На данной фазе разработчики производят итеративное построение реальной системы на основе полученных в предыдущей фазе моделей, а также требований нефункционального характера. Программный код частично формируется при помощи автоматических генераторов, получающих информацию из репозитория CASE-средств. Конечные пользователи на этой фазе оценивают получаемые результаты и вносят коррективы, если в процессе разработки система перестает удовлетворять определенным ранее требованиям. Тестирование системы осуществляется в процессе разработки.

После окончания работ каждой отдельной команды разработчиков производится постепенная интеграция данной части системы с остальными, формируется полный программный код, выполняется тестирование совместной работы данной части приложения, а затем тестирование системы в целом. Завершается физическое проектирование системы:

- определяется необходимость распределения данных;
- осуществляется анализ использования данных;
- производится физическое проектирование базы данных;
- определяются требования к аппаратным ресурсам;
- определяются способы увеличения производительности;
- завершается разработка документации проекта.

Результатом фазы является готовая система, удовлетворяющая всем согласованным требованиям.

На фазе *внедрения* производятся обучение пользователей, организационные изменения и параллельно с внедрением новой системы осуществляется работа с существующей системой (до полного внедрения новой). Так как фаза построения достаточно непродолжительна, планирование и подготовка к внедрению должны

начинаться заранее, как правило, на этапе проектирования системы. Приведенная схема разработки ИС не является абсолютной. Возможны различные варианты, зависящие, например, от начальных условий, в которых ведется разработка: разрабатывается совершенно новая система; уже было проведено обследование предприятия и существует модель его деятельности; на предприятии уже существует некоторая ИС, которая может быть использована в качестве начального прототипа или должна быть интегрирована с разрабатываемой.

Следует, однако, отметить, что методология RAD, как и любая другая, не может претендовать на универсальность, она хороша в первую очередь для относительно небольших проектов, разрабатываемых для конкретного заказчика. Если же разрабатывается типовая система, которая не является законченным продуктом, а представляет собой комплекс типовых компонентов, централизованно сопровождаемых, адаптируемых к программно-техническим платформам, СУБД, средствам телекоммуникации, организационно-экономическим особенностям объектов внедрения и интегрируемых с существующими разработками, на первый план выступают такие показатели проекта, как управляемость и качество, которые могут войти в противоречие с простотой и скоростью разработки. Для таких проектов необходимы высокий уровень планирования и жестка дисциплина проектирования, строгое следование заранее разработанным протоколам и интерфейсам, что снижает скорость разработки.

Методология RAD неприменима для построения сложных расчетных программ, операционных систем или программ управления космическими кораблями, т.е. программ, содержащих большой объем (сотни тысяч строк) уникального кода.

Не подходят для разработки по методологии RAD приложения, в которых отсутствует ярко выраженная интерфейсная часть, наглядно определяющая логику работы системы (например, приложения реального времени), и приложения, от которых зависит безопасность людей (например, управление самолетом или атомной электростанцией), так как итеративный подход предполагает, что первые несколько версий наверняка не будут полностью работоспособны, что в данном случае исключается.

Оценка размера приложений производится на основе так называемых функциональных элементов (экраны, сообщения, отчеты, файлы и т.п.). Подобная метрика не зависит от языка программирования, на котором ведется разработка. Размер приложения, которое может быть выполнено по методологии RAD, для хорошо

отлаженной среды разработки ИС с максимальным повторным использованием программных компонентов определяется следующим образом:

- менее 1000 функциональных элементов – один человек;
- 1000-4000 функциональных элементов – одна команда разработчиков;
- более 4000 функциональных элементов – 4000 функциональных элементов на одну команду разработчиков.

Итак, перечислим основные принципы методологии RAD:

- разработка приложений итерациями;
- необязательность полного завершения работ на каждом из этапов жизненного цикла;
- обязательность вовлечения пользователей в процесс разработки ИС;
- необходимость применения CASE-средств, обеспечивающих целостность проекта;
- применение средств управления конфигурацией, облегчающих внесение изменений в проект и сопровождение готовой системы;
- необходимость использования генераторов кода;
- использование прототипирования, позволяющее полнее выяснить и удовлетворить потребности конечного пользователя;
- тестирование и развитие проекта, осуществляемые одновременно с разработкой;
- ведение разработки немногочисленной хорошо управляемой командой профессионалов;
- грамотное руководство разработкой системы, четкое планирование и контроль выполнения работ.

6.5 Консалтинг в области информационных технологий

Термин *консалтинг* в России является каким-то аморфным и неконкретным, Практически каждая фирма, работающая на рынке информационных технологий, заявляет о предоставлении ею неких консалтинговых услуг. Что же следует понимать под консалтингом на самом деле? Какие требования предъявляются к консалтинговым структурам? Постараемся ответить на эти вопросы.

Консалтинг - это деятельность специалиста или целой фирмы, занимающихся стратегическим планированием проекта, анализом и формализацией требований к информационной системе, созданием системного проекта, иногда - проектированием приложений. Но все это до этапа собственно программирования или настройки каких-то уже имеющихся комплексных систем управления предприятием, выбор которых и осуществляется на основе системного проекта. И уж ни в коей мере сюда не входит системная интеграция. Консалтинг предваряет и регламентирует названные этапы.

Фактически консультантом выполняется два вида работ. Прежде всего это элементарное наведение порядка в организации: бизнес-анализ и реструктуризация (реинжиниринг бизнес-процессов). Это направление получило название "бизнес-консалтинг". В конечном итоге речь, разумеется, идет об автоматизации, однако если мы будем автоматизировать существующий хаос, царящий на российских предприятиях, то в итоге получим не что иное как "автоматизированный хаос".

Любая организация - это довольно сложный организм, функционирование которого одному человеку понять просто невозможно. Руководство в общих чертах представляет себе общий ход дел, а клерк досконально изучил только свою деятельность, уяснил свою роль в сложившейся системе деловых взаимоотношений. Но как организация функционирует в целом, не знает, как правило, никто. И именно деятельность, направленная на то, чтобы разобраться в функционировании таких организмов, построить соответствующие модели и на их основе выдвинуть некоторые предложения по поводу улучшения работы некоторых звеньев, а еще лучше - бизнес-процессов (деятельностей, имеющих ценность для клиента) считается бизнес-консалтингом.

Другой вид работ - собственно системный анализ и проектирование- Выявление и согласование требований заказчика приводит к пониманию того, что же в действительности необходимо сделать. За этим следует проектирование/выбор готовой системы, которая как можно в большей степени удовлетворяла требованиям заказчика.

Кроме того важный элемент консалтинга - формирование и обучение рабочих групп. Речь идет не только о традиционной учебе, но и об участии сотрудников заказчика в проекте с самых ранних стадий. Тогда по окончании работ они будут способны анализировать и улучшать заложенные в системе бизнес-процессы в рамках своей организации.

Цели и этапы разработки консалтинговых проектов

Основными целями разработки консалтинговых проектов являются:

• представление деятельности предприятия и принятых в нем технологий в виде иерархии диаграмм, обеспечивающих наглядность и полноту их отображения;

• формирование на основании анализа предложений по реорганизации организационно-управленческой структуры;

• упорядочивание информационных потоков (в том числе документооборота) внутри предприятия;

• выработка рекомендаций по построению рациональных технологий работы подразделений предприятия и его взаимодействию с внешним миром;

• анализ требований и проектирование спецификаций корпоративных информационных систем;

• рекомендации и предложения по применимости и внедрению существующих систем управления предприятиями (прежде всего классов MRP - manufacturing resource planning ERP - enterprise resource planning).

Структура подхода к разработке консалтинговых проектов приведена на рис. 6.5. Опишем перечисленные этапы более подробно.

1. Анализ первичных требований и планирование работ

Данный этап предваряет инициацию работ над проектом. Его основными задачами являются: анализ первичных бизнес-требований, предварительная экономическая оценка проекта, построение план-графика выполнения работ, создание и обучение совместной рабочей группы.

2. Проведение обследования деятельности предприятия

В рамках данного этапа осуществляется:

• предварительное выявление требований, предъявляемых к будущей системе;

• определение оргштатной и топологической структур предприятия;

• определение перечня целевых задач (функций) предприятия;

• анализ распределения функций по подразделениям и сотрудникам;

• определение перечня применяемых на предприятии средств автоматизации.

При этом выявляются функциональные деятельности каждого из ' подразделений предприятия и функциональные взаимодействия между ними, информационные потоки внутри подразделений и между ними, внешние по отношению к предприятию объекты и внешние информационные взаимодействия.

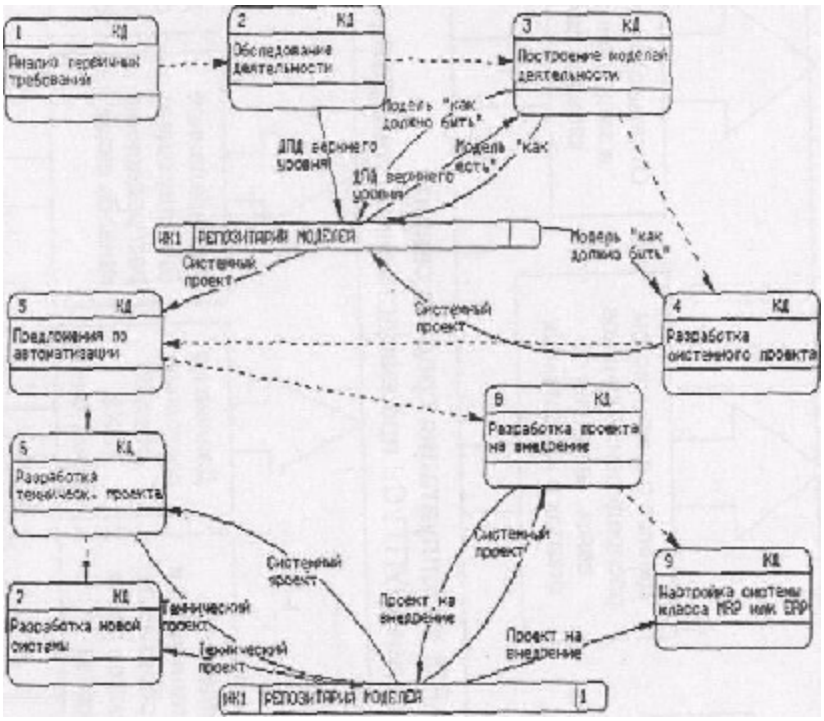


Рис. 6.5I. Структура подхода

В качестве исходной информации при проведении обследования и выполнении дальнейших этапов служат:

• .данные по оргштатной структуре предприятия;

• информация о принятых технологиях деятельности;

• стратегические цели и перспективы развития;

• результаты интервьюирования сотрудников (от руководителей до исполнителей нижнего звена);

• предложения сотрудников по усовершенствованию бизнес-процессов предприятия;

• нормативно-справочная документация;

• опыт системных аналитиков в части наличия типовых решений.

Длительность обследования составляет 1-2 недели. По окончании обследования строится и согласуется с заказчиком предварительный вариант функциональной модели предприятия, включающей идентификацию внешних объектов и информационных взаимодействий с ними, а также детализацию до уровня основных деятельностей предприятия и информационных связей между этими деятельностями.

3. Построение моделей деятельности предприятия

На данном этапе осуществляется обработка результатов обследования и построение моделей деятельности предприятия следующих двух видов:

• *модели "как есть",* представляющей собой "снимок" положения дел на предприятии (оргштатная структура, взаимодействия подразделений, принятые технологии, автоматизированные и неавтоматизированные бизнес-процессы и т.д.) на момент обследования и позволяющей понять, что делает и как функционирует данное предприятие с позиций системного анализа, а также на основе автоматической верификации выявить ряд ошибок и узких мест и сформулировать ряд предложений по улучшению ситуации;

• *модели "как должно быть",* интегрирующей перспективные предложения руководства и сотрудников предприятия, экспертов и системных аналитиков и позволяющей сформировать видение новых рациональных технологий работы предприятия.

Каждая из моделей включает в себя полную структурную функциональную модель деятельности (например, в виде иерархии диаграмм потоков данных с разработанными для всех процессов нижнего уровня подробными их спецификациями на структурированном естественном языке или в виде иерархии ЗАОТ-диаграмм), информационную модель (как правило, с использованием нотации "сущность-связь"), а также, в случае необходимости, событийную (описывающую поведение) модель (с использованием диаграмм переходов состояний).

Переход от модели "как есть" к модели "как должно быть" осуществляется следующими двумя способами.

1) Совершенствование технологий на основе оценки ид эффективности. При этом критериями оценки являются стоимостные и временные затраты выполнения бизнес-процессов, дублирование и противоречивость выполнения отдельных задач бизнес-процесса, степень загруженности сотрудников ("легкий" реинжиниринг).

2) Радикальное изменение технологии и переосмысление бизнес-проиессов ("жестким" ремнжиниринг). Например, вместо попыток улучшения бизнес-процесса проверки кредитоспособности клиента, может быть следует задуматься, а нужна ли вообще такая проверка?

Возможно затраты на такие проверки каждого из клиентов во много раз превышают убытки, которые может понести компания в отдельных случаях недобросовестности (а случае, когда клиентов много, а суммы закупок незначительны).

Построенные модели являются не просто реализацией начальных этапов разработки системы и техническим заданием на последующие этапы. Они представляют собой самостоятельный отделяемый результат, имеющий большое практическое значение, в частности:

1) Модель "как есть" включает в себя существующие неавтоматизированные технологии, работающие на предприятии. Формальный анализ этой модели позволит выявить узкие места в технологиях и предложить рекомендации по их улучшению (независимо от того, предполагается на данном этапе автоматизация предприятия или нет).

2) Она позволяет осуществлять автоматизированное и быстрое обучение новых работников конкретному направлению деятельности предприятия (так как ее технология содержится в модели) с использованием диаграмм (известно, что "одна картинка стоит тысячи слов"),

3) С ее помощью можно осуществлять предварительное моделирование нового направления деятельности с целью выявления новых потоков данных, взаимодействуюших подсистем и бизнес-процессов.

4.Разработка системного проекта

Данный этап является первой фазой разработки собственно системы автоматизации (именно, фазой анализа требований к системе), на которой требования заказчика уточняются, формализуются и документируются. Фактически на этом этапе дается ответ на вопрос: "Что должна делать будущая система?". Именно здесь лежит ключ к успеху всего проекта автоматизации- В практике создания больших программных систем известно немало примеров неудачной реализации именно из-за неполноты и нечеткости определения системных требований.

На этом этапе определяются:

- архитектура системы, ее функции, внешние условия ее функционирования, распределение функций между аппаратной и программной частями;
- интерфейсы и распределение функций между человеком и системой;
- требования к программным и информационным компонентам системы, необходимые аппаратные ресурсы, требования к базе данных, физические характеристики компонент системы, ж интерфейсы;
- состав людей и работ, имеющих отношение к системе;
- ограничения в процессе разработки (директивные сроки завершения отдельных этапов, имеющиеся ресурсы, организационные процедуры и мероприятия, обеспечивающие защиту информации).

Системный проект строится на основе модели "как должно быть" и включает функциональную модель будущей системы в соответствии с одним из общеупотребительных стандартов (например, 1DEFO или IDEF3), информационную модель, например, в соответствии со стандартом IDEF1X, а также техническое задание на создание автоматизированной системы (например, в соответствии с ГОСТ 34.602-S9).

По завершении данного этапа (после согласования системного проекта с заказчиком) изменяется роль консультанта. Отныне он как бы становится на сторону заказчика, и одной из его основных функций на всех последующих этапах работ будет являться контроль на соответствие требованиям, зафиксированным а системном проекте.

Необходимо отметить следующее достоинство системного проекта. Для традиционной разработки характерно осуществление начальных этапов кустарными неформализованными способами- В результате заказчики и пользователи впервые могут увидеть систему после того, как она уже в большей степени реализована. Естественно, эта система отличается от того, что они ожидали увидеть. Поэтому далее следует еще несколько итераций ее разработки или модификации, что требует дополнительных (и значительных) затрат денег и времени. Ключ к решению этой проблемы и дает системный проект, позволяющий:

• описать, "увидеть" и скорректировать будущую систему до того, как она будет реализована физически;

• уменьшить затраты на разработку и внедрение системы;

• оценить разработку по времени и результатам;

• достичь взаимопонимания между всеми участниками работы (заказчиками, пользователями, разработчиками, программистами и т.д.);

• улучшить качество разрабатываемой системы, а именно: создать оптимальную структуру интегрированной базы данных, выполнить функциональную декомпозицию типовых модулей.

Системный проект полностью независим и отделяем от конкретных разработчиков, не требует сопровождения его создателями и может быть безболезненно передан другим лицам. Более того, если по каким-либо причинам предприятие не готово к реализации на основе проекта, он может быть положен "на полку" до тех пор, пока в нем не возникнет необходимость. Кроме того, его можно использовать для самостоятельной разработки или корректировки уже реализованных на его основе программных средств силами программистов отдела автоматизации предприятия.

5. Разработка предложений по автоматизации предприятия

На основании системного проекта осуществляется:

• составление перечня автоматизированных рабочих мест предприятия и способов взаимодействия между ними;

• анализ применимости существующих систем управления предприятиями для решения требуемых задач и формирование рекомендаций по выбору такой системы;

• совместное с заказчиком принятие решения о выборе конкретной системы управления предприятием или разработке собственной системы;

• разработка требований к техническим средствам;

• разработка требований к программы» средствам;

• разработка предложений по этапам и срокам автоматизации.

6. Разработка технического проекта

На данном этапе на основе системного проекта и принятых решений по автоматизации осуществляется проектирование системы. Фактически здесь дается ответ на вопрос: "Как (каким образом) мы будем строится систему, чтобы она удовлетворяла предъявленным *к* ней требованиям?". Этот этап разделяется на два подэтапа:

• проектирование архитектуры системы, включающее разработку структуры и интерфейсов ее компонент (автоматизированных рабочих мест), согласование функций и технических требований к компонент там, определение информационных потоков между основными компонентами, связей между ними и внешними объектами;

• детальное проектирование, включающее разработку спецификаций \ каждой компоненты,

разработку требований к тестам и плана интеграции компонент, а также построение моделей иерархии программных модулей и межмодульных взаимодействий и проектирование внутренней структуры модулей.

При этом происходит расширение системного проекта:

• за счет его уточнения;

• за счет построения моделей автоматизированных рабочих мест, включающих подсхемы информационной модели и функциональные модели, ориентированные на эти подсхемы вплоть до идентификации конкретных сущностей информационной модели;

• за счет построения моделей межмодульных и внутримодульных взаимодействий с использованием техники структурных карт-

7. Последующие этапы

В случае разработки собственной системы последующие этапы являются традиционными: по спецификациям технического проекта осуществляется программирование модулей, их тестирование и отладка и после-) дующая комплексация в автоматизированные рабочие места и в систему в целом. При этом схема интегрированной базы данных, как правило, генерируется автоматически на основании информационной модели.

Настройка существующей системы MRP млн ERP осуществляется • по следующим этапам:

• наполнение системы фактическими данными;

• построение процедур их обработки;

• интеграция процедур внутри автоматизированных рабочих мест,

• интеграция автоматизированных рабочих мест в систему.

CASE-технологии - методологическая и инструментальная база консалтинга

За последнее десятилетие сформировалось новое направление в программотехнике - CASE (Computer-Aided Software/System Engineering). В настоящее время не существует общепринятого определения CASE. Содержание этого понятия обычно определяется перечнем задач, решаемых с помощью CASE. а также совокупностью применяемых методов и средств. Очень грубо, CASE •-технология представляет собой совокупность методологий анализа, проектирования, разработки и сопровождения сложных систем программного обеспечения (ПО), поддержанную комплексом взаимоувязанных средств автоматизации, CASE - это инструментарий для системных аналитиков, разработчиков и программистов, заменяющий им бумагу и карандаш на компьютер для автоматизации процесса проектирования и разработки ПО.

К настоящему моменту дисциплина CASE оформилась в самостоятельное наукоемкое направление в программотехнике, повлекшее за собой образование мощной CASE-индустрии, объединившей сотни фирм и компаний различной ориентации. Среди них выделяются компании • разработчики средств анализа и проектирования ПО с широкой сетью дистрибьютерских и дилерских фирм; фирмы - разработчики специальных средств с ориентацией на узкие предметные области или на отдельные этапы жизненного цикла ПО; обучающие фирмы, которые организуют семинары и курсы подготовки специалистов; консультационные фирмы, оказывающие практическую помощь при использовании CASE-пакетов для разработки конкретных приложений; фирмы, специализирующиеся на выпуске периодических журналов и бюллетеней по CASE. Основными покупателями CASE-пакетов за рубежом являются военные организации, центры обработки данных и коммерческие фирмы но разработке ПО.

Существует мнение, что CASE является наиболее перспективным направлением в программотехнике. С этим, естественно, можно и нужно спорить, но то. что CASE - наиболее бурно и интенсивно развиваемое направление, является в настоящее время фактом. Практически ни один серьезный зарубежный программный проект не осуществляется без использования CASE-средств. Известная методология структурного системного анализа SADT (точнее ее подмножество 1DEFO) принята а качестве стандарта на разработку ПО Министерством обороны США. Более того, среди менеджеров и руководителей компьютерных фирм считается чуть ли не правилом хорошего тона знать основы SADT и при обсуждении каких-либо вопросов нарисовать простейшую диаграмму, поясняющую суть дела.

CASE позволяет не только создавать "правильные" продукты, но и обеспечить "правильный" процесс их создания. Основная цель CASE состоит в том, чтобы отделить проектирование ПО от его кодирования и последующих этапов разработки, а также скрыть от разработчиков все детали среды разработки и функционирования ПО- Чем больше деятельности будет вынесено в проектирование из кодирования, тем лучше-

При использовании CASE-технологий изменяются все этапы жизненного цикла программной системы, при этом наибольшие изменения касаются этапов анализа и проектирования- В большинстве современных CASE-систем применяются методологии структурного анализа и проектирования, основанные на наглядных диаграммные техниках, при этом для описания модели проектируемой системы используются графы, диаграммы, таблицы и схемы. Такие методологии обеспечивают строгое и наглядное описание проектируемой системы, которое начинается с ее общего Обзора и затем детализируется, приобретая иерархическую структуру со все большим числом уровней.

Несмотря на то, что структурные методологии зарождались как средства анализа и проектирования ПО, сфера их применений в настоящее время выходит далеко за рамки названной предметной области. Поэтому CASE-технологии успешно применяются для моделирования практически всех предметных областей, однако устойчивое положение они занимают в следуюших областях:

• бизнес-анализ (фактически, модели деятельности предприятий "как есть" и "как должно быть" строятся с применением методов структурного системного анализа и поддерживающих их CASE-средств);

• системный анализ и проектирование (практически любая современная крупная программная система разрабатывается с применением CASE-технологий по крайней мере на этапах анализа и проектирования, что связано с большой сложностью данной проблематики и со стремлением повысить эффективность работ).

Следует отметить, что CASE - не революция в программотехнике, а результат естественного эволюционного развития всей отрасли средств, называемых ранее инструментальными или технологическими. Однако это и не *Confuse Array of Software that does Everything,* существует ряд признаков и свойств, наличие которых позволяет классифицировать некоторый продукт как CASE-средство. Одним из ключевых признаков является поддержка методологий структурного системного анализа и проектирования.

С самого начала CASE-технологии развивались с целью преодоления ограничений при использовании структурных методологий проектирования 60-70-х годов (сложности понимания, большой трудоемкости и стоимости использования, трудности внесения изменений в проектные спецификации и т.д.) за счет их автоматизации и интеграции поддерживающих средств. Таким образом, CASE-техиологии, вообще говоря, не могут считаться самостоятельными методологиями, они только развивают структурные методологии и делают более эффективным их применение за счет автоматизации.

Помимо автоматизации структурных методологий и. как следствие, возможности применения современных методов системной и программной инженерии, CASE обладают следующими основными достоинствами:

• улучшают качество создаваемого ПО за счет средств автоматического контроля (прежде всего, контроля проекта);

• позволяют за короткое время создавать прототип будущей системы, что позволяет на ранних этапах оценить ожидаемый результат;

• ускоряют процесс проектирования и разработки;

• освобождают разработчика от рутинной работы, позволяя ему целиком сосредоточиться на творческой части разработки;

• поддерживают развитие и сопровождение разработки;

• поддерживают технологии повторного использования компонент разработки.

Большинство CASE-средств основано на парадигме методология/метод/нотация/средство. Методология определяет руководящие указания для оценки и выбора проекта разрабатываемого ПО, шаги работы и их последовательность, а также правила распределения и назначения методов. Метод - это систематическая процедура или техника генерации описаний компонент ПО (например, проектирование потоков и структур данных). Нотации предназначены для описания структуры системы, элементов данных, этапов обработки и включают графы, диаграммы, таблицы, блок-схемы, формальные и естественные языки. Средства - инструментарий для поддержки и усиления методов. Эти инструменты поддерживают работу пользователей при создании и редактировании графического проекта в интерактивном режиме, они способствуют организации проекта в виде иерархии уровней абстракции, выполняют проверки соответствия компонентов.

7. Интеллектуальные технологии обработки информации в корпоративных сетях.

7.1 Интеллект баз данных: Активные базы данных.

При решении многих обсуждаемых проблем, связанных с хранилищами данных, репозиториями и т.д., так же как и при определении общей архитектуры приложений, можно извлечь пользу из свойств активных баз данных.

Свойства активных баз данных заключаются в том, что процедурные элементы общей среды встраиваются в систему базы данных и управляются декларативным образом. Развитие технологии активных баз данных рассматривается в настоящее время как одна из главных тенденций, которая будет революционизировать разработку приложений. Философия, на которой основана эта технология - хранение операций над данными и процедур вместе с самими данными, - широко используется в других областях, например в объектно-ориентированных базах данных. Поэтому, если даже в ваших трех- или пятилетних планах не фигурировали заметным образом объектно-ориентированные базы данных, вы можете тем не менее добиться получения конкретной пользы от них благодаря включению конструкций активных баз данных в ваши среды.

Введение

Традиционные базы данных являются пассивными. Объекты данных *обычно* помещаются в базу данных пользователем или приложением. Выборка объектов осуществляется опять-таки под воздействием внешних источников. Подобным же образом под влиянием какого-либо внешнего источника информация изменяет место своего хранения в базе данных (например, переносится из одной таблицы в другую). Бизнес-правила, применяемые к содержимому базы данных (например, правила обновления конкретного элемента данных, последующие за этим действия над другими элементами как результат такого обновления), также управляются некоторым внешним источником. Короче говоря, традиционные базы данных не являются активными "игроками" в информационных системах и вместо этого играют организационную роль, направленную на обеспечение хранения данных.

Рис. 16.1. Пассивная система базы данных

В последние годы эта роль изменяется, и важность концепции *активных баз данных будет* возрастать на протяжении оставшихся лет текущего десятилетия. Можно, по существу, утверждать, что технология активных баз данных - это врата, открывающие путь к базам знании, исследуемым в области искусственного интеллекта (ИИ). Действительно, в конце этой главы мы кратко обсудим некоторые прогнозы относительно взаимопроникновения искусственного интеллекта и баз данных.

Однако перед этим мы должны обсудить принципы, технологии и направления развития активных систем баз данных - от основных моделей триггеров и хранимых процедур до более сложных элементов технологии.

Главное различие между активными базами данных и традиционными пассивными базами данных заключается в конечном счете в том, что в системах последнего типа вся процедурная логика, включая выборку и модификацию данных, управляемых СУБД, координируется вне сферы управления данными. Если предполагается, что в результате выполнения определенной операции обновления данных (рассматриваемой как некоторое событие) должна вызываться какая-либо другая последовательность действий, выполнение этих других действий должно инициироваться логикой приложения или некоторыми иными внешними агентами. Напротив, среда активных баз данных поддерживает инициацию таких других действий и управление ими внутри среды базы данных в соответствии с предварительно установленными правилами и без необходимости получения каких-либо дальнейших управляющих воздействий от приложений или от каких-либо других внешних источников.

Наряду с основными принципами активных баз данных такие их составные элементы, как модели управления транзакциями, модели переходов состояний и техника использования более чем одного триггера на событие, помогут распространить возможности существующих в настоящее время технологий активных баз данных на сферу интеллектуальных баз данных.

Принципы активных систем баз данных

Как мы уже упоминали, системы баз данных традиционно были пассивными. На примере с магазином видеозаписей на рис. 16.1 приложения, пользователи, программное обеспечение операционной системы или некоторый другой внешний источник должны были уведомлять СУБД , каким образом хранить, осуществлять выборку или реорганизовывать информацию. Даже вне такой сферы, как операции случайного пользователя, мы имеем в данном случае тем не менее нагрузку по логике обработки содержимого базы данных, возлагаемую на такие сущности, как прикладные программы.

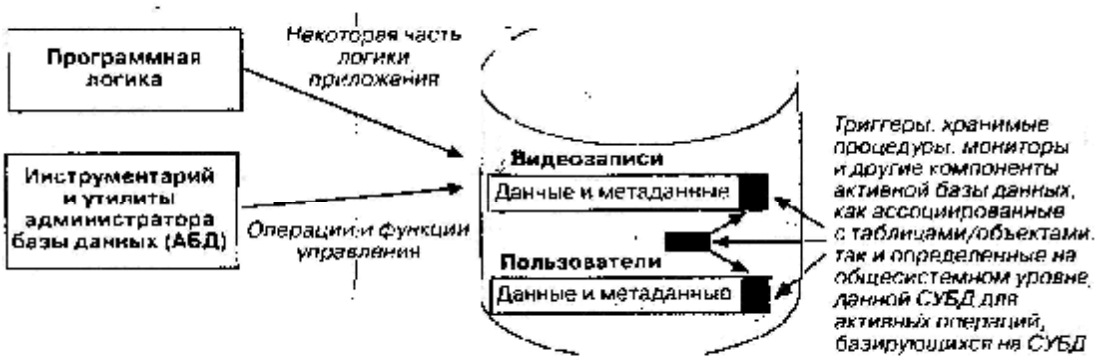


Рис. 16.2. Активная система базы данных

С другой стороны, *активные системы баз данных* предусматривают возможности позволяющие:

• содержать логику обработки (до некоторой степени) в самой базе данных так, чтобы она управлялась СУБД, а не прикладным программным обеспечением приложений;

• обеспечивать некоторую форму мониторинга событий и условий, которые воздействуют на данные и могут инициировать обработку, управляемую базой данных;

• включать в систему базы данных также некоторое средство, с помощью которого эти события и условия могли бы запускать логику внутри базы данных.

Как показано на рис. 16.2, все эти три возможности - логика, триггеры для этой логики и средства мониторинга для активизации триггеров - выносятся из программ приложений в саму базу данных, обеспечивая более тесную связь системных данных и операций над данными, чем это было принято в традиционных пассивных управляемых СУБД системах.

В создании активных сред помогает несколько основных конструкций базы

данных: ограничения, утверждения, хранимые процедуры и триггеры. Давайте обсудим каждую из них.

В традиционных пассивных базах данных для обработки сложных бизнес-правил, в которых результат некоторого действия вызывает исполнение одной или другой альтернативы, необходимы следующие шаги:

1. Клиентское приложение издает направленный серверу запрос на какую-либо операцию над базой данных, например, с помощью выполнения некоторого оператора UPDATE (обновить) языка SQL, который должен обновить значение указанного элемента данных.

2. Сервер базы данных выполняет запрашиваемую операцию и возвращает код состояния SUCCESS (удачно) клиентскому приложению. *Ограничения* являются относительно простыми конструкциями, имеющими вид от спецификации связей первичного и внешнего ключей, используемых в ограничениях целостности по ссылке* (Referential Integrity), до SQL-подобных ограничений CHECK, используемых для проверки принадлежности значения заданному диапазону или списку значений. Ограничения могли бы рассматриваться как первое средство для встраивания бизнес-правил в базу данных вместо использования для этой цели логики приложения.

Активная база данных может быть охарактеризована как система, следующая правилам Событие-Условие-Действие. Ключ к будущим активным системам баз данных состоит в том, чтобы иметь возможность выражать эти правила более перспективным декларативным образом, который допускает расширения в конструкциях искусственного интеллекта, например в механизмах вывода "от фактов к цели" (Forward Chaining) (т.е. путем следования множеству действий, основанных на правилах, и вывода некоторого типа результата из цепочки действий). Например, система Starburst ("Вспышка звезды") компании IBM поддерживает концепцию *переходов состояний* - переходов S -> S' - и возможность моделирования функционирования системы как последовательности таких переходов".
Система Starbrst включает также и другие базы данных, которые не только выполняют предопределенные правила, но создают новые правила, основываясь на некоторой логике транзакций, дезактивируют одно или более правил или изменяют приоритеты одного или более правил. По существу, среда активной базы данных должна допускать операции, основанные на правилах, не только относительно таких объектов, как таблицы и столбцы , но также и относительно самих правил.

Другая система - POSTGRES* - обеспечивает иные активные возможности, основанные на правилах. В ней предусматриваются, в частности, правило "Do *'Instead " (выполнить вместо)* и правило *"Do Refuse" (отвергнуть выполнение).* В соответствии с

правилом "Do Instead" если во время выполнения некоторой операции над базой данных имеет место определенное событие или состояние, то вместо нее будет выполняться другая заданная операция. Правило "Do Refuse" предписывает отвергнуть выполнение некоторой операции, если не удовлетворяется

заданное ограничение (например, не допускать возможности обновлять цену на видеозапись, если пользователь не является управляющим магазином либо помощником управляющего).

Модели транзакций и активные базы данных

Обсудим различные модели обработки транзакций, в том числе вложенные транзакции (транзакции, содержащиеся в других транзакциях). Такая модель, в частности, хороню подходит для активных баз данных, поскольку иерархическая структура модели вложенных транзакций позволяет значительно легче согласовывать связь между запускаемой с помощью триггера транзакцией и исполнением правил. Поскольку дочерние транзакции могут исполняться параллельно друг с другом, множество функций, обрабатывающих правила, может вызываться одновременно.

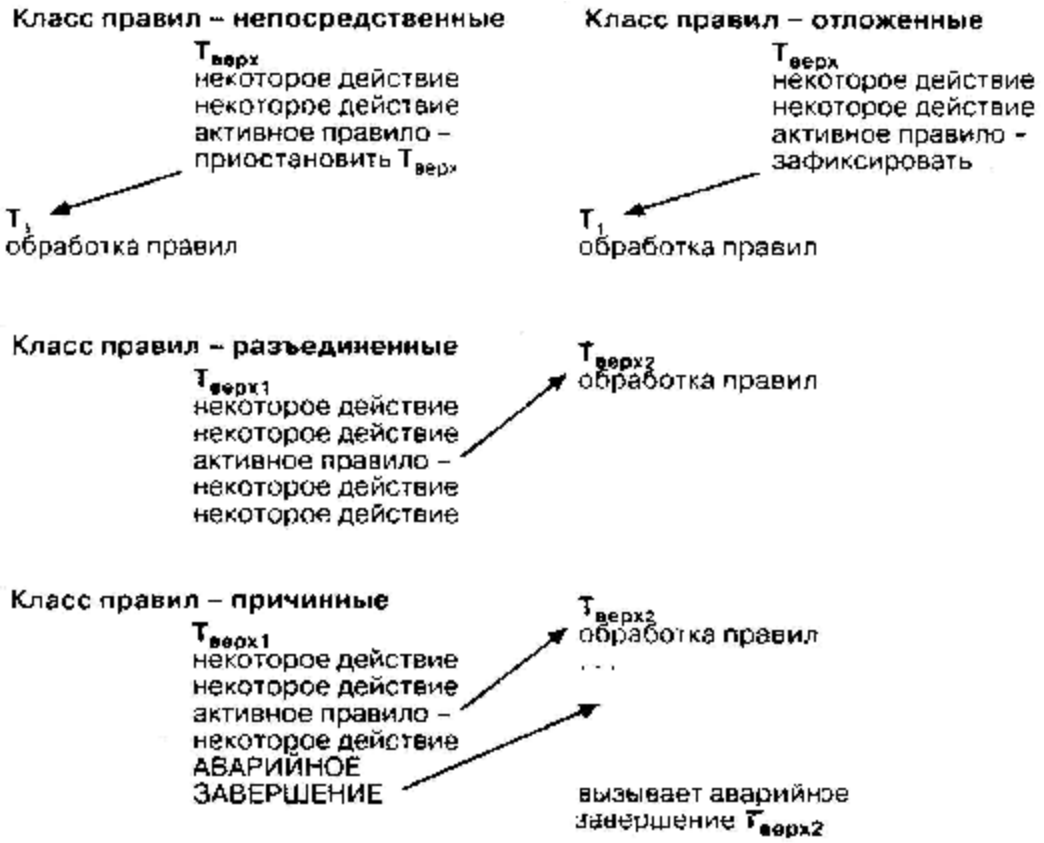


Рис. 16.4. Вложенные транзакции и активные базы данных

На рис. 16.4 показано использование модели вложенных транзакций в активной базе данных. Пользовательская транзакция исполняется как корневая (транзакция верхнего уровня) с множеством классов правил, исполняемых во вложенных

транзакциях. Эти классы правил включают :

• *непосредственные (Immediate),* в которых запускаемая триггером транзакция Т1 приостанавливается, а правило R активизируется и исполняется как дочерняя транзакция Т;

• *отложенные (Defen'ed), в* которых непосредственно перед фиксацией транзакции Т создается правило R и оно исполняется как дочерняя транзакция Т;

• *разъединенные (Decoupled),* в которых стартует новая транзакция верхнего уровня;

• *причинные (Causality),* когда запускаемая триггером транзакция верхнего уровня Т2 становится в очередь за транзакцией Т1; если аварийно завершается Т1 , то аварийно завершается и Т2.

Следует здесь отметить, однако, что языки спецификации транзакций могут быть расширены для сред активных баз данных более богатой моделью ограничений, процедурным языком, также механизмами триггеров в дополнение к спецификациям многозвенных и вложенных моделей самих транзакций.

Искусственный интеллект и технология баз данных

А как именно могли бы более тесно интегрироваться эти две дисциплины - искусственный интеллект и технология баз данных? В течение многих лет мы слышали о базах знаний (Knowledge Base) и системах управления базами знании (Knowledge Base Managment System, KBMS), которые включают в среду СУБД не только данные, но и интеллектуальные возможности.

В 1988 г. состоялись два отдельных симпозиума по интеграции ИИ и баз данных. Один из них был организован исследователями в области ИИ , а другой - в области баз данных.

Указанная проблема в значительной мере остается таковой, даже несмотря на высокую степень интереса к интеграции ИИ и баз данных, восходящего к так называемому тесту Тьюринга, предложенному Аланом Тьюрингом в 1951 г. В нем предсказывалась возможность существования машины, имитирующей человеческий интеллект. При создании такой машины потребовались бы какого-либо рода возможности управления данными, которые бы поддерживали "интеллект" машины.

К числу тех областей ИИ, в которых представляла бы ценность более тесная интеграция с технологией баз данных, относятся :

• *совместное использование знаний,* или общая база знаний, разделяемая множеством приложений и пользователей;

• *независимость представления знаний*, или ограничение того негативного эффекта. который связан с изменениями в системах;

• *распределение знаний*, при котором представление знаний фрагментируется и дублируется в значительной мере таким же образом, как это делается с данными в среде распределенной базы данных.

Будут или не будут в конце концов развиваться элементы "истинной" интеграции ИИ с базами данных в этих областях, технология активных баз данных, которая обсуждалась в этой главе, будет проходить долгий путь к достижению этих целей. Так, например, возможности активных баз данных (триггеры, утверждения и другие), надстроенные над средой распределенных баз данных, привели бы к модели распределения знаний. Подобным же образом семантически богатая концептуальная модель данных (с развитыми возможностями выражения правил), на основе которой могут разрабатываться приложения, могла бы, бесспорно, рассматриваться как реализация независимости представления знаний.

Короче говоря, нет какой-либо определенности в том, будут ли когда-либо широко развернуты успешные с коммерческой точки зрения системы управления базами знаний или базы данных с возможностями ИИ или они будут оставаться уделом экспериментальных исследовательских разработок. Привнесение спецификации правил и их обработки в сами базы данных, однако, без сомнения приведет в конце концов к улучшению управления информацией.

Выводы и рекомендации

Технология активных баз данных покрывает широкий спектр возможностей - от базисных средств использования триггеров, свойственных сегодняшним коммерческим продуктам, до развитых моделей спецификации и исполнения правил под управлением вложенных транзакций, которые, вероятно, станут обыденным явлением в ближайшем будущем. Далее, можно считать, что активные базы данных открывают двери на пути к иску еще неуловимому будущему интеллектуальных баз данных с высоким уровнем технологии искусственного интеллекта. Независимо от того, в каком направлении будет развиваться модель интеграции ИИ и баз данных, активные базы данных будут основным инструментальным средством разработки приложений благодаря управлению бизнес- правилами средствами СУБД.

Что может дать эта технология

Поскольку технология активных баз данных стала значительно более доступной, пользователи могут получить благодаря этому ряд преимуществ. Эти преимущества

относятся к трем различным областям: спецификация и поддержка бизнес-правил, разработка приложений и производительность функционирования.

Бизнес -правила

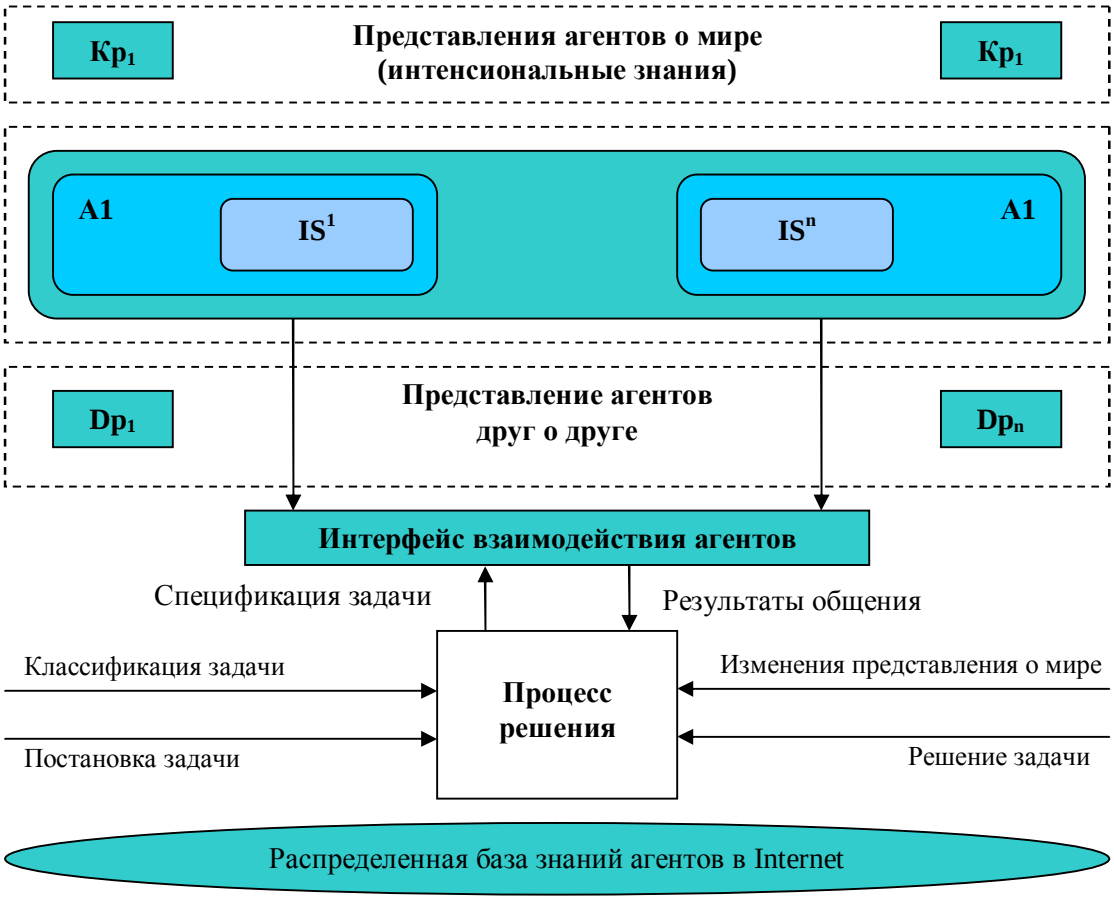
Одними из наиболее трудных областей применения вычислительных средств в настоящее время являются представление бизнес-правил и организация управления ими, иначе говоря, спецификация способов передачи информации от одной сущности информационной системы (например, от пользователя или от какого-либо компонента системы) к другой, а также условий, при которых эти потоки информации имеют место. Активные базы данных обеспечивают общую платформу для представления, поддержки и эффективного исполнения бизнес-правил в единой среде информационной системы.

Разработка приложений

Спецификация компонентов бизнес-правил (в частности, триггеров и последовательностей действий, вызываемых запуском этих триггеров) является в активных базах данных декларативной, а не процедурной, как в традиционных СУБД. Иначе говоря, такие операторы SQL, как CREATE TRIGGER (создать триггер) и CREATE ASSERTION (создать утверждение), специфицируют некоторые компоненты бизнес-правил, и хранимые процедуры могут присоединяться к объектам, которыми они оперируют.

Декларативная спецификация бизнес-правил - значительно более продуктивный процесс, чем процедурная, требуемая в пассивных системах. Такое соотношение этих двух подходов объясняется главным образом тем, что управляющая логика (мониторы, механизмы восстановления) уже представлена в среде СУБД, и нет необходимости возлагать задачу ее разработки на пользователей в их приложениях.

7.2 Среда взаимодействия интеллектуальных агентов



Каждый из агентов обладает локальными знаниями Кр и представлением о знаниях других агентов Dp. В процессе решения задач модифицируются личные базы знания и представлений агентов. Распределенная база знаний агентов в Internet представляет собой суперпозицию личных знаний и представлений агентов.

Intelligent Agents (IA – интеллектуальные агенты). Вообще понятие IA становится все более популярным и вместе с тем более размытым, теряя при этом первоначальный смысл и искажая основную идею. Напомним вкратце, в чем эта идея состоит.

Если необходимо решить какую-либо сложную, нетривиальную задачу, связанную с использованием совершенно экзотических математических методов, о которых вы имеете слабое представление, или выяснить некий малоизвестный исторический факт, например, уточнить родословную любимой собаки Цезаря, найти и загрузить из Internet последнюю версию драйвера видеокарты от малоизвестного производителя, то дальнейшие действия должны происходить по следующему сценарию:

1. Вы активизируете программу-агента на вашем компьютере и в достаточно свободной форме описываете ему интересующую вас задачу.
2. Агент связывается с другими сетевыми агентами, пытаясь выяснить, известно ли им что-либо о решении данной задачи.
3. Ваш агент осуществляется фильтрацию найденной информации с целью идентификации необходимых решений и отсеивания ненужных данных.
4. Каждый (или по крайней мере некоторые) из агентов обращаются к соседним агентам, чтобы узнать возможные адреса информационных хранилищ и/или профессиональных «решателей» данной задачи.
5. Если за предварительно оговоренный период времени агент, инициировавший запрос, не получает ответа, он сообщает вам о том, что решение вашей задачи современной науке неизвестно.

Следует отметить, что приведенный сценарий значительно упрощает содержание реальных процедур, выполняемых агентами: эвристического поиска, интеллектуальных взаимодействий, накопления и обобщения информации, распознавания и классификации. Однако сходство с принципами организации службы имен доменов (DNS-сервиса) действительно имеет место, что придает системе агентов гибкость и живучесть, свойственные сервисам Internet.

В последние годы неоднократно появлялись сообщения о создании коммерческих продуктов на базе интеллектуальных агентов. На самом же деле подобные сообщения не слишком соответствуют действительности. Пока не будут решены некоторые основные проблемы, ожидать появления по-настоящему интеллектуальных агентов не приходится. Среди этих проблемы можно выделить три:

- разработка стандартного языка общения агентов;
- разработка методов эффективной обработки знаний, классификации распознавания;
- разработка «живого» пользовательского интерфейса.

7.3 Организация взаимодействия агентов в многокомпонентных САПР

Имеются другие подходы и принципы построения распределенных гетерогенных САПР на основе технологии многоагентных систем [1]и методологии агентно-ориентированного проектирования по разработке среды проектирования структурированных объектов (Design of Structured Objects – DESO), основанной на концепции интеллектуальных агентов (ИА) [2, 3].

Проблема организации взаимодействия компонент очевидна: одна часть программного обеспечения САПР должна получать доступ к сервисным функциям, предоставляемым другой частью. Другое дело, что в каждом конкретном случае форма и механизм доступа могут быть разными. В одном случае, это может быть обмен знаниями, запрос данных или достижение согласованного решения посредством переговоров, в другом – вызовы локальных функций, передача сообщений между процессами, сетевые коммуникации.

Поэтому и усилия по решению проблемы взаимодействия компонент в распределенных гетерогенных системах предпринимаются в нескольких направлениях. Так, технология распределенного объектного программирования концентрирует усилия на описании стандартного механизма, с помощью которого одна часть программного обеспечения предоставляет свои сервисные функции другой, предоставляя программисту прозрачность механизмов доступа. Речь идет о таких стандартах и подходах к организации коммуникации как CORBA, DCOM, Inter-Language Unification (ILU) [4, 5].

Технология интеллектуальных агентов (ИА) уделяет основное внимание разработке соглашений по спецификации семантики взаимодействия на основе сервисных функций, предоставляемых агентами. Примером такого подхода являются спецификации языка взаимодействия агентов FIPA и язык запросов и обработки знаний (Knowledge Query and Manipulation Language – KQML) [6, 7]. А если добавить сюда усилия, направленные на разработку инженерных онтологий или формальных словарей для представления знаний о технических системах и процессе проектирования [8, 9], то получим тот базис, на котором можно строить взаимодействие в многокомпонентных САПР.

Базовые механизмы взаимодействия в многокомпонентных САПР

Многокомпонентная САПР, как и другие компьютерные системы предприятия, должна обеспечивать кооперацию между компонентами, расширение функциональных возможностей и интеграцию в общую информационную среду. Она является структурированной системой и представляет собой совокупность процессов, как правило распределенных по различным компьютерным платформам, объединенным в сеть. При этом новые процессы должны сопрягаться с существующими носителями и обработчиками информации. До тех пор, пока взаимодействие сложных программных систем (например, взаимосвязь знаний в операциях assert, retract или распределение сервисных функций) остается предметом взаимных соглашений между программистами, кооперация будет ограничиваться простым набором программ, разработанных для взаимодействия друг с другом. Решение данной проблемы лежит в стандартизации механизмов взаимодействия программных компонент.

Основы технологии распределенных объектов

Перспективным, с точки зрения сокращения влияния сложности САПР на ее разработку, сопровождение и использование, видится применение стандартов и соглашений распределенной вычислительной среды (Distributed Computing Environment

– DCE). DCE основана на компонентной архитектуре, которая рассматривает каждый элемент системы (объект) в качестве повторно используемой программной компоненты. Инкапсуляция обеспечивается использованием стандартного интерфейса, который скрывает от пользователя язык или среду разработки компоненты. Тем самым объекты могут легко воспроизводиться и перемещаться по сети, создавая гибкие конфигурации.

Один из подходов к стандартизации взаимодействия компонент предлагает модель многокомпонентных объектов (Component Object Model – COM), которая определяет соглашения и обеспечивает сервисы описания, использования и коммуникации объектов. Этот стандарт позволяет двум приложениям взаимодействовать посредством объектного интерфейса, специфицированного на языке описания, который не зависит от языка реализации объекта. Распределенная COM (Distributed COM – DCOM) – это дальнейшее развитие модели, в которой термин «распределенная» подразумевает способность запуска клиентов и серверов на распределенных платформах Интернет или Интранет. DCOM работает так же как и COM: клиент запрашивает регистр, в котором находится информация о сервере, и вместо того, чтобы использовать локальный указатель, использует непосредственно IP адрес (например, 123.234.199.27) для доступа к серверу. Спецификация DCOM использует протокол объектного удаленного вызова процедур Object RPC. Он представляет собой адаптацию стандарта DCE, имеет ту же структуру пакета, но расширяет систему соглашений о взаимодействии клиент/сервер. Протоколы CN(CO) и DG(CL) используются над транспортным уровнем протоколов TCP или UDP соответственно .

Очевидным недостатком рассмотренной модели с точки зрения построения многокомпонентных САПР является отсутствие описания содержательного аспекта взаимодействия. Это обусловлено трудностями работы приложений в динамических распределенных средах и, как следствие, основная задача, которую преследуют стандарты в этой области заключается в том, чтобы обеспечить приложениям обмен на уровне структур данных и вызова удаленных методов. Однако компоненты САПР (агенты), в том смысле, как обсуждалось в статье [1], не сводятся только к структурам данных и операциям над ними. Поэтому указанные стандарты и протоколы могут рассматриваться лишь в качестве основы, на которой строится язык взаимодействия агентов, определяющий, в первую очередь, семантику взаимодействия.

KQML – язык запросов и обработки знаний

Другая парадигма построения распределенных систем непосредственно строится на концепции агента и использовании языка взаимодействия между агентами для координации их деятельности. Примером такого подхода является спецификация языка запросов и обработки знаний KQML. Он основан на теории речевого взаимодействия (Speech-Act Theory), т.е. сообщение представляет собой исполняемую команду (performative), предусматривающую конкретное действие получателя [10]. Например, простое сообщение «сообщить (tell)» предполагает, что получатель должен верить представленным в теле сообщения фактам, тогда как, сообщение «спросить (ask)» предполагает ответ на присланный запрос.

KQML представляет собой структурированный язык взаимодействия агентов. Его можно рассматривать как двухуровневую структуру: на первом, содержательном уровне описывается содержание сообщения на языке представления агента, а на уровне сообщений или коммуникации описываются параметры сообщения. KQML может являться носителем любого языка представления, включая обыкновенные ASCII строки или бинарные коды. Это обеспечивается тем, что все реализации KQML игнорируют содержательную часть информации, исключая лишь признак ее завершения.

Синтаксис сообщения в семантике KQML включает команду, за которой следует набор слотов в формате: ключевое слово – значение. Параметры сообщения определяют характеристики низкого уровня коммуникации, такие как идентификаторы отправителя и получателя, идентификатор самого сообщения и задают характер взаимодействия с KQML агентом. Основной функцией коммуникационного уровня является идентификация протокола передачи сообщения и поддержка процесса выполнения задания, которое связывается отправителем с содержательной частью сообщения.

KQML используется в качестве языка взаимодействия в различных многоагентных системах и средах для их программирования, таких как Agent-K, LALO, Java(tm) Agent Template (JATLite) [11-13].

Структура взаимодействия в первых двух системах близка. Обе используют концепцию агентно-ориентированного программирования, основанную на ментальных категориях [14], и механизм правил обязательств агентов для описания стратегии их поведения при получении сообщений, записанных на языке KQML. Последние доставляются адресату непосредственно транспортным протоколом TCP/IP .

JATLite использует аналогичную схему организации взаимодействия, но без определения какой-либо теории агентов, т.е. в данном случае уровень описания стратегий взаимодействия отсутствует.

Учитывая, что запросы и интересы выражаются в KQML формальным декларативным языком, значение терминов не зависит от конкретных программ и, следовательно, может разделяться программами с различной реализацией контекста и представлением знаний. Таким образом, взаимодействие описывается на высоком уровне, тогда как детали посылки сообщений, связанные с различием операционных сред и алгоритмов принятия решений, оказываются скрытыми. Вместе с тем, в системах, состоящих из различных приложений, таких как САПР, использование языка KQML для спецификации взаимодействия только на уровне знаний не в силах полностью решить проблемы интеграции, хотя бы по той простой причине, что многие компоненты не являются интеллектуальными и не поддерживают этот уровень взаимодействия.

Модель взаимодействия агентов в многокомпонентной САПР

На текущем этапе разработки среды DESO было признано целесообразным использовать интеграцию концепций агентов и распределенных объектов на основе многоуровневой структуры взаимодействия агентов. Каждый уровень определяет управляющий интерфейс между различными уровнями абстракции, механизмами и протоколами.

Разделяемая онтология обеспечивает общий словарь для решения прикладных задач проектирования, определяет семантику сообщений и отвечает за интерпретацию контекста сообщения. Его расссмотрение выходит за рамки данной статьи.

Протокол взаимодействия агентов определяет стратегию высокого уровня и структуру переговоров между агентами. В динамическом и кооперативном контексте многокомпонентных САПР ее можно характеризовать как комбинацию взаимодействий типа «запрос/ответ» и «обмен знаниями (или информирование)». На этом уровне семантика сообщений связывается с действиями, предпринимаемыми агентом:

Правило обязательства определяет действия агента как реакцию на сообщение «ask-one», которое запускает на выполнение предикат WorkingArea и возвращает результат в формате KQML агенту источнику сообщения.

Язык взаимодействия агентов определяет семантику взаимодействия независимо от приложения, определяет тип и содержание запроса. Этот уровень преобразует соответствующую часть правил, связанную с генерацией агентом сообщений в выражения языка KQML и обратно. Приведем в качестве примера запрос о величине рабочей зоны конкретного станка из типового диалога агентов в DESO:

В приведенном сообщении ask-one – это KQML команда, содержание сообщения – working_ area(Machine, X), а предполагаемая онтология имеет идентификатор MachineDBManager.

Протокол взаимодействия объектов определяет стандартные механизмы, которые используются для логического взаимодействия между компонентами. В DESO механизмы OLEnterprise выполняют посылку выражений KQML в качестве семантических событий (вызовы сервера ActiveX). ActiveX обеспечивает функции совместимости, используя Object Data Base Connectivity (ODBC), Object Request Broker (ORB), Internet Inter-ORB Protocol (IIOP), и Object Remote Procedure Calls (ORPC). При этом Interface Definition Language (IDL) используется для описания отображения команд и параметров KQML и DCOM. Аналогичный подход, основанный на интеграции KQML и CORBA в архитектуре COBALT, представлен в работе [15].

Транспортный протокол представляет модель и механизм физического взаимодействия. Механизмы OLEnterprise используют протокольный стек Transmission Control Protocol/Internet Protocol (TCP/IP) для реализации сообщений низкого уровня.

Организация взаимодействия между агентами DESO на основе механизмов OLEnterprise

Рассмотрим некоторые особенности реализации рассмотренной модели. В качестве платформы для реализации прототипа DESO были использованы средства OLEnterprise, обеспечивающие динамический OLE доступ к распределенным объектам [16]. OLEnterprise основан на спецификации DCOM, тем самым обеспечивая полную интеграцию со средствами разработки и приложениями Windows. OLE компонент может выполняться в качестве in-process или out-of-process по отношению к клиенту. В первом случае он реализуется в виде файла библиотеки динамического линкирования (Dynamic Link Library – DLL) и выполняется в процессе клиента. Во втором случае компонент выполняется в виде исполняемого файла и размещается в своем адресном пространстве.

OLEnterprise включает 3 компоненты: Object Explorer (броузер локальных и удаленных объектов), Object Agent (локальный сервер OLE Automation) и Object Factory (удаленный контроллер OLE Automation). Эти компоненты интегрированы в среде DESO и позволяют приложениям клиентам использовать объекты OLE Automation и объекты RPC сервера независимо от их расположения в сети. Кроме того, гибкий механизм управления сервисами обеспечивает прозрачность размещения и балансировку загрузки серверов, авторизацию пользователей и защиту данных. Общая схема обработки сообщений агентов DESO на базе OLEnterprise показана на рис. 3.

Поясним назначение и функционирование каждого элемента приведенной схемы. Object Agent – это in-process сервер OLE Automation, исполняемый как DLL. Object Agent располагается на машине клиента и обеспечивает динамический прозрачный доступ к любым объектам OLE или RPC. Он определяет механизм

посылки сообщений локальных агентов-клиентов DESO, реализованных как Automation Controller [3]. Для агента DESO он выглядит так же как и любой другой

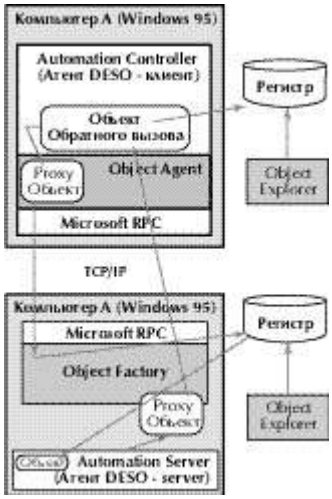


Рис. 3. Архитектура взаимодействия агентов DESO с использованием механизмов OLEnterprise

сервер OLE Automation. Но он способен перенаправлять запрос агента удаленному серверу. Если удаленный объект является объектом OLE, то запрос адресуется Object Factory, которая воспроизводит запрос ActiveX на удаленной машине. Если удаленный объект является объектом OLEnterprise, то Object Agent автоматически преобразует запрос в RPC запрос и направляет его соответствующему серверу.

Object Factory – удаленный OLE controller, обеспечивает доставку распределенных сервисов OLE Automation (менеджер удаленных OLE запросов). Object Factory располагается вместе с агентами-серверами DESO, реализованными как Automation Server, и отвечает за их активизацию и формирование запросов от лица удаленных агентов-клиентов. Object Factory является одновременно RPC сервером и клиентом OLE Automation. Он выполняет функции виртуального (proxy) клиента OLE Automation, представляя удаленного агента-клиента.

Object Explorer содержит три основные функциональные компоненты: броузер для просмотра локальных и удаленных регистров, механизм экспортирования для выбора серверов OLE Automation для удаленного доступа и механизм импортирования для локальной регистрации удаленных серверов OLE Automation. Он используется для инициализации агентов DESO. На сервере генерирование и регистрация сервера OLE Automation в регистре производится с использованием стандартных утилит OLE.

Архитектура многокомпонентной САПР

В заключение рассмотрим общую концепцию архитектуры многокомпонентной САПР, вписывающую концепцию агентов в общую архитектуру клиент-сервер, построенную на основе концепции распределенных объектов (рис. 4). Серым цветом выделены компоненты среды DESO.

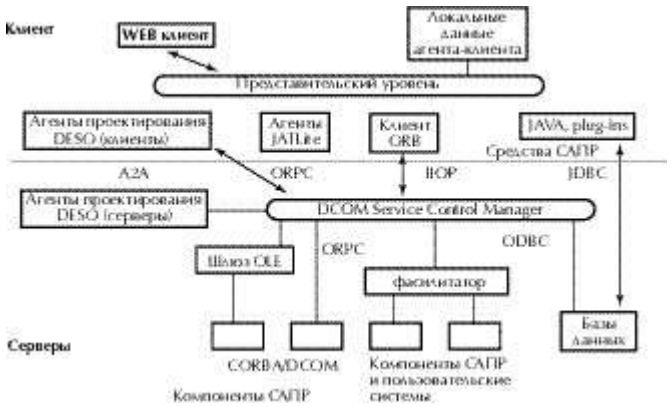


Рис. 4. Архитектура многокомпонентной САПР

Взаимодействие пользователя с распределенными компонентами САПР существенно упрощается при использовании Интернет/Интранет решений для разработки пользовательского интерфейса. В этом случае компоненты САПР (клиенты) могут быть интегрированы посредством так называемой «шины представительского уровня». Ее цель – позволить компонентам клиентам (агенты DESO, агенты-Java, plug-ins и ActiveX, ORBs) разделять модели взаимодействия распределенных объектов. В качестве реализации этого уровня может быть использован, например, Netscape's LiveConnect [17].

Агенты могут использовать структуру коммуникации «каждый-с-каждым» (peer-to-peer или A2A). Так, агенты DESO могут непосредственно использовать протокол A2A, который выполняется на стандартных протоколах, таких как сокеты. С другой стороны, использование протоколов взаимодействия объектов (механизмов OLEnterprise) обеспечивает взаимодействия без точного указания (знания клиентом) места расположения вызываемого метода. В системах этого типа существует особый сервер, склеивающий клиентов и серверы. В DCOM – это Service Control Manager (SCM), в CORBA – это брокер объектов, поддерживающие удаленную активацию объектов (с использованием интерфейса IActivation). В мультиагентной системе сервер имен агентов (Agent Name Server – ANS или Agent Message Router – AMR в JATLite) выполняет те же функции, преобразуя логическое имя агента в его адрес, соответствующий физическому расположению. ANS поддерживает регистр агентов и услуг, за которые он отвечает.

Реализация компонент САПР может быть выполнена в виде любой комбинации Java аплетов (applets), встраиваемых фрагментов (plug-ins), компонент ActiveX, и компонент сервера, поддерживающих IIOP. При этом стандартные протоколы, типа HTTP, RMI и JDBC, дополняются IIOP связи с DCOM сервером. Интерфейсы выставляются ORB путем компиляции их спецификации, написанной на IDL, что обеспечивает возможность подключения внешних приложений. Кроме того, OLEnterprise включает шлюз OLE Gateway, который эффективно расширяет возможности OLE, поддерживая механизмы RPC и CORBA и обеспечивая интеграцию Windows компонент DESO с компонентами Unix и других платформ.

Для поддержки обработки описанной выше структуры сообщений между подсистемами могут быть введены специальные коммуникационные агенты (фасилитаторы). Использование фасилитаторов становится особенно важным в условиях интеграции новых компонент САПР с существующими, которые не имеют необходимой степени гибкости в преобразовании входных/выходных форматов.

Литература

1. Смирнов А.В., Шереметов Л.Б. Многоагентная технология проектирования сложных систем. Автоматизация проектирования, №№ 3 – 1998, 1 – 1999

2. Smirnov, A.V., Sheremetov, L.B., Romanov, G.V. & Turbin, P.A. Multi-Paradigm Approach to Cooperative Decision Making. In Proc. of the II International Conference on Concurrent Engineering: A Global Perspective. McLean, Virginia, August 23-25, Concurrent Technology Corporation, 1995. pp. 215 – 222.

3. Sheremetov, L.B. & Smirnov, A.V. A Model of Distributed Constraint Satisfaction Problem and an Algorithm for Configuration Design. Computacion y Sistemas, 1(2): 91-100, 1997.

4. CORBA 2.0 Specification, Object Management Group, ptc/96-03-04, 1996.

5. Distributed Component Object Model protocol DCOM/1.0. Network Working Group, Internet-Draft, 1996.

6. FIPA7412, Draft 1.0, document of the Foundation for Intelligent Physical Agents (FIPA), 1997.

7. Finin, T., Labrou, Y., & Mayfield, J. KQML as an agent communication language, In Jeff Bradshaw (Ed.), Software Agents, MIT Press, 1995.

8. Bradshaw, J. M., Holm, P. D., Boose, J. H., Skuce, D., & Lethbridge, T. C. Sharable ontologies as a basis for communication and collaboration in conceptual modeling. In Proceedings of the Seventh Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada, 1992, pp. 3.1 – 3.25.

9. Patil, R.S., Fikes, R.E., Patel-Schneider, P.F., etc., The DARPA Knowledge Sharing Effort: Progress Report. In Huhns, M.N. & Singh, M.P. (Eds.), Readings in Agents, Morgan Kaufmann Publishers, Inc. 1998, pp. 243 – 254.

10. Cohen P.R. & Levesque H.J. Communicative actions for artificial agents. In Proceedings of the First International Conference on Multi-agent Systems (ICMAS'95), San Francisco, CA, 1995.

11. Davies, W.H.E. & Edwards, P. Agent-K: An Integration of AOP and KQML. AUCS/TR9406, Univ. of Aberdeen, Scotland, UK, AB92UE, 1994.

12. Gauvin, D. Un environnement de programmation oriente agent. Dans Troisiemes journees francophones sur l'intelligence artificielle distribuee et les systemes multiagents. St-Baldoph, Savoie, France, 15-17 mars 1995.

13. Frost, H.R. & Cutkosky, M.R. Design for Manufacturability via Agent Interaction. In 1996 ASME Design for Manufacturing Conf., Irvine, CA, Aug., 1996, pp. 18 – 22.

14. Shoham, Y. Agent-oriented Programming. Artificial Intelligence 60(1):51-93. 1993.

15. Benech, D., Desprats, T. & Raynaud, Y. A KQML-CORBA Based Architecture for the Communication of Intelligent Agents in Cooperative Service and Network Management. In Proc. of the First IFIP/IEEE International Conference on Management of Multimedia Networks and Services '97, July 8-10, Montreal, Canada. 1997.

16. Koulinitch, A.S. & Sheremetov, L.B. Agents coordination and communication in distributed configuration design. In Primer Encuentro Nacional de Computacion, Taller de Sistemas Distribuidos y Paralelos, Queretaro, Mexico, Sept., 12-15, 1997, pp. 92 – 99.

17. Bradshaw, J. M. KAoS: An open agent architecture supporting reuse, interoperability, and extensibility. B. R. Gaines & M. Musen (Ed.), Proceedings of the Tenth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada, 2 (pp. 48:1 – 48:20).

8 Технологии интеграции современных сервисов в корпоративной сети - SOA.

8.4 *Технологии SOA. Общие характеристики и проблеммы. GRID- среда.*

Каждый предшествующий век начинался с очередной технической революции: XIX век — с появления паровых машин, XX-й — с электричества и двигателей внутреннего сгорания. Лет через пятьдесят потомки скажут с какой революции начался XXI век. Пока же можно предположить, что среди претендентов на первенство есть еще не нашедшие своего точного названия системные изменения в компьютинге. Возможно, в этом ряду могут оказаться и Архитектуры, ориентированные на сервис.

Увы, современники спешат с оценками. Часто получается так, что, к сожалению, компьютерные аналитики и пресса берут на себя примерно ту же самую функцию, что и дома высокой моды. Их объединяет общее стремление пропагандировать то, что еще пока никому не нужно — и, быть может, не станет нужным никогда. К тому же, любая мода — и компьютерная мода не исключение — обязательно должна быть не просто новой, а радикально новой. Старая мода нужна не больше, чем прогноз погоды на вчера.

Вокруг новинок создается излишняя шумиха, мешающая отделить мнимые новации от подлинных ценностей. К числу последних вполне можно отнести хиты нынешнего сезона: «Архитектура, ориентированная на сервисы» (Service Oriented Architecture — SOA) и «Сервисная шина предприятия» (Enterprise Service Bus — ESB). Это явления более чем серьезные, хотя есть и скептические мнения. Так, заметный резонанс получила статья инженера из американского отделения компании Software AG Майкла Чампиона [1], который язвительно называет хвалу SOA и ESB «пропагандой очередного сдвига парадигмы» и «десятой революцией из двух предсказанных». Под влиянием SOA может измениться, в том числе, и мир разработки приложений [2]. Чем же обусловлены грядущие изменения?

8.4.1 Мнения аналитиков

Gartner Group утверждает, что следующие годы должны стать переломным в истории индустрии программного обеспечения. Заголовки предсказаний Gartner звучат так: «SOA изменяет программное обеспечение», «SOA вступает в эпоху Web-сервисов», «SOA взбудоражит рынок серверов приложений», «Сервисная шина предприятия вышла на свет».

IDC опубликовала отчет «Сервисная шина предприятия революционизирует информационные технологии», где в качестве базиса для этого технологического решения фигурирует, SOA. Менее известная аналитическая компания ZapThink также утверждает, что SOA станет в ближайшие годы основой для ориентированной на процессы интеграции приложений, а объем этого сегмента рынка всего за несколько лет вырастет до 8-10 млрд. долл. Схожие заявления о перспективности SOA можно найти и у Giga Group, Meta Group и др. Однако надо признать, что все более или менее именитые аналитики по части ИТ уступили пальму первенства небольшой компании из Сан-Франциско Stencil Group.

Специалисты Stencil одними из первых смогли не просто оценить перспективность SOA, но и показать глубинную природу изменений, происходящих в архитектуре корпоративных систем [3]. Согласно их трактовке, появление SOA есть следствие закономерного эволюционного процесса, который наблюдается в течение последних 40-50 лет (таблица 8.1).

Таблица 8.1. Три поколения корпоративных систем по классификации Stencil Group

АРХИТЕКТУРА	МЭЙНФРЕЙМ	КЛИЕНТ-СЕРВЕР	ОРИЕНТИРОВАННАЯ НА СЕРВИСЫ
Платформа	монолитная и централизованная	изолированные локальные сети	взаимоувязанная Internet-технологиями
Форматы данных	закрытые и недоступные	двоичные и специализированные	семантические и распределенные
Ключевая технология	операционная система	база данных	интерфейс
Пользователь	оператор	прикладные специалисты	сотрудники, поставщики и заказчики
Значение для бизнеса	оцифровка операций, связанных с данными	передача данных в руки менеджеров	обеспечение эффективности и возможности взаимодействия

Архитектура SOA, как считают в Stencil, представляет собой новый и вполне закономерный этап эволюции корпоративных систем. Предпринятая попытка классификации интересна и полезна, но стоит учесть, что она несколько поверхностна. В ней все сводится лишь к технической стороне, к тому, что обычно называют информационной системой; между тем, она, в свою очередь, является подсистемой всей корпоративной системы в целом. Информационная система, как совокупность аппаратных и программных средств, эволюционирует не автономно; она развивается в сложной взаимосвязи с развитием всего корпоративного организма. Поэтому первые четыре строки, непосредственно относящиеся к технологиям, можно принять без всякой критики; что же касается «пользователей» и «значения для бизнеса», то и с этой частью можно согласиться, но с определенной натяжкой. Дело в том, что при переходе от мэйнфремов к клиент-серверным конфигурациям и далее к SOA изменяется функциональность, пересматриваются представления об информации и данных, а это явления более глубокие, чем изменения в аппаратной или программной архитектуре.

Соответственно архитектуре меняются и средства разработки. Еще в 1998 году по используемым средствам разработки совокупность новых проектов делилась поровну [12]: 50% приходилось на долю технологии Microsoft DNA и 50% — на долю всех остальных. Язык Java обладал долей, составляющей всего лишь около 1%. В 2005 году средства разработки, связанные с Java, отнимут чуть менее 40% рынка и станут самым большим сегментом. Оставшуюся часть разделят примерно поровну средства Microsoft .Net и традиционные программные технологии, причем на Microsoft DNA придется всего несколько процентов. Соответственной этому будет и динамика занятости.

8.4.2 Попытки определения SOA

Неопределенность, наблюдаемая сегодня в представлениях о SOA, хорошо знакома — явный «эффект дежавю». Когда в конце 90-х заговорили о Web-сервисах, на читателей обрушился поток противоречивых определений: на сайте компании Systinet справедливо утверждается, что, если попросить помощи у пяти специалистов, то можно получить, как минимум, шесть определений [4].

Столь же противоречиво определяют сегодня и SOA.

I. Stencil Group. Архитектура, ориентированная на сервисы имеет четыре основные характеристики.

1. Является распределенной. Функциональные элементы приложений могут быть распределены по множеству вычислительных систем и способны к взаимодействию с использованием локальных или глобальных сетей. В частности, Web-сервисы позволяют использовать существующие протоколы, например, HTTP.

2. Строится с использованием слабосвязанных интерфейсов. Обычно приложения проектируются в расчете на жесткую связь всех элементов. Как следствие, система должна иметь целостный проект, его изменения в процессе эксплуатации затруднительны. Работу компонентов в слабосвязанных системах проще координировать, системы проще реконфигурировать.

3. Базируется на общепринятых отраслевых стандартах.

4. Проектируется с ориентацией на процессы (process-centric) с использованием сервисов, каждый из которых ориентирован на решение отдельных задач (task-centric).

II. Gartner Group. В Gartner видят в качестве основной черты SOA деление на два или более уровня [5]. Обращенный к пользователю презентационный уровень отделен от внутреннего, где реализуется бизнес-логика. Последний создается из крупных блоков (coarse-grained chunk), которые обеспечивают сервис клиентским программам презентационного уровня. В логический уровень могут включаться компоненты управления бизнес-процессами (business process management — BPM). Практически, как считают аналитики этой компании, SOA можно реализовать, в том числе, многозвенной клиент-серверной архитектурой, но не двухуровневой архитектурой с толстым клиентом, который совмещает как логику, так и презентацию.

III. International Data Corp. Аналитики IDC [6] определяют SOA как архитектуру, предназначенную для свободного размещения функциональных модулей, каждый из которых способен выполнять определенные действия. Эти модули

представляют собой описывающие сами себя программные компоненты, достижимые через сеть. Модули публикуют свои интерфейсы таким образом, что их использование не требует знания использованных в них решений и технологий; их удобно воспринимать как черный ящик. SOA можно реализовать и без использования Web-сервисов, однако сервисы рассматриваются в качестве основного средства для создания подобной архитектуры.

IV. Barry & Associates. Данная консалтинговая компания определяет SOA как набор взаимодействующих между собой модулей [7]. Модули могут просто обмениваться между собой данными или каким-то образом координировать свои действия. Архитектура, ориентированная на сервисы, не представляет собой ничего нового, считают здесь. Ее первыми примерами стали брокеры объектных запросов, основанные на спецификации CORBA.

V. В компании ThoughtWorks утверждают: SOA представляет собой набор сервисов, которые совместно образуют систему, призванную заменить монолитные корпоративные приложения типа ERP [8]. Примерно такого же мнения придерживается обозреватель журнала EAI Майкл Паллос, считающий, что SOA представляет собой набор модулей для обеспечения бизнес-процессов [9]. Дон Редер из компании Iona Technologies соглашается с таким мнением, отмечая, что SOA предлагает методологию и набор средств интеграции приложений [10].

Обобщая эти и многие другие определения SOA, можно согласиться с Майклом Чампионом [1]: «И все же, мне кажется, за всей этой шумихой скрывается мощная идея, которая еще не вполне точно обозначена; она и называется сервисами». Эта идея размыта, поскольку пока не поддержана прозрачной и общепринятой моделью или набором моделей, подобных семиуровневой модели открытых систем.

Не находя общего языка, авторы в своих определениях под SOA понимают близкие, но все же разные вещи. Одни рассматривают SOA как своего рода механизм, создающий коммуникационную среду для модулей, реализующих прикладную бизнес-логику. Примерно в этом ключе высказывается Адам Босуорт, вице-президент компании BEA Systems [11]. В его представлении

посредством сервисов вычислительная система преобразуется в своего рода коммуникационную систему для снабжения информацией; она превращается в сервис сродни водопроводу. Стоит обратить внимание на близость взглядов Босуорта с прогнозами Stencil Group относительно будущего баз данных. Различие между ними заключается в том, что Босуорт предлагает их прямую транспортировку в потоках данных вместо накапливания данных в базах. Подобный коммуникационный взгляд на SOA можно назвать «взглядом снизу». Но есть другой, более системный взгляд, «взгляд сверху» [8-10]. Прикладные модули, получающие данные или информацию посредством сервисов, сами в свою очередь являются сервисами, которые необходимо каким-то образом связать в одну систему.

8.4.3 SOA и Web-сервисы

Архитектура SOA широко заявила о себе, став логическим продолжением технологий Web-сервисов. Однако способы реализации сервисов могут быть различными. В общем случае любая архитектура, ориентированная на сервисы, обязательно содержит три компонента: поставщика сервиса, потребителя сервиса и брокера (рис. 8.1).

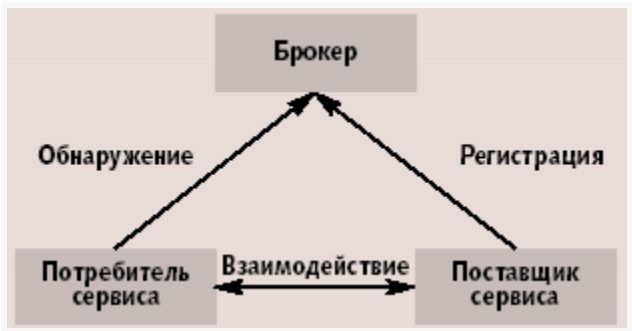


Рис. 8.1. Архитектура, ориентированная на сервисы

Поставщик сервиса регистрирует его у некоторого брокера, потребитель там же его обнаруживает, после чего между ними устанавливается связь в соответствии с контрактом и осуществляется требуемое действие. Для создания SOA необходимы три основных типа соглашений:

- транспортное, определяющее форматы и протоколы;

- описание сервиса, для чего нужен некий читаемый машиной язык, на котором описываются выполняемые сервисом операции; после компиляции описания формируется код, определяющий процесс взаимодействия между клиентом и поставщиком сервиса;
- протокол, который определяет способ обнаружения сервиса.

Идея сервисов в информационных системах как таковая не нова. Хорошо известны следующие подходы к ее реализации:

- Java RMI от компании Sun Microsystems (Java Remote Method Invocation);
- CORBA консорциума Open Management Group (Common Object Request Broker Architecture);
- DCE предложенная ассоциацией Open Group (Distributed Computing Environment);
- Microsoft DCOM (Distributed Component Object Model).

Каждую из архитектур, которая реализуется этими средствами, вполне можно назвать ориентированной на сервисы, но при этом каждая из них определяет свои собственные форматы и протоколы, механизмы вызовов, интерфейсы для прикладных программ. Недостаток универсальности ограничивает их распространение. В сегодняшней трактовке SOA под сервисами понимают Web-сервисы, в основе которых лежат общепринятые Internet-технологии и развитая инфраструктура.

Главная отличительная черта Web-сервисов состоит в использовании XML. Что же привнес этот язык в идею сервисов? Это чрезвычайно интересный вопрос, на который нельзя дать полноценный ответ без глубинного анализа, без рассмотрения различий между данными и информацией, не обращаясь не столько к техническим, сколько к философским вопросам теории информации.

Первыми и наиболее очевидными соглашениями, позволяющими реализовать Web-сервисы, стали «стандарты» SOAP, WSDL и UDDI, в шутку называемые сегодня «святой троицей». Фактически это не стандарты, а три предложения от группы крупных производителей, заинтересованных в развитии SOA, возможно, наиболее рациональны, но далеко не единственны. Известно мнение,

что Web-сервисы — это только то, что реализуется стеком из SOAP, WSDL и UDDI. Существует, скажем, предложенный Роем Филдингом альтернативный подход, называемый REST, который позволяет реализовать функции Web-служб без привлечения этих стандартов. Так или иначе, в общественном мнении Web-сервисы главным образом ассоциируются именно с этими технологиями.

В таблице 8.2 приведена классификация сервисов и поддерживающих соглашений и стандартов[4].

Таблица 8.2. Сравнение архитектур, ориентированных на сервисы

	JAVA RMI	CORBA	DCE	WEB-СЕРВИСЫ
Механизм вызова	Java RMI	CORBA RMI	RPC	JAX-RPC, .Net
Формат данных	Serialized Java	CDR	NDR	XML
Формат обмена	Stream	GIOP	PDU	SOAP
Протокол передачи	JRMP	IIOP	RPC CO	HTML, SMTP
Описание интерфейса	Java Interface	CORBA IDL	DCE IDL	WSDL
Механизм обнаружения	Java Registry	COS naming	CDS	UDDI

8.4.4 Два лица SOA

Наличие двух возможных точек зрения на SOA и на Web-сервисы привело к тому, что над вопросами стандартизации в параллель работают две группы: в рамках комитета W3C и в составе Organization for the Achievement of Structured Information Standards (OASIS). Работа первой группы сосредоточена на основных спецификациях сервисной инфраструктуры, а второй — на расширении функциональности.

В рамках W3C работы по стандартизации Web-сервисов начались в 2000 году, когда была создана рабочая группа XML Protocol Working Group (XMLP), а затем сложилось объединение Web Services Activity, состоящее из трех рабочих групп и одной координирующей. Деятельность Web Services Architecture Working Group (WSAWG) сосредоточена на развитии стандартов для трех компонентов сервисов, включая транспорт, описание и обнаружение, максимально близких к SOAP, WSDL и UDDI. Работа групп WS-Desc и XMLP

непосредственно связана с разработкой XML-спецификаций для описания Web-сервисов. Координирующая группа WSCG связывает деятельность W3C в области сервисов с другими организациями.

OASIS имеет в своем составе более тридцати технических комитетов, деятельность которых связана архитектурой Web-сервисов; в большинстве своем они были созданы после 2000 года. Так, Web Security Technical Committee разрабатывает стандарты, обеспечивающие безопасную передачу данных средствами, построенными на основе SOAP. Со своей стороны, XML-Based Security Service Technical Committee работает над стандартизацией механизма авторизации и аутентификации с использованием XML, в том числе, языка Security Assertion Markup Language (SAML). Есть комитеты, отвечающие за выработку языков управления ресурсами, за обмен биометрической информацией и целый ряд других.

8.4.5 Web-сервисы в интерпретации W3C

До начала прошлого года деятельность в области Web-сервисов в основном сводилась к самоутверждению и конкуренции крупных компаний на поле стандартов, подобных SOAP, WSDL, UDDI и ряду других. Это был совершенно необходимый этап, в войне стандартов идеи Web-сервисов прошли обкатку, и теперь появилась вполне реальная возможность для выработки практических решений, которые могут быть приняты отраслью в целом.

В качестве рабочего органа, который должен взять на себя эту обязанность, в январе 2002 года в рамках W3C была образована рабочая группа Web Services Architecture Working Group [13]. За прошедшее время группой опубликован целый ряд документов, в том числе глоссарий терминов из области Web-сервисов [14]. В августе 2003 года была опубликована рабочая версия документа «Архитектура Web-сервисов» [15], устранив противоречия, которыми страдали отдельные публикации, и предложив единую платформу для понимания того, что такое Web-сервисы и SOA. Оговоримся сразу — все, что предлагает эта рабочая группа, относится к действующей модели Web; кроме того, существуют академические исследования, в которых разрабатываются

Web-сервисы для Семантической Сети, но это сфера пока еще далека от реального внедрения.

WSAWG предлагает следующее определение: «Web-сервис — это реализуемая программными средствами система для поддержки межмашинного взаимодействия через сеть. Интерфейс сервиса описывается на языке, читаемом машиной, например, WSDL. Другие системы взаимодействуют с Web-сервисом способом, указанным в его описании, используя сообщения в стандарте SOAP, передаваемые с использованием HTTP и XML и в сочетании с другими стандартами, относящимися к Web». Физически Web-сервис представляет собой фрагмент программного обеспечения, называемый «агентом». Агент способен передавать и принимать сообщения, он реализует абстрактную функциональность сервиса. Следует проводить различие между агентом и сервисом, один и тот же сервис может быть обеспечен разными агентами. Это вполне тривиальная с инженерной точки зрения идея: скажем, в компьютере должен быть отвод тепла, в таком случае сервис — это охлаждение, а вентилятор — это агент; один вентилятор может быть заменен другим.

Web-сервис предназначен для предоставления некоторой функциональности от лица ее владельца (поставщика), это может быть человек или организация (provider entity), которая использует соответствующий собственный агент (provider agent). Запрашивающая сторона (requester entity) тоже может быть человеком или организацией, желающими использовать сервис. Для этого она тоже использует свой агент (requester agent). Агенты взаимодействуют между собой посредством обмена сообщениями. Для того чтобы этот обмен был успешным, стороны прежде должны достичь соглашения об общем понимании семантики механизме обмена сообщениями.

Механизм обмена сообщениями документирован в описании сервисов (WSD), которое представляет собой читаемую машиной спецификацию интерфейса, включающую: форматы сообщений, типы данных, транспортные протоколы, форматы сериализации, используемые при обмене между агентами заказчика и поставщика услуг. Она также содержит указание на одну или несколько точек в сети (endpoint), откуда поставщик доступен и еще может содержать информацию о предполагаемом шаблоне (pattern) сообщения, используемого для обмена.

Семантика сервиса представляет собой содержание так называемого «контракта» между сторонами; она также содержит некоторые дополнительные сведения о механизмах обмена сообщениями, которые не отражены в WSD. Важно отметить, что, хотя контракт содержит всеобъемлющее представление о сервисе, он не должен быть документально зафиксированным. Контракт может быть явным или скрытым, устным или письменным, в форме, доступной машине или человеку. Различие между описанием сервиса и контрактом заключается в том, что первое определяет механизм взаимодействия с сервисом, а второй — смысл и предназначение этого взаимодействия.

Web-сервисы в основном предназначены для автоматизации процессов управления, которые могли бы быть выполнены вручную, однако в архитектуре Web-сервисов остается место и человеку. Во-первых, его участие требуется при достижении соглашения по семантике и описанию. Человек (или организация) является юридическим владельцем Web-сервисов; именно люди должны согласиться с тем, что данная семантика и данное описание будут управлять взаимодействием между сервисами. Возможно, что со стороны поставщика участие может ограничиться публикацией и предоставлением возможности использовать сервис только в той форме, в какой тот существует (take-it-or-leave-it); иными словами, контракт предполагает неизменяемые условия с его стороны. Есть и другие формы установления отношений, в том числе, через координирующие организации или даже по запросу заказчика. Во-вторых, агенты заказчика и поставщика сервисов создаются людьми, которые отвечают за исполнение соглашений об описании и семантике услуги.

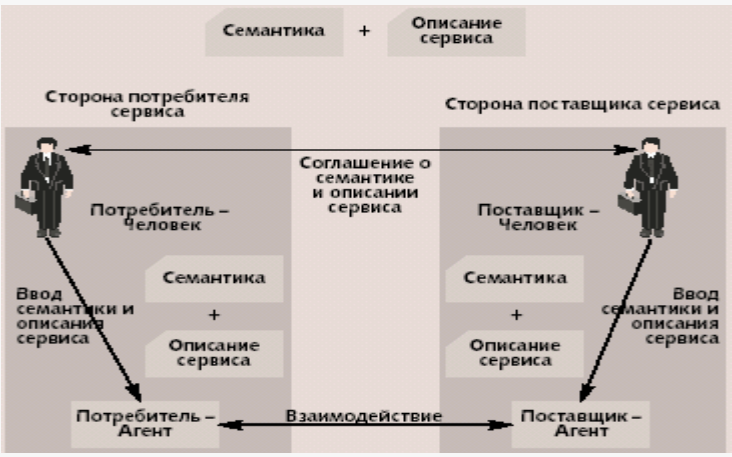


Рис. 8.2. Архитектура, ориентированная на сервисы, с учетом предложений WSAWG

На рис. 8.2 представлена схема обмена между потребителем и поставщиком сервисов с учетом предложений, сделанных WSAWG.

8.4.6 Архитектурные модели SOA

Одной из самых интересных находок, сделанных в [15], можно назвать предложенные архитектурные модели SOA.

- **Модель, ориентированная на сообщения** (Message Oriented Model, MOM), сосредоточена на сообщения, их структуру, способы транспортировки и другие компоненты, ни как не связанные с причинами обмена сообщениями и их семантикой.
- **Модель, ориентированная на сервисы** (Service Oriented Model, SOM), сосредоточена на действиях, выполняемых сервисами.
- **Модель, ориентированная на ресурсы** (Resource Oriented Model, ROM), сосредоточена на системных ресурсах.
- **Модель политик** (Policy Model, PM) определяет политики, связанные с архитектурой, в основном она представляет собой ограничения, накладываемые на поведения агентов и сервисов, в том числе ограничения, связанные с требованиями безопасности, качества обслуживания.
- **Модель управления** (Management Model, MM).

Анализ SOA с использованием каждой из моделей, характеристики моделей и их взаимосвязь являются отдельной темой. В контексте данной статьи достаточно согласиться с тем, что SOA как система сложнее других информационных систем, и поэтому она требует для своего описания не одной, а нескольких моделей. Наличие разных моделей позволит обеспечить согласованную работу специалистов разных профилей, согласование различных стандартов, образование структуры стандартов. Привычное представление о такой простой структуре стандартов, как стек, в случае SOA явно не подходит. Для установления связей между компонентами и сборки единой системы из этих слабосвязанных и асинхронных компонентов требуются совершенно иные

приемы средства, которые получили довольно непривычно звучащие названия: «оркестровка» (orchestration) и «хореография» (choreography).

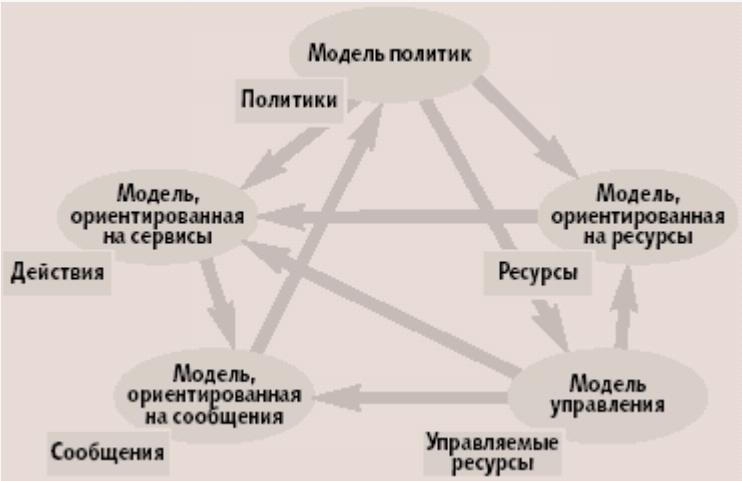


Рис. 8.3. Архитектурные модели SOA

На Рисунке 8.3 приведена схема взаимодействия различных моделей между собой.

8.4.7 Оркестровка и хореография Web-сервисов

С появлением Web-сервисов возникли термины «композиция Web-сервисов» (Web-services composition) и «поток Web-сервисов» (Web-services flow), используемые для описания потока работ, выполняемых с их помощью. Недавно им пришли на смену два более выразительных термина — оркестровка и хореография Web-сервисов. Под оркестровкой понимают то, как сервисы взаимодействуют друг с другом на уровне сообщений, включая бизнес-логику и кооперацию при выполнении сложных процессов в пределах одного предприятия. Оркестровка основывается на открытых стандартах для создания бизнес-процессов, включая BPEL4WS, WSCI и BPML. Хореография в данном контексте охватывает более широкий круг участников взаимодействия, в том числе поставщиков, потребителей и партнеров предприятия. Хореография ассоциируется с публичным обменом сообщениями между множеством Web-сервисов, а не с одним бизнес-процессом, осуществляемым на одном предприятии [15]. Хореография — более публичный процесс.

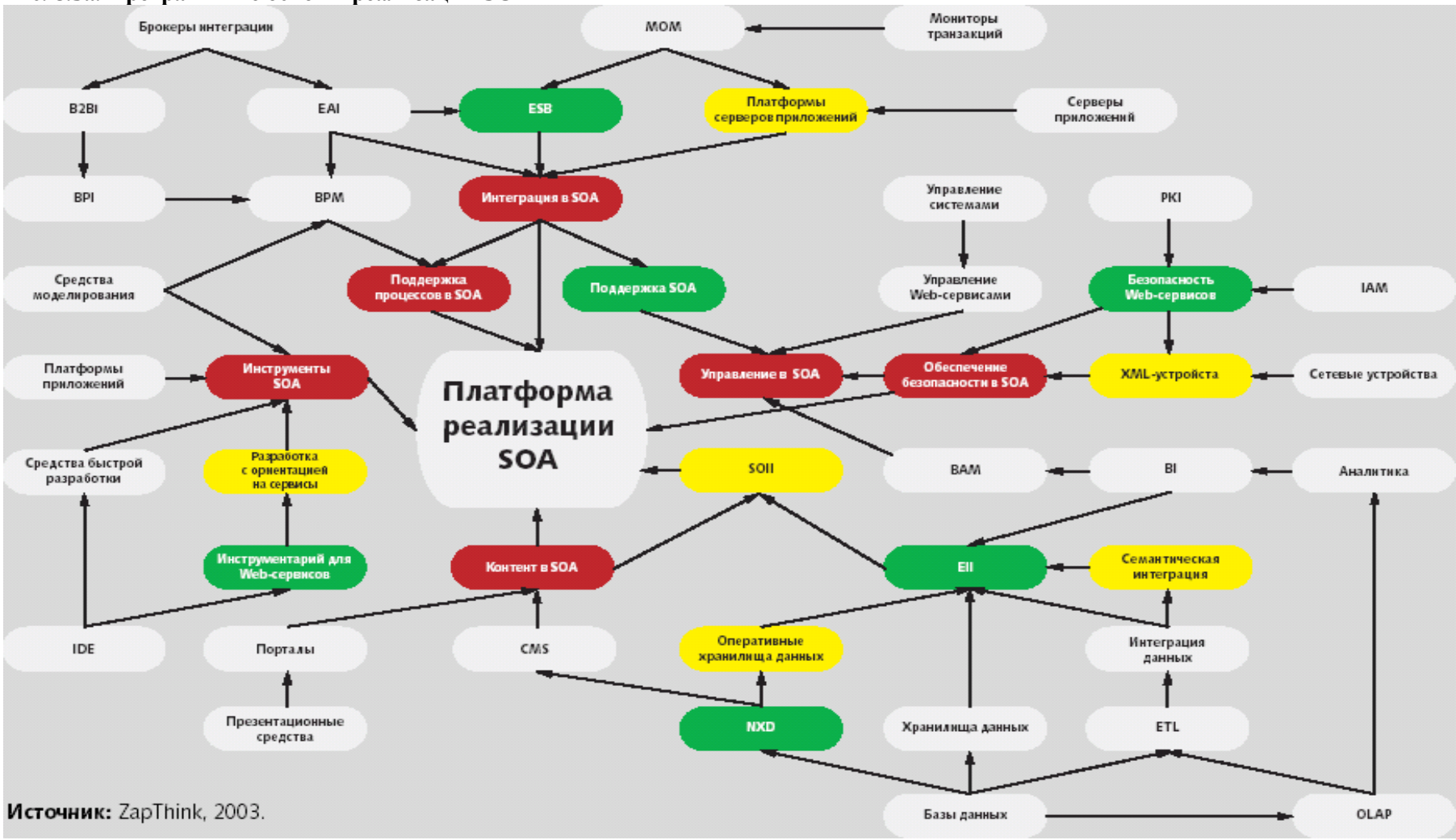
Одним из наиболее интересных средств описания оркестровки стал язык BPEL4WS (Business Process Execution Language for Web Services), предложенный компаниями IBM, Microsoft и BEA Systems. Язык предназначен для описания исполнения исполняемых бизнес-процессов и позволяет описать поведение и взаимодействия Web-сервисов в бизнес-процессе.

Появление языков класса BPEL4WS в определенном смысле логически завершает общую картину. Теперь можно — оговоримся, лишь условно — рассматривать SOA как метакомпьютер, на котором выполняются процессы, «запрограммированные» на языке описания процессов. Остается сделать всего несколько шагов, чтобы системы управления бизнесом по своей логической завершенности приблизились к техническим системам управления, а это открывает путь к применению кибернетических методов управления с их безграничными возможностями. Если учесть, что Web-сервисам понадобилось всего несколько лет, чтобы превратиться в первооснову для создания современных архитектур, то ждать новых открытий придется недолго.

8.4.8 Программные основы реализации SOA

На Рисунке 8.3а показаны основные компоненты прикладной корпоративной сети, на базе которых реализуются основтые технологии SOA и взаимодействуют известные системы интеграции типа ERP, CRM, B2B, B2C, др.

Рис. 8.3а. Программные основы реализации SOA



Источник: ZapThink, 2003.

8.5 *Модель, ориентированная на сообщения (MOM).*

Технологии обмена сообщениями в качестве основы для организации асинхронного взаимодействия между компонентами информационной системы вовсе не новы, они известны с 70-х годов, когда использовались в основном для общения между задачами, работающими на мэйнфреймах.

В последующем на фоне всеобщего «сдвига ИТ-парадигм», вызванного взрывным ростом Internet-технологий, в этой области сохранялась несколько необычная ситуация стабильности. Она настолько затянулась, что, например, в мае 2000 года в статье «Демистификация программного обеспечения промежуточного слоя», опубликованной в журнале CIO Magazine, средства MOM (message-oriented middleware) были названы «старыми и добрыми» (dear old MOM). И вот неожиданно эта «пожилая» технология оказалась в центре внимания. За последние два года она вышла из тени и попала в число самых «горячих». Что же произошло?

8.5.1 «Электронная почта» для корпоративных приложений

Итак, десятилетиями события, происходившие в лагере MOM, не отличались особой экстраординарностью. Все здесь развивалось на редкость спокойно. Первое оживление датируется 1993 годом, когда была образована ассоциация Message Oriented Middleware Association (MOMA). Затем на протяжении нескольких лет в этой области доминировали две всем известные корпорации со своими известными продуктами, а именно IBM с программным обеспечением MQSeries, которой принадлежало 75% рынка, а также несколько позже вышедшей на него Microsoft с продуктом MSMQ. Оставшийся небольшой сегмент достался компаниям существенно меньшего масштаба, скажем, Talarian с продуктом SmartSockets или Tibco с Tibco RV. Традиционное видение места средств MOM до событий последних дней в общей картине ПО промежуточного слоя описано в [17].

Изменение ситуации вызвано развитием разного рода Internet-приложений. Признание электронной почты в качестве классического примера «убойного приложения» (killer application), давшего сильнейший импульс развитию Internet, стало каноническим. Но это уже история, а вот теперь, когда не только люди, но и приложения вышли в Сеть, возникла логическая необходимость сделать

следующий шаг в этом направлении, установить привычный «почтовый» способ взаимодействия, но теперь не между людьми, а на этот раз между приложениями, наступил момент для конвергенции МОМ и электронной почты.

Справедливости ради надо сказать, что раньше специализированных ИТ-компаний идею интеграции корпоративных приложений на почтовых принципах, близкую к современному видению предложили специалисты из корпорации Cargill, крупнейшего дистрибьютора продуктов питания. Они построили ее непосредственно на базе корпоративной почтовой системы HP OpenMail.

А затем обнаружилось, что подобный подход соответствует более широкому классу приложений. Разумеется, речь не идет об универсальном подходе, и не ко всем приложениям вообще, а только о таких приложениях, где транзакции осуществляются в основном через Internet, т.е. для взаимодействий категорий business-to-business и business-to-consumer. Примечательная особенность этого типа приложений заключается в том, что в отличие от, скажем, сложных математических вычислений или управления объектами в реальном времени, эти приложения слабо связаны между собой (loosely coupled) и, следовательно, допускают асинхронный обмен сообщениями (asynchronous messaging). Именно такой стиль взаимодействия и реализует электронная почта.

Однако почему именно в последнее время так обострился интерес к новым системам для обмена сообщениями? Дело в том, что асинхронный обмен показал свою эффективность как коммуникационная модель при разработке критически важных приложений уровня предприятия. По сравнению с традиционными системами клиент-сервер, система с обменом сообщениями обладает большей масштабируемостью и гибкостью, она не требует, чтобы все приложения «шагали в ногу». (Английское слово lockstep, переведенное в данном случае как «в ногу», гораздо шире по смыслу, оно не просто означает единообразие шага, но еще и относится к строгим процедурам, требующим отказ от индивидуальности, бессловесное подчинение)

Приложения деловой направленности не нуждаются в подобной жесткости. Отсюда и возник интерес к более «демократическим» решениям.

8.5.2 Архитектура систем MOM

Теоретически существует два альтернативных архитектурных подхода к системам обмена сообщениями приложениями: топология «ступица и спицы» (hub-and-spoke) и шинная топология.

В централизованной архитектуре hub-and-spoke все приложения подключаются к центральному процессу, называемому сервером сообщений (message server), он отвечает за корректность маршрутизации сообщений, аутентификацию и авторизацию пользователей. Сами приложения рассматриваются как клиенты; они могут быть передатчиками и/или приемниками.

Централизация в стиле hub-and-spoke обладает рядом преимуществ.

- Ограниченное количество сетевых соединений: каждому приложению достаточно иметь одну единственную связь с сервером сообщений.
- Гибкость, поскольку маршрутизацией управляет сервер; приемники и передатчики независимы друг от друга, они могут изменяться с минимальной деформацией всей системы в целом.
- Минимальные требования к клиентскому программному обеспечению, поскольку вся логика соединений сосредоточена на сервере.
- Возможность обеспечения требуемой степени отказоустойчивости, один логический процесс на сервере может быть физически размещен на аппаратном сервере с кластерной конфигурацией.

Этих преимуществ лишена шинная архитектура, где каждое приложение (обычно оно подключается на сетевом уровне многоуровневого стека протоколов TCP/IP) должно обладать функциональностью сервера.

Но в этом году все радикально изменилось: появилась созданная в корпорации Sun Microsystems служба сообщений Java Message Service (JMS), для того чтобы обеспечить асинхронный обмен сообщениями по IP-сетям между слабо связанными Java-приложениями. И вот она-то и внесла заметную смуту в этот остров стабильности.

Однако если представить MOM на самом поверхностном уровне, то можно сказать, что прикладные системы, построенные на основе механизма обмена сообщениями,

целесообразно разделить на три основные категории в соответствии с тем, как именно организована дисциплина обмена.

1. «Точка-точка» (P2P — point-to-point messaging) где разные приложения могут посылать сообщения одному из них (рис. 1а). Клиент может выступать в роли передатчика, приемника, а также одновременно и приемника и передатчика. В JMS использована концепция очереди (queue). Передатчик помещает сообщение в очередь, а приемник их последовательно выбирает — полная аналогия с электронной почтой.
2. «Публикация и подписка» (P/S — publish-and-subscribe messaging) где множество приложений получают одно и то же сообщение . Сообщения помещаются в общую разделяемую область, называемую topic (в данном случае это слово можно перевести как «предмет обсуждения»). Помещенное туда сообщение становится доступным всем клиентам, которые могут выступать и в роли передатчика, и в роли приемника.

3. «По запросу» (request-reply messaging) где приложение посылает запрос приемнику.

8.5.3 Системы нового поколения

Итак, технологической основой для всех сред обмена сообщениями нового поколения стала спецификация Java Message Service, в деталях определяющая, как взаимодействуют клиенты и серверы в среде асинхронных сообщений (рис. 2). Она была очень быстро воспринята рынком. К числу достоинств JMS относится то, что она соответствует современным представлениям о взаимодействии приложений и не требует специальных знаний и доступна для любого программиста, работающего на языке Java. Этими качествами она заметно отличается от инструментария MQSeries, где необходима специальная подготовка. 21 месяц спустя после релиза J2EE 1.2, корпорация Sun Microsystems объявила о версии 1.3 Java 2 Enterprise Edition куда JMS уже включена как стандартный компонент.
Еще один стимул для второго рождения MOM — появление расширяемого языка разметки данных XML в качестве своеобразного «лингва-франка» для Internet-приложений, позволяющего одним приложениям понимать другие.

Спецификация JMS построена на основе топологии «ступицы и колеса», обеспечивая такой API-интерфейс на базе Java, который позволяет разработчикам писать клиентские приложения, не заботясь о том, какое конкретно программное обеспечение MOM будет использоваться. Спецификация только определяет доступ к корпоративной системе обмена сообщениями. При этом каждый производитель ПО, опирающийся на JMS, самостоятельно разрабатывает инструменты для администрирования среды обмена сообщениями.

Сегодня JMS стала основой целого инженерного направления; полноценно ее можно представить только в объеме большой книги, и такая книга уже есть. Первый труд, полностью посвященный JMS, написан коллективом авторов из нескольких лидирующих в этой области компаний [18].

Почти мгновенно JMS оказалась востребованной со стороны бизнеса. Вскоре появился класс новых программных продуктов категории MOM: MQ Express (компания Talarian), Tib/Rendezvous (Tibco Software), SonicMQ (Progress Software), iBus (Softwired) FioranoMQ (Fiorano Software), возможно, есть и другие. Эти новые представители программного обеспечения промежуточного слоя, построенные на фундаменте Java, оказались в состоянии составить конкуренцию MQSeries и по цене, и по гибкости, производительности и другим показателям — но не в одночасье. Как считают эксперты, никакого резкого отхода от MQSeries не будет.

Среди новых продуктов наибольшей популярностью обладают SonicMQ и FioranoMQ; сейчас именно эти две системы следует рассматривать в качестве наиболее близких конкурентов. SonicMQ обеспечивает полную реализацию JMS; помимо моделей P2P и P/S поддерживаются клиенты ActiveX/COM. В FioranoMQ также реализованы все основные функции сервера сообщений, есть возможность интегрировать его с другими серверами приложений. FioranoMQ и SonicMQ обеспечивают работу с такими расширениями, как сообщения XML и кластеризация серверов. Хотя эти две программные системы часто сравнивают между собой, однако в качестве эталона обе компании — Progress Software и Fiorano Software — неизменно используют MQSeries.

Существует несколько свойств, по которым их можно сравнивать между собой, например, по качеству управления потоками, по способности к масштабированию и по времени задержки сообщений. Одним из важнейших показателей серверов сообщений, естественно, является их способность управлять потоками сообщений. Обычно скорость, с которой сообщения производятся, превосходит скорость их

потребления, поэтому очень важно «дросселировать» эти потоки, чтобы исключить потерю сообщений. В частности, для модели «публикации и подписки» существенно, чтобы самый медленный клиент не тормозил работу всей системы в целом.

Общепризнанной выработанной методики для сравнения систем MOM пока еще не существует. Поэтому у каждого из производителей есть возможность доказать преимущество своих программных продуктов над другими. Если кому-то покажется это интересным, рекомендую скачать с сайтов Progress Software и Fiorano Software документы под приблизительным названием «High-performance Messaging with JMS» и убедиться в их взаимной ортогональности. Каждая из компаний умело доказывает преимущество собственной разработки над другой и над IBM с ее MQSeries.

8.5.4 Общая шина предприятия

В стремительном процессе эволюции компьютерных технологий иногда совершенно неожиданно раскрываются новые возможности старых идей. Например, такой, как, казалось бы, давно забытая память на ферритовых сердечниках. При всех недостатках ее положительным качеством была способность запоминать и сохранять данные бесконечно долго без подпитки — в отличие от современных модулей RAM на полупроводниках. И вот буквально на наших глазах она возрождается в виде многообещающей технологии MRAM, в которой точно так же используется двухпозиционность петли магнитного гистерезиса. Следовательно, она будет сохранять свое состояние без питания. С возрождением магнитной памяти уйдет в прошлое такая привычная процедура загрузки компьютера.

Совсем недавно идея «общей шины», как центрального узла, вновь неожиданно появилась на горизонте, однако на этот раз в качестве средства для интеграции приложений в масштабах предприятия.

Нечто подобное происходит и с магистралями обмена данными, построенными по принципу «общей шины». Сейчас трудно оценить революционность идеи общей шины, а ведь в свое время это был настоящий переворот. Общая шина Unibus, предложенная три десятка лет назад инженерами корпорации Digital Equipment в

качестве архитектурной основы для миниЭВМ PDP-11, оказалась чрезвычайно эффективным (а главное, дешевым) средством интеграции разнотипных устройств. В последующем на шинном принципе было построено множество компьютеров, в том числе все современные ПК. Собственно, с общей шины и начал формироваться рынок периферийных устройств. Однако со временем шины, используемые в качестве центрального архитектурного элемента компьютера, стали уступать свое место более быстродействующим коммутаторам, оставаясь при этом одним из основных вариантов подключения периферийных устройств. Сегодня шина, которую называют Enterprise Service Bus, может сыграть примерно ту же роль, что и шина Unibus, со всеми достоинствами, но на более высоком уровне.

События и в самом деле развиваются стремительно. Всего лишь год назад один из ведущих аналитиков Gartner Group Ефим Натис высказал следующее предположение: «Один из основных подходов к созданию корпоративной инфраструктуры приложений строится с использованием слабосвязанных асинхронных процессов». А уже в октябре 2002 года в еженедельнике InfoWorld в статье Джона Уделла можно было прочитать: «Теперь, когда мы все согласны с тем, что Web-службы должны взаимодействовать в асинхронной манере, стало ясно, что программное обеспечение промежуточного слоя, ориентированное на обмен сообщениями (message-oriented middleware, MOM), приобретает решающее значение».

Как видим, всего за год предположение превратилось в утверждение. В том, что это произошло, заметную роль сыграла компания Sonic Software, образованная несколькими выходцами из BEA Systems и сегодня признаваемая в качестве одного из лидеров в разработке программного обеспечения промежуточного слоя. Очень интересные работы проделаны еще в нескольких небольших компаниях (например, Collaxa), однако Sonic одной из первых предложила свою реализацию слабосвязанных асинхронных процессов. При всей новизне, в своем программном продукте SonicXQ ESB компания, по сути, реализует старую, заимствованную у миниЭВМ идею общей шины, но при этом воплощает ее в новом обличии.

В данном случае шина ESB (Enterprise Service Bus) является общей в том смысле, что объединяет все приложения предприятия. ESB, реализованная с использованием архитектуры SOA (Service-Oriented Architecture), предназначена для интеграции корпоративных приложений на основе ориентированных на документы асинхронных Web-служб и J2EE Connector Architecture (JCA). Две этих

технологии обеспечивают контентную маршрутизацию сообщений и позволяют так организовать взаимодействие между приложениями, и так интегрировать управление бизнес-процессами, что появляется возможность обойтись без дорогостоящих брокеров.

Оригинальность разработки SonicXQ привлекала к себе значительное внимание. Исторически первыми появились интеграционные брокеры (иногда их называют интеграционными серверами). Решения, построенные на основе интеграционных брокеров, можно представить в виде коммутаторов. С их помощью формируется некоторый гипотетический метакомпьютер, где все управление строится по централизованному принципу. В результате получается что-то вроде гипер-мэйнфрейма. Sonic совершила примерно то же самое, что DEC, предложившая три десятилетия назад шинные миниЭВМ в качестве недорогой альтернативы мэйнфремам; решение Sonic позволяет построить своего рода метакомпьютер для всего предприятия, но более дешевый. В итоге получается аналог мини-метакомпьютера: вместо дорогого коммутатора предлагается информационная магистраль предприятия Enterprise Service Bus.

Технология SonicXQ появилась не вдруг. У нее два достаточно хорошо известных источника. Первый — программное обеспечение промежуточного слоя на основе сообщений. Этот тип программного инструментария переживает настоящую реинкарнацию, особенно в связи с появлением Java Message Service от компании Sun Microsystems. О происходящем на этом фронте можно прочитать в [1], а более подробно о SonicMQ, непосредственном предшественнике SonicXQ, — в [2]. Обе эти публикации сохраняют актуальность, но за прошедший год пейзаж корпоративного программного обеспечения заметно изменился, особенно под воздействием Web-служб. Еще год назад, когда готовились указанные публикации, представление о том, что такое Web-службы и каково их значение, было достаточно расплывчатым. За прошедшее время ситуация заметно прояснилась, и Web-службы следует назвать в качестве второго источника SonicXQ.

8.5.5 Enterprise Service Bus - информационная магистраль

Среди событий прошедших лет следует отметить появление в профессиональной терминологии нечто нового и непривычного. Одни, приверженцы лагеря Micrososft/IBM, называют это «оркестровкой» (orchestration) Web-служб, другие, из лагеря Sun/BEA, — «хореографией» (choreography) [19]. Разгорается очередная

битва в войне стандартов, за то, как лучше наладить согласованную работу корпоративных приложений с использованием Web-служб. Причина новой активности заключена в том, что всем, наконец, стало ясно: в сложившихся условиях исчерпаны возможности жестко связанных приложений, сложность систем стала слишком велика. Однако исходная схема распространения Web-служб с использованием построенных по стандарту UDDI репозиториев оказалась малоприменимой для корпоративных целей. В то же время, Web-службы и особенно их асинхронные ориентированные на документы версии предлагают реальный выход из «тупика сложности». С технической точки задача создания корпоративной инфраструктуры приложений с использованием слабосвязанных асинхронных процессов имеет несколько альтернативных решений.

Enterprise Service Bus, построенная на основе SonicXQ является одним из них. С помощью формируемой SonicXQ корпоративной магистрали реализуется распределенная архитектура, ориентированная на службы. ESB позволяет создавать контейнеры для размещения служб. Службы легко собрать и согласовать, поскольку упакованная в контейнер и являющаяся частью ESB служба представима другим частям ESB. При этом вся конструкция является виртуальной; реальная физическая сеть, в которой она «живет», может подвергаться изменениям без потери функциональности.

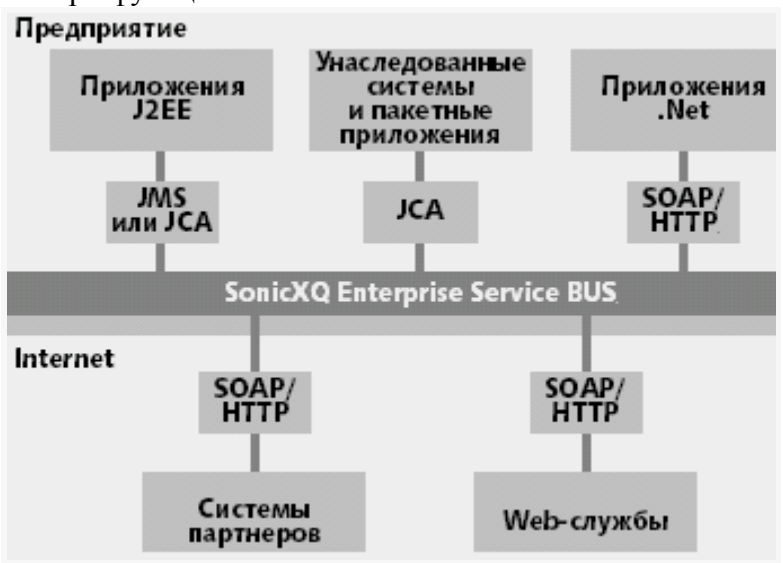


Рис. 8.4. Enterprise Service Bus — информационная магистраль предприятия

В процессе функционирования ESB одна или несколько связанных служб находятся в специальном контейнере (service container). Контейнеры являются средством для продвижения служб по распределенному процессу в соответствии с

маршрутами сообщений (message itinerary). Процедура прохождения сообщения выглядит следующим образом. Сообщение поступает на вход шины ESB. Здесь к нему добавляется маршрут, который позволяет организовать контентно-управляемое продвижение по распределенному процессу, этот процесс имеет децентрализованное управление. В рамках этого процесса сообщение проходит через ряд служб, достигая конечной точки, где извлекается из контейнера.

Для указания конечных точек могут быть использованы не физические, а логические имена. Установление соответствия между физическими и логическими именами (mapping) осуществляет специальный имеющийся в составе ESB механизм. Таким образом, в архитектуру изначально заложена способность к виртуализации; система может изменяться без модификации кода и разрушения действующих бизнес-процессов. Конфигурация допускает несколько уровней качества обслуживания (Quality of Service, QoS), гарантирующих надежное прохождение сообщений между приложениями. В общем случае, когда сообщение проходит весь свой маршрут, оно выходит за конечную точку получателя, а отправителю посылается подтверждающее получение сообщение. Достоинство распределенного процесса передачи сообщений на основе ESB заключается в том, что по своей логике он очень близок взаимодействию в реальном мире.

8.5.6 Основы: JCA и Web-службы

Предлагаемая в ESB интеграция приложений стала возможной благодаря появлению архитектуры соединений JCA от Sun Microsystems и SOAP, стандартного протокола для Web-служб. JCA, специально разработанная для преодоления сложностей, связанных с интеграцией приложений, предлагает стандартизированные методы для решения этой задачи. Корпоративная информационная система, построенная по принципам JCA, использует для доступа к приложениям интерфейс JDBC. Сегодня этот подход весьма популярен; большинство современных серверов приложений, в том числе, BEA WebLogic и IBM WebSphere поддерживают адаптеры JCA. Кроме того, многие поставщики пакетных решений намерены поддерживать JCA в будущих версиях своих продуктов.

Оригинальность использования Web-служб в SonicXQ заключается в том, как организован процесс «оркестровки» (или «хореографии»). В его основе лежит протокол SOAP, но наложенный на простой и масштабируемый формат сообщений. При этом SonicXQ Enterprise Service Bus обеспечивает совместимость и с асинхронной документальной моделью SOAP (document asynchronous model), так и с синхронной моделью SOAP, построенной по принципу вызова удаленных процедур (RPC). В SonicXQ службы описываются на языке WSDL, но WSDL непосредственно интегрирован Distributed Processing Framework. В результате служба может быть зарегистрирована во внешнем каталоге UDDI, а может и не быть, если в этом нет необходимости.

Технологию SonicXQ можно без всякого преувеличения назвать экстремальной: она собрала в себе целый ряд современных тенденций в корпоративных вычислениях. Но, пожалуй, самое интересное заключается в том, что она позволяет лучше понять, что такое Web-службы. Причем не на словах, а на деле.

8.3 Модель, ориентированная на сервисы (SOM). Web-сервисы и Grid- сервисы.

В 2004 году grid-сервисы и Web-сервисы «встретились». И тут обнаружилось, что ни в тех, ни в других нет того, что позволило бы превратить набор компонентов в систему.

Вполне вероятно, в будущем удастся реализовать идеи utility computing, организовав доступ к компьютерным ресурсам как к другим коммунальным службам. Тогда grid как основа распределенных вычислений с полным основанием будет причислен к разряду великих инженерных достижений человечества, которые формируют нашу среду существования и без которых мы ее просто не представляем. Однако если подобное и случится, то, к сожалению, не завтра, что обидно — особенно для апологетов идеи создания всеобщей распределенной вычислительной среды.

История технологий свидетельствует, что путь идей подобного масштаба от замысла до воплощения никогда не был простым. Не одно десятилетие ушло на то,

чтобы аэропланы превратились в современные самолеты, а наивные проекты космических летательных аппаратов воплотились в почти рутинные полеты на Международную космическую станцию. Это утверждение справедливо и по отношению к grid (даже с поправкой на то, что ИТ внедряются быстрее традиционных технологий). И этой идее предстоит пройти отмеренный ей путь.

В новом столетии невиданные прежде возможности распределенных вычислений обеспечены, помимо Internet, рядом основополагающих технологий и факторов:

- современные сетевые технологии (в том числе Gigabit Ethernet и Infiniband);
- кластерные технологии, серверы-лезвия и многоядерные процессоры;
- интеграция приложений и построение систем с использованием сервисов.

Как известно, количество рано или поздно переходит в качество. Наступил момент, когда сумма технологий computing + communication достигла той критической массы, которая обусловила формирование новой реальности. Ключевыми словами, определяющими эту реальность, стали «доступ» и «сервисы», а характеризуется она следующими факторами:

- доступ к вычислительным ресурсам, данным, устройствам, измерительным инструментам должен быть беспрепятственным, прозрачным, удаленным, безопасным и беспроводным;
- доступ должен быть виртуальным (нужен доступ не к серверам, а к сервисам, поставляющим данные или вычислительные ресурсы — причем без необходимости знания инфраструктуры, обеспечивающей эти сервисы);
- доступ должен осуществлять по требованию (с заданным качеством), а ресурсы должны предоставляться тогда, когда в них возникает нужда;
- доступ должен быть распределенным, допуская организацию совместную коллективную работу виртуальных команд;
- доступ должен быть устойчив к сбоям, а при выходе из строя серверов приложения должны автоматически мигрировать на резервные серверы;
- доступ должен обеспечивать взаимодействие с гетерогенными платформами.

На фоне захватывающих перспектив этой новой реальности сохраняется дисбаланс между собственно технологиями и способностью к осмыслению их возможностей. Как обычно, техника явно опережает науку. Очевидный недостаток «научности» (а зачастую и просто «паранаучность») взглядов отраслевых «проповедников»

заставил их искать простые объяснения. Не обремененные излишком фундаментальных знаний, в поисках точки опоры они обратили взоры на прежде отдельное скромное направление grid, в котором увидели многие признаки новой реальности. Надо сказать, прежде увлечение grid казалось скорее чудачеством, и это направление представляло интерес в основном для ученых и искателей внеземных цивилизаций.

Хотя термин был занят, евангелисты-технологи стали усматривать в нем новый смысл. Проповедуемые ими подходы к grid начали квалифицировать как «коммерческие».

Сложность ситуации усугубляется тем, что коммерческие grid-решения и близкие им инициативы называют по-разному: в IBM и Computer Associates — вычислениями по требованию (On-demand Computing), в Hewlett-Packard — адаптивным предприятием (Adaptive Enterprise), в Sun Microsystems — стратегией N1, в Microsoft — Dynamic Systems Initiative, а в Oracle — «попросту» Grid Computing. Но при всем различии фирменных названий суть таких начинаний примерно одна и та же. В конечном счете она сводится к попытке создать условия для динамического распределения вычислительных ресурсов и ресурсов хранения. Общность между перечисленными подходами заключается еще и в том, что соответствующие им решения так или иначе реализуют архитектуры, ориентированные на сервисы (service-oriented architecture, SOA), которые, в свою очередь, обеспечивают управление бизнес-процессами (business process management, BPM).

Несмотря на поднявшуюся пропагандистскую волну, говорить о практическом внедрении коммерческих grid-решений еще рано.

8.5.7 Grid как результат эволюции

И все же, несмотря на шумиху, коммерциализацию и «захват» самого термина, «возвышение» grid вполне объективно. Обращения к истории компьютерной отрасли (рис. 8.5) достаточно, чтобы последовать совету Н.К. Рериха — «смотреть в прошлое, чтобы понять настоящее и в настоящем прозреть будущее».

Сорокалетний период ее развития можно интерпретировать как движение к распределенной среде, позволяющей оптимально использовать вычислительные ресурсы, — пусть это будет grid (назовем кошку кошкой).

Рис. 8.5. Коэволюция вычислительных и сетевых структур

Все начиналось с мэйнфреймов, работавших в пакетном режиме и выполнявших задания последовательно, одно за другим. Среда была централизованной, КПД — низким. Для повышения эффективности работы мэйнфреймов были созданы операционные системы, предусматривающие режим разделения времени, затем появились суперкомпьютеры, еще позже — мощные Unix-серверы и, наконец, центры обработки данных. Все эти системы представляют собой централизованные решения с распределенным доступом. Был, правда, и инициированный академиком В.М. Глушковым утопический проект ОГАС (общегосударственная автоматизированная система сбора и обработки информации) для учета, планирования и управления (www.mccme.ru/free-books/djvu/bib-kvant/ogas.htm), который, впрочем, остался нереализованным.

С отставанием лет на десять началось развитие децентрализованного направления — с работ Джозефа Ликлайдера и Роберта Тейлора, ставших затем основой проекта ARPANET. Позднее появились протоколы Ethernet и TCP/IP и соответственно возможности построения локальных и глобальных сетей. Наступление XXI века принесло два открытия, Web-сервисы и XML, которые радикально изменили представления о возможностях организации децентрализованных структур и дали новый импульс к формированию идей grid.

Собственную историю grid можно разделить на два периода. В течение первого из них (1998-2002 годы), «классического» периода предпринимались попытки

создавать сетевые структуры для использования географически разнесенных и находящихся в разном подчинении вычислительных мощностей.

По функциональному назначению grid-среды можно разделить на предназначенные для вычислений (computational grids) и хранения данных (data grids). Первые стандарты спецификаций предложил Global Grid Forum, а инструментарий Globus Toolkit является фактическим стандартом программного обеспечения промежуточного слоя для grid.

Летом 2002 года наступил новый период; стройная картина была нарушена. Возмутителем спокойствия оказалась компания Sun Microsystems. ее сотрудники Джеймс Кумер и Чару Чаубал в серии Sun BluePrints OnLine выпустили многостраничный труд Introduction to the Cluster Grid. В нем они стали обозначать термином grid кластерные конфигурации разных масштабов, от уровня подразделения до глобального уровня. Зачем Sun, которая отождествляла сеть и компьютер, было посягать на уже «занятое» название, понять сложно, но именно с этого момента началась дискуссия о том, что такое кластер, а что — grid.

С тех пор к ней подключились представители индустрии и ученые, позиции которых обуславливают выбор ими того или иного определения. Теперь так называют и небольшую кластерную конфигурацию, собранную в одной стойке из серверов-лезвий (в этом особенно преуспела компания Oracle), и классические конструкции, основанные на принципах Open Grid Service Architecture (OGSA), которые пропагандирует корпорация IBM.

Распространение языка XML и Web-сервисов дало старт конвергенции grid и Web-сервисов, которая привела к формированию концепции Grid-сервисов. В рамках Global Grid Forum начались работы по стандартизации в новой области. Предложенная архитектура объединяет основные технологии Grid с Web-сервисами, что позволяет создавать каркасы инфраструктуры для интеграции, виртуализации и управления разнородными ресурсами в виртуальной организации.

Для создания справочной модели (reference model) была организована рабочая группа Open Grid Service Infrastructure. В июне 2003 года она выпустила спецификацию OGSI, в которой определены механизмы создания, управления и обмена данными между Grid-сервисами. Одним из последних событий в этой цепи стало появление Web Service Resource Framework (WSRF), где еще точнее отражена

конвергенция двух миров. Как и любые другие Web-сервисы, сервисы «по OGSA» могут быть определены в терминах языка WSDL, что позволяет использовать преимущества известных стандартов SOAP, XML и WS-Security.

Помимо Global Grid Forum вопросами стандартизации теперь ведает и созданный летом 2004 года Enterprise Grid Alliance — консорциум ИТ-производителей, заинтересованных в развитии идей grid. Создавая свой альянс, они руководствовались стремлением применять технологии grid в бизнес-приложениях, а не для решения научных или технических задач, требующих большой вычислительной мощности. Поэтому в центре внимания альянса оказались статические grid-структуры, расположенные в одном географическом месте и являющиеся составляющей корпоративного центра обработки данных. Намерения участников альянса прагматичны; в основном они сводятся к адаптации существующих стандартов, а не к созданию новых.

8.5.8 Неполная сумма вендоров?

Предложение об организации консорциума Enterprise Grid Alliance (EGA) прозвучало на конференции OracleWorld в сентябре 2003 года, одновременно с анонсом Oracle 10g. Его основателями стали EMC, Fujitsu Siemens, HP, Intel, NEC, Network Appliance, Oracle, Sun Microsystems и целый ряд меньших по размеру компаний, таких как AMD, Ascential Software, Cassatt, Citrix, Data Synapse, Enigmatec, Force 10 Networks, Novell, Optena, Paremus и Topspin. В ноябре 2004 года в состав консорциума вошла компания Dell; однако в нем до сих пор не числятся ни Microsoft, ни IBM.

В соответствии с программными документами Enterprise Grid Alliance представляет собой консорциум основных производителей и потребителей технологий работы с данными, основной целью которого является разработка решений, ориентированных на построение корпоративных grid-сред (Enterprise Grid).

- EGA сфокусировал внимание на двух классах приложений. Один из этих классов — коммерческие приложения с большой транзакционной составляющей, такие как ERP, CRM или бизнес-аналитика. Второй — так называемые «технические приложения».
- EGA намерен заниматься как стандартизацией, так и маркетингом, поскольку в современных условиях одно без другого невозможно. Однако, с

учетом отрицательного опыта, EGA постарается не создавать избыточной шумихи.

- EGA намерен сотрудничать с другими организациями, занятыми стандартизацией в области grid, разумно пользуясь накопленным ими опытом.
- EGA не против членства в организации корпораций IBM и Microsoft, но на общих принципах, таких как «одна компания — один голос» и «равенство в оплате членства».

Несмотря на громкие заявления, сделанные более года назад, пока каких-либо конструктивных предложений EGA не сделал. Почему? Ответ, по всей видимости, можно найти в работе Кристиана Антонини (Is Oracle Database Moving Toward Grid Computing, www.trivadis.com): «Oracle Database 10g не относится к категории Grid! Во всяком случае, не соответствует определению, которое используют специалисты, уже более десяти лет работающие в этом направлении. Можно сказать, что некоторые черты приближают 10g к grid-вычислениям, и компания явно движется в верном направлении. История повторяется, и, обратившись к прошлому, мы обнаружим схожую ситуацию. В 1999 году появилась первая СУБД для Internet Oracle8i, но практически ни одно из качеств Internet-компьютинга в ней не было реализовано — скорее всего, Gridaware мы обнаружим в версии 11 или 12».

8.5.9 Grid-сервисы и все, все, все

В поисках решения, которое обеспечит распределенный доступ к сетевым ресурсам, группа исследователей из Чикагского университета, возглавляемая Йаном Фостером, поддерживаемая IBM и Национальной арагонской лабораторией, разработала архитектуру Open Grid Service Architecture (OGSA). Получилось решение со специфическим университетским и академическим «акцентом», основанное на специализированных grid-сервисах. Их можно рассматривать как надстройку над обычными Web-сервисами (см. Рис.8.6). Новый тип сервисов был предложен для утверждения в Global Grid Forum, где его одобрили и предложили рассматривать в качестве средства реализации Open Grid Service Initiative (OGSI).

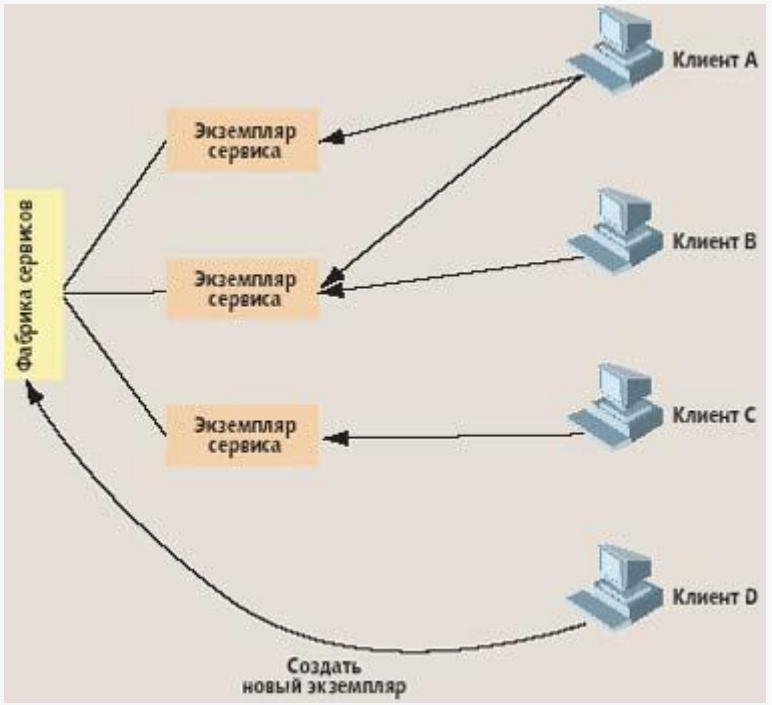


Рис. 8.6. Архитектура factory/instance

Так началась полоса самостоятельного развития grid-сервисов, которая практически закончилась в 2004 году с появлением Web Services Recourse Framework (WSRF). В качестве практического инструмента для создания grid-сред тогда был предложен Globus Toolkit. Его последняя версия, GT3, основана на представлении о сервисах. Набор GT3 сменил GT2 и GT1, однако на его базе, как и на основе его предшественников, не было создано заметного числа работающих систем.

Отметим, что открытую архитектуру grid-сервисов создали в 1998 году, а это, по любым меркам, было совсем недавно. На тот момент признанные сегодня сервис-ориентированные архитектуры и Web-сервисы находились в зародышевом состоянии, и решение использовать их именно так, а не иначе было вполне оправданным. Как следствие, на протяжении пяти лет два подхода к идее использования сервисов в качестве системообразующего механизма развивались не параллельно. Две непараллельные прямые рано или поздно пересекутся; в данном случае точкой пересечения стал стандарт WSRF.

Рис. 8.7. OGSA, OGSI, GT3 и grid-сервисы

На рис 8.7 представлены основные составляющие архитектуры OGSA, которая определяет общие стандарты для приложений и интерфейсы к таким типам grid-сервисов, как управление работами, ресурсами и обеспечение безопасности. Средством для реализации архитектуры, которая определяет работу приложений в grid-среде, аналогичной OGSA, могло бы стать распределенное программное обеспечение промежуточного слоя, базирующееся на различных технологиях (в том числе CORBA, RMI или RPC). Но авторы OGSA предпочли архитектуру на основе сервисов, точнее — на основе расширенного набора Web-сервисов, названных grid-сервисами. Спецификация OGSI является формальным описанием того, что представляют собой grid-сервисы. Наконец, GT3 — это набор инструментальных средств, реализующих OGSI.

Взаимосвязь между этими тремя уровнями представления инфраструктуры, работающей на основе сервисных принципов, полностью соответствует тому, что происходит при создании физических конструкций. Сначала средствами OGSA фиксируется архитектурный замысел, затем с использованием OGSI выполняется инженерный проект, а из компонентов GT3 строится сооружение с grid-сервисами.

Между grid-сервисами и Web-сервисами обнаруживается ряд различий, которые можно было бы назвать «глубиной» услуги, тем, насколько «далеко заходит» сервер в предоставлении услуги (grid-сервер заходит «дальше»). Главное различие

состоит в том, что Web-сервисы не имеют состояния, а срок их жизни определяется поставщиком услуги, то есть они существуют независимо от потребителя. Грубо говоря, сервер не подстраивается под клиента, а просто дает ему возможность воспользоваться той или иной услугой, и на этом все заканчивается — полная аналогия с бытовым пониманием сервиса.

Авторы grid-сервисов считают основным преимуществом их интерпретации идеи обслуживания то, что предлагаемые ими сервисы имеют состояние и ограниченный потребностями пользователя срок жизни. Они представляют сервер не как что-то, оказывающее услуги, а как фабрику, на которой по заказу клиента создается нужный сервис, используемый в заданное время. Этот подход так и назвали — factory/instance, то есть буквально «фабрика образцов».

Сам сервер услуг не оказывает, он лишь фабрикует некий «экземпляр» услуг, осуществляющий сервис от его имени. С точки зрения вычислений это решение не вызывает возражений. Если требуется решить какую-то математическую задачу, где-то в Сети под нее создается виртуальный компьютер, предоставляющий нужный сервер. Физический сервер может поддерживать множество подобных виртуальных вычислительных машин, создаваемых под требования клиента и уничтожаемых, когда в них отпадает необходимость.

Виртуальному временному компьютеру требуется поддержка данными. Этой цели служит еще одна опция grid-сервисов. Service Data позволяет использовать наборы структурированных данных, которые распадаются на две категории:

- сведения о состоянии (state information), в том числе промежуточные и окончательные результаты вычислений;
- сервисные метаданные (service metadata), то есть сведения, относящиеся непосредственно к сервису, в том числе системные данные, поддерживаемые интерфейсы, стоимость услуги и др.

Для удобства пользования клиенты-подписчики могут уведомляться об изменениях в обслуживании. Инициатива OGSI предполагает, что сервисы могут группироваться, образуя сервисные группы, «члены» которых имеют одну точку входа. Получается что-то вроде экземпляра сервиса, состоящего из нескольких сервисов (см. Рис.8.8).

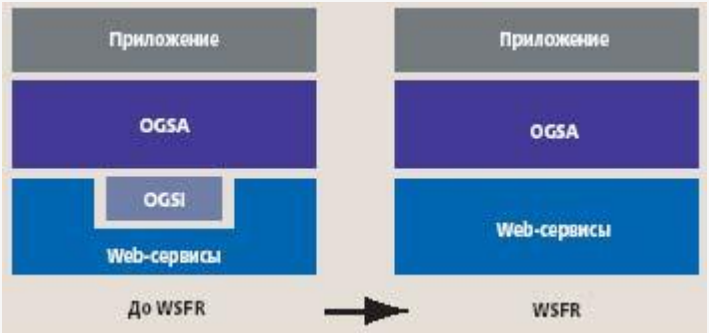


Рис. 8.8 От OGSI до WSRF

Наконец, еще одна группа отличий связана с практическими способами работы с сервисами. В Web-сервисах для адресации используются URI, а в grid-сервисах — расширенная схема адресации Grid Service URI или Grid Service Handle (GSH). А информация о том, как взаимодействовать с сервисами, находится в Grid Service Reference.

8.5.10 Событие января 2004 года

В OSGI была заложена концептуальная основа grid, но обеспечиваемое ею квазистабильное состояние grid-сервисов продолжалось недолго.

В январе 2004 года компании BEA Systems, Microsoft и Tibco опубликовали свою спецификацию WS-Eventing для Web-сервисов, основанную на принципах публикации/подписки. Она не привязана непосредственно к grid, а предназначена для описания коммуникаций между событиями в архитектуре Web-сервисов. Фактически принцип публикации/подписки призван помочь в создании временной среды, имеющей состояние (этой цели пытались достичь и авторы OSGA, но другими средствами). Буквально через две недели компании Akamai, Globus Alliance, Hewlett-Packard, IBM, Sonic Software и все та же Tibco предложили спецификацию WS-Notification, входящую в состав WS-Resource Framework и опять-таки реализующую принцип публикации/подписки.

Разработку упомянутых спецификаций вполне можно считать первыми серьезными попытками компенсации недостатков, характерных для нынешних стандартов на Web-сервисы. Это, в конечном счете, те же самые действия, которые предпринимались создателями OGSI. Одновременность объявлений о создании

спецификаций свидетельствует, с одной стороны, об острой конкурентной борьбе, а с другой — о незрелости технологий, поскольку реальных результатов таких заявлений можно ожидать не раньше середины 2005 года. Положительным эффектом от появления альтернативных подходов можно считать, подход по принципу публикации/подписки, что гораздо образнее и точнее, чем «создание экземпляров». Разногласие в отношении стандартизации Web-сервисов между лагерями IBM и Microsoft совместно с BEA Systems несколько необычно — одна только Tibco присутствует сразу в двух лагерях. Это вызвало пересуды среди аналитиков, однако все они сходятся в том, что сосуществование двух подходов имеет временный характер и в недалеком будущем удастся выработать общее решение.

В январе аналитикам казалось, что конвергенция WS-Eventing и WS-Resource Framework произойдет примерно к 2006 году, но реальные события опережают прогнозы. В августе к лагерю приверженцев WS-Eventing присоединились IBM, CA и Sun. Кроме того, EDS, CA и ряд других компаний опубликовали предложения по языку управления центрами обработки данных Data Center Markup Language (DCML), который должен закрыть пробел между двумя точками зрения на SOA в отношении предоставления услуг, обслуживания и управления. Эти предложения, наряду с WSRF и WS-Eventing, переданы в OASIS. Трудно сказать точно, что сейчас происходит, но одно очевидно: в 2005 году есть основания ожидать окончания смутного времени для мира grid.

8.5.11 Этот загадочный рефакторинг

Практика показала: Web-сервисы прижились гораздо быстрее, чем grid-сервисы, которые, несмотря на громкие заявления, остались в основном на бумаге. Одно из объяснений замедленного развития grid заключается в том, что создававшее Web-сервисы сообщество оказалось прагматичнее, поскольку задалось целью создать жизнеспособную универсальную архитектуру, ориентированную на сервисы и способную работать через Internet. Эта цель конкретна, реализуема и, естественно, собрала вокруг себя достаточные силы.

grid-сообщество изначально представило свою задачу как значительно более узкую, ограничив ее распределением ресурсов (вычислительные данные и даже научные

приборы) и созданием виртуальных организаций. Сработал традиционный механизм академического мышления, нарочитой замкнутости «яйцеголовых». Ситуация поневоле напоминает противоборство Windows и Unix, происходившее «на поле» операционных систем в середине 90-х.

Если же перевести объяснения из области общих рассуждений на программистский уровень, неуспех OGSI можно объяснить несколькими обстоятельствами. Во-первых, спецификация OGSI слишком сложна и длинна (а если признать, что все спецификации длинны, то особенно длинна). Во-вторых, в ней плохо учтены существующие средства создания Web-сервисов. В-третьих, она слишком объектно ориентирована (Web-сервисы оказались фактической альтернативой объектному подходу). Последний недостаток расценивают как один из самых существенных; он является чуть ли не главным родовым признаком grid-сервисов (отметим сохранение состояний и модель factory/instance model).

Подвергая критике OSGI как средство реализации OGSA, никто не сомневается в достоинствах архитектуры OGSA — просто требуется иная реализация. Чтобы избавить OGSA от перечисленных проблем и обеспечить конвергенцию двух типов сервисов, в январе 2004 года была предложена структура Web Services Resource Framework (WSRF), которая отличается отсутствием промежуточного слоя в виде OGSI. Важно, что новый стандарт влияет не на всю концепцию OGSA — на верхнем уровне все остается неизменным. Принципиально новым должен стать инструментальный набор GT4, хотя его авторы и утверждают, что на концептуальном уровне между OGSI и WSRF большой разницы нет и что различия имеют лишь синтаксический характер.

Со времени выхода GT3 прошло совсем немного времени. Чтобы избежать неловкости и сохранить лицо, авторы OGSI нашли удачное слово для наименования перехода от OGSI к WSGF, назвав его рефакторингом (от factoring — «разложение на множители»). Не ищите это слово в словарях; оно укрепилось в программистской терминологии с подачи Мартина Фаулера, автора книги «Рефакторинг. Улучшение действующего кода», который определяет рефакторинг как процесс изменения программного обеспечения, внешне не изменяющий поведения программного кода, но способствующий улучшению его внутренней структуры.

Разделение модификации на два этапа — рефакторинга и собственно модификации — дает определенные преимущества. Гораздо проще сначала усовершенствовать сам код, не меняя его функциональности, а затем — функциональность предварительно вычищенного кода. Со временем область применения нового термина расширилась, и теперь его используют при определении продвижения вперед небольшими шагами с попеременным совершенствованием внутренней структуры и функциональных характеристик системы. Рефакторинг — своего рода реорганизация, не предполагающая полной переделки.

8.5.12 Знакомьтесь, WSRF

8.5.12.1 Что собой представляет справочная структура Web-сервисов (Web Service Reference Framework, WSRF)?

Скорее всего, в перспективе WSRF будет представлять собой набор из шести спецификаций, которые поддерживают grid-сервисы и другие ресурсы, имеющие собственное состояние (stateful). Это — «стандарт» в том смысле, который подразумевается в англоязычном варианте данного слова. Цель, преследуемая при создании таких спецификаций, заключается в сближении OGSA с Web-сервисами и SOA. С помощью средств, соответствующих этим спецификациям, может быть реализован подход к моделированию и управлению состоянием в контексте Web-сервисов. Иными словами, делается попытка реализовать «состояние» — именно то, что отличает grid-сервисы от Web-сервисов. Подобный подход позволяет пользователям, находясь в контексте Web-сервисов, контролировать и изменять состояние доступных им ресурсов (WS-Resource).

Перечислим эти спецификации.

- WS-ResourceLifetime определяет механизм прекращения существования WS-Resource (включая обмен сообщениями), позволяющий немедленно удалить ресурс или через указанное время;
- WS-ResourceProperties определяет, как WS-Resource может быть ассоциирован с интерфейсом, описывающим Web-сервис (включая обмен сообщениями) и позволяющим извлекать, изменять и уничтожать свойства ресурса WS-Resource.

- WS-Notification определяет механизм обработки сообщений о событиях, основанный на принципах подписки/публикации.
- WS-RenewableReferences определяет механизм расширения обычной системы адресации WS-Addressing, принятой в Web-сервисах.
- WS-ServiceGroup определяет интерфейс к набору гетерогенных Web-сервисов.
- WS-BaseFaults определяет механизм обработки сообщений об ошибках.

Перечисленные спецификации находятся в процессе разработки, который неформально распределен между командами, работающими в Globus Alliance, IBM и HP, Национальной лаборатории Лоуренса и ряде университетов.

8.5.12.2 WS-Eventing — скрытый «рефакторинг» Web-сервисов?

Предложенный Microsoft и BEA стандарт WS-Eventing обеспечивает общий метод взаимодействия Web-сервисов. Объявленная цель создания WS-Eventing заключается в поддержке обмена данными о событиях по схеме подписки/публикации. Формальная модель WS-Eventing строится на основе схемы XML, спецификации WSDL и поддерживает SOAP. Кажется, на WS-Eventing «сойдутся» все основные производители, о чем свидетельствует вхождение в союз с Microsoft и BEA еще и Sun с IBM.

В августе 2004 года был опубликован документ Web Services Eventing, авторами которого являются эксперты из Microsoft, Tibco, BEA, Sun и CA. В нем описывается протокол, позволяющий Web-сервисам предоставлять подписку «на себя» и подписываться на другие сервисы. Трудно оценить технические достоинства WS-Eventing, но бросается в глаза слабая системная подготовка авторов. Складывается впечатление, что они считают достаточным установить связи между компонентами для построения систем.

Вот дословный перевод части раздела «Введение» этого документа.

«При определенных обстоятельствах Web-сервисы хотели бы получать сообщения о событиях, происходящих в других сервисах и приложениях. Эта потребность не может быть определена заранее, поэтому требуется специальный механизм для регистрации возникающего у Web-сервисов интереса и отслеживания изменений во

времени. Данная спецификация определяет протокол, согласно которому может работать этот механизм. Он определяет роли, Web-сервис (подписчик) проявляет свой интерес к другому сервису (источнику событий) в форме подписки, а значит, желает получать сообщения о событиях (извещения).

Странный для технического документа антропоморфизм. Что это за «интересы» сервисов, которые что-то «хотели бы»? Подобная терминология используется при описании модели подписки/публикации, но лишь в качестве образного представления. Если вчитаться в документ внимательнее, можно обнаружить признание в том, что чисто асинхронной модели взаимодействия в некоторых случаях недостаточно и необходимо встроить механизм, позволяющий обрабатывать сообщения о состоянии. Такое признание дорого стоит, поскольку свидетельствует, что в систему Web-сервисов нужно включить обратную связь (до сих пор стек протоколов обеспечивал исключительно однонаправленное распространение сообщений, в нем не было регулирования). Почему бы об этом не сказать честно? Зачем снова прикрываться «рефакторингом»?

Складывается впечатление, что grid-сервисы и Web-сервисы, каждые со своей стороны, подошли к одному и тому же рубежу. В обоих случаях уже набрался почти достаточный набор составляющих для построения функциональной части системы и осталось добавить лишь регулирование. Творцам архитектуры, ориентированной на сервис, нужно это осознать, и тогда может произойти немало интересного. В отношении же «коммерческих гридов» (точнее, динамических кластеров) следует отметить: это безусловно перспективное направление будет развиваться независимо от направления grid, что нисколько не умаляет его достоинств.

8.4 *Модель, ориентированная на ресурсы (ROM). Особенности кластерных и суперкомпьютерных ресурсов. Центры Обработки Данных (ЦОД).*

Компания Computer Associates обнародовала свою стратегию Managing on-demand computing, включившись в новую Большую Игру, у которой пока даже не устоялось название. Под разными терминами по существу, понимается одно и то же — а именно, новый подход к построению информационной инфраструктуры предприятий и глобальной ИТ-инфраструктуры. В соответствии с этим подходом пользователь получает те ресурсы и в том объеме, которые ему в данный момент требуются, не заботясь о том, где они находятся и как работают.

Наиболее часто употребляемая аналогия для новых инициатив — привычные всем системы электроснабжения или услуги связи. Мы не задумываемся, кто и как поставляет нам электричество и насколько сложна инфраструктура электросетей, однако, нажав на выключатель осветительного прибора или компьютера, получаем ровно то, что требуется. Но за этой удобной простотой стоит действительно сложная, четко контролируемая и в значительной степени самоуправляемая система. Точно так же, чтобы реализовать модель ИТ-услуг, предоставляемых по требованию бизнес-пользователя, необходима система управления информационной инфраструктурой, поддерживающая парадигму «вычислений по требованию».

Инфраструктура призвана создать такую операционную среду, которая отвечает потребностям текущих бизнес-процессов и способна гибко подстраиваться под динамику деловых задач. Основные технологические составляющие подобной среды — виртуализация вычислительных ресурсов и систем хранения, а также полная автоматизация таких задач, как обнаружение и предоставление информационных ресурсов для различных бизнес-процессов и в ответ на изменение их потребностей, конфигурирование системных ресурсов, выявление и коррекция проблемных ситуаций.

Ключевой для данной инфраструктуры является возможность управления изменениями в постановке задач при стабильности архитектуры. Это управление ИТ с точки зрения бизнеса. Это возможности самоуправления

информационной инфраструктуры, которые обеспечивают автоматизацию контроля за изменениями ресурсов. Это сервисный подход к предоставлению ресурсов в динамическом режиме, который опирается на консолидацию серверов и систем хранения, а также на интеграцию данных по всем видам ресурсов, включая клиентские системы, серверы, сетевое оборудование, системное программное обеспечение и приложения.

В CA любят давать своим стратегическим инициативам математическое обозначение. Так, объявленная два года назад стратегия управления электронным бизнесом укладывалась в формулу 3x6x4 — три категории решений, шесть технологических направлений, четыре программных брэнда. Нынешняя стратегия — 3x3. CA формулирует три основных принципа управления средой вычислений по требованию, которые реализуются в продуктах семейства Unicenter, сгруппированных в три области управления ИТ-инфраструктурой.

8.4.1 Три принципа управления инфраструктурой

Фундаментальными для реализации управления инфраструктурой «по требованию» в версии CA являются следующие три принципа:

- представление ИТ как услуги;
- самоуправляемая инфраструктура;
- архитектура, ориентированная на сервисы (service oriented architecture — SOA).

8.4.1.1 Представление ИТ как услуги

Инфраструктура вычислений по требованию призвана гарантировать отображение компонентов ИТ-инфраструктуры в бизнес-процессы и решение различных задач ИТ-подразделения в соответствии с установленными соглашениями об уровне обслуживания. Основной механизм, который позволяет достичь такого отображения, — это мониторинг транзакций и сетевого трафика и генерация сообщений о производительности с точки зрения приоритетов бизнеса. В CA Unicenter уже реализованы механизмы

автообнаружения систем, сетей, баз данных, приложений и других инфраструктурных компонентов и их взаимосвязи с различными бизнес-процессами. В дополнение к существующим возможностям развивается новая технология Sonar, которая позволит определять все инфраструктурные компоненты, поддерживающие работу конкретного приложения, и их взаимозависимости.

Собранная информация используется для динамического построения карты бизнес-процессов, которая будет меняться вместе с изменениями в инфраструктуре.

Управление инфраструктурой с точки зрения бизнес-процессов будет неполным без развитых аналитических возможностей. Следующий шаг в эволюции средств анализа корневых причин проблем и механизмов корреляции событий заключается в сборе обобщенных данных по работе сетей, систем, приложений, задействованных в различных бизнес-процессах. Sonar включает в себя средства анализа и диагностики трафика между бизнес-процессами. Принципиальная новизна состоит в том, что теперь средствами управляющей системы можно будет выполнять мониторинг межпроцессных взаимосвязей и влияния изменений, что очень важно для поддержки инфраструктуры вычислений по требованию. Новые сервисы управления представлением бизнес-процессов Business Process View Management Services (BPVMS) позволят выявлять причины проблем с производительностью, выполнением транзакций, работоспособностью сетей и даже безопасностью, определять их возможное влияние на деловую активность организации и выполнять необходимую корреляцию.

Технологию Sonar планируется включить в службы Common Services, которые лежат в основе архитектуры Unicenter, используются различными модулями Unicenter и тем самым обеспечивают их интегрированность. Поддержка Sonar позволит усовершенствовать средства отображения ИТ-инфраструктуры в бизнес-процессы и средства диагностики, анализа и корреляции проблем, которые уже существуют в базовой системе Unicenter Network and Systems Management (NSM), а также специальных модулях Unicenter для управления определенными приложениями (SAP, PeopleSoft, IBM/Lotus Domino, MS Exchange и т.д.).

Концепция предоставления ИТ как услуги включает в себя не только взаимное отображение бизнес-процессов и ИТ-операций. Не менее важно предоставить бизнес-пользователям средства определения и измерения информационных сервисов. Unicenter включает в себя серию решений по управлению услугами. Они, в частности, будут предоставлять каталоги услуг, с которыми могут быть соотнесены управляемые соответствующими модулями соглашения об уровне обслуживания. Средствами Unicenter Service Level Management, а также ряда новых решений семейств iCan for Unicenter и Unicenter Argis можно задать и предоставить пользователю метрики, определяющие уровни обслуживания и производительность приложений, проанализировать степень использования тех или иных ИТ-ресурсов, проконтролировать финансовые показатели, связанные с этими ресурсами, выполнить биллинг ИТ-услуг.

8.4.1.2 Самоуправляемая инфраструктура

Самоуправление в трактовке CA включает в себя интеграцию средств управления непосредственно в элементы инфраструктуры, возможности саморазвертывания, самоконфигурирования и «самолечения» (selfhealing) в случае сбоев, а также динамическое управление ресурсами.

Система управления для вычислений по требованию должна обладать возможностями автоматического развертывания и конфигурирования. В модулях Unicenter по управлению сетями и системами, управлению активами и развертыванию приложений — Unicenter NSM, Unicenter Asset Management, Unicenter Software Delivery — реализованы механизмы автоматической установки и самоконфигурирования. Так, Unicenter выявляет серверы в сети, определяет, какие задачи решаются на каждом таком сервере, развертывает и конфигурирует соответствующие модули управления, затем загружает политики, в соответствии с которыми будет осуществляться управление. Возможности автоматического восстановления работы управляющих модулей в ближайшее время будут реализованы в системах Unicenter Asset Management, Unicenter Software Delivery и Unicenter Remote Control, а в Common Services уже поддерживаются службы обеспечения высокой доступности для кластерных сред. Средствами самомониторинга и автоматического восстановления обладает решение Unicenter ServicePlus Service Desk.

Для того чтобы система управления могла автоматически принимать решения по выполнению тех или иных операций, в Unicenter технологические компоненты инфраструктуры — сетевые устройства, приложения и проч. — дополняются интерфейсами Web-служб; это позволяет полностью автоматизировать их взаимодействие со службой help desk и тем самым обеспечить оперативное реагирование на возникающие проблемы и изменения.

Для Unicenter стали традиционными возможности упреждающего мониторинга с помощью технологии нейросетей Neugents. В дополнение к ним разрабатывается технология «управления желаемым состоянием» (desired state management) ИТ-инфраструктуры. Для каждого элемента инфраструктуры определяются особенности его работы и «желаемое» состояние, описываемое некоторым набором метрик, всякое отклонение от которого система управления должна идентифицировать и автоматически корректировать. Для этого потребуется анализ корневых причин проблем и, возможно, перераспределение или выделение дополнительных ресурсов. Первым шагом к реализации такого управления станет создание распределенной интеллектуальной архитектуры управления (Distributed Intelligence Architecture, DIA), в которой специальные агенты будут проводить локальную аналитику для элементов сетевой инфраструктуры (серверов и сетевых устройств) и выполнять их мониторинг на базе политик. Такие агенты будут обладать возможностями саморазвертывания, а также автообнаружения сегментов сети и сетевых элементов. Все эти средства в совокупности — Neugents, service aware API и агенты DIA — обеспечат «самоизлечимость» компонентов ИТ-инфраструктуры. DIA планируется как единая технология для всех решений в семействе Unicenter.

Концепцию вычислений по требованию не реализовать без поддержки технологий виртуализации и динамического перераспределения ресурсов в единых пулах серверов и систем хранения. CA решает эти вопросы с помощью средств динамического управления ресурсами. В базовом модуле Unicenter NSM появилась новая возможность Dynamic Reconfiguration Option (DRO), которая в неоднородной среде из серверов разных поставщиков обеспечивает возможность оценивать уровень загрузки каждого сервера и инициирует оптимальное перераспределение ресурсов между различными задачами (такими, скажем, как обработка заказов, расчет платежных ведомостей, резервное

копирование). В Unicenter NSM появится также средство предоставления пропускной способности сети на базе заданных политик Network Performance Option.

8.4.1.3 Архитектура, ориентированная на сервисы

Архитектура управляющей системы, ориентированная на сервисы (service-oriented architecture, SOA), в трактовке CA представляет собой модульное кросс-платформнное решение, которое базируется на стандартах, включая XML, SOAP и UDDI, и поддерживает широкий спектр возможностей управления, интегрированных между собой как на функциональном уровне, так и на уровне данных.

На пользовательском уровне ориентированная на сервисы архитектура Unicenter реализует ролевой принцип доступа к функциям управления. В Unicenter Management Portal будет 50 пользовательских консолей, поддерживающих специфику определенных функциональных ролей — от бизнес-менеджера до сетевого оператора — и предоставляющих ту и только ту информацию, которая нужна данному пользователю в его работе. Эти консоли группируются в три категории в зависимости от типа пользователей системы управления.

- **Консоли для бизнес-пользователей** - средства представления информации для директоров информационной службы, финансовых директоров, руководителей отдела кадров, менеджеров других бизнес-подразделений и рядовых сотрудников, не связанных с ИТ. На таких консолях будут выводиться обобщенные данные по производительности и степени использования ИТ-ресурсов с точки зрения их влияния на бизнес-процессы: параметры SLA, стоимость ИТ-услуг, распределение и использование ресурсов, финансовые метрики наподобие ROI и TCO, данные для обоснования новых инвестиций в ИТ и т.д.

- **Консоли оперативного управления** - предназначены для сотрудников ИТ-отдела, вовлеченных в повседневную деятельность по поддержке информационной инфраструктуры. Консоли этого типа будут предоставлять исторические данные и информацию в реальном времени по элементам инфраструктуры. Функциональная интеграция управляющих модулей и прозрачный доступ к данным из разных источников - два следующих уровня SOA - обеспечат координацию действий и возможность совместной работы пользователей разных оперативных консолей, тем самым ликвидируя традиционную изоляцию

друг от друга основных процессов управления, таких как управление сетями, системное управление, контроль приложений, поддержка уровней обслуживания.

- **Консоли управления услугами** предназначены для пользователей, вовлеченных в процессы предоставления и управления ИТ-услуг: разработчиков и менеджеров ИТ-услуг, бизнес-аналитиков, операторов службы help desk и т.д. Информация на консолях этого типа связана с мониторингом производительности служб help desk/service desk, анализом соответствия производительности ИТ-инфраструктуры заданным параметрам SLA, представлением полных данных о доступности ИТ-услуги, автоматизацией поддержки потоков работ, управлением соглашениями об уровне обслуживания и другими задачами.

Такая характеристика, как интегрированность функциональных модулей, всегда была ключевой для системы управления неоднородной ИТ-средой, а в управлении вычислениями по требованию приобретает особое значение. В Unicenter контроль за ресурсами с точки зрения предоставления услуг бизнесу опирается на глубокую интеграцию между всеми решениями, входящими в состав семейства. Предполагается обеспечить взаимодействие модулей семейства не только между собой, но и с продуктами внешних поставщиков (включая конкурентов — IBM Tivoli и HP OpenView).

Для предоставления прозрачного доступа к данным по разным дисциплинам управления разработан механизм интеграции управляющих данных Management Data Integrator (MDI), который позволяет агрегировать данные из разных источников информации и формировать составные объекты в едином хранилище объектов управления Unicenter (Common Object Repositary, CORe).

Достоинства интеграции и автоматизации процессов управления лучше всего проиллюстрировать примером. Предположим, возникла проблема в пользовательском приложении на настольной системе (к примеру, в PowerPoint). Благодаря наличию в этом приложении интерфейса service aware API система Unicenter ServicePlus ServiceDesk автоматически помещает информацию по проблеме в help desk, затем ищет возможное решение в Unicenter ServicePlus Knowledge Tools. Для исправления ошибки инициируется работа модуля Unicenter Asset Management, который анализирует конфигурацию пользовательского компьютера и заказывает установку соответствующей заплаты системе Unicenter Software Delivery. Этот модуль, в свою очередь,

проводит анализ уже используемого на данной машине программного обеспечения и запрашивает у Unicenter NSM информацию о производительности и доступности системы. На основе полученных данных Unicenter Software Delivery проводит инсталляцию и оповещает Unicenter Service Level Management о соответствии сделанных изменений установленному уровню обслуживания.

8.4.2 Три области управления ИТ-инфраструктурой

В связи с появлением новой стратегии в CA предложили и новую классификацию решений, которые теперь объединены в три области управления ИТ-инфраструктурой:

- оперативное управление;
- управление ИТ-ресурсами;
- управление услугами.

8.4.2.1 Оперативное управление

Это традиционная категория дисциплин управления, связанная с повседневным мониторингом и контролем операций в ИТ-инфраструктуре.

Решения семейства Unicenter по оперативному управлению группируются по трем направлениям:

- контролю доступности и производительности (задача — обеспечить доступность ИТ-ресурсов в соответствии с бизнес-потребностями);
- управлению заданиями (автоматизировать распределение и выполнение работ в различных ИТ-доменах и между ними);
- управлению выводом (обеспечить генерацию сообщений в нужном месте в нужное время).

Мониторинг фактического потребления и производительности ИТ-ресурсов и сравнение полученных данных с параметрами, заданными в SLA, —

необходимый компонент управления средой вычислений по требованию. Интеграция центрального модуля оперативного управления Unicenter NSM с системой управления услугами Unicenter Sevice Level Management закладывает основу для реализации автономного компьютинга. Сопоставление SLA с полученной в реальном времени информацией о производительности позволяет автоматически определять, какие изменения необходимы в распределении ресурсов для бизнес-процессов. Постоянный мониторинг аппаратных платформ, операционных систем и сетевой инфраструктуры в совокупности с автоматизированным анализом корневых причин сбоев дает возможность выявлять зоны риска для SLA. Сравнение данных мониторинга в реальном времени с параметрами SLA и средства интеллектуального анализа помогают выявлять фактические и потенциальные проблемы и автоматически выполнять упреждающую коррекцию, включая переназначение неиспользуемых ресурсов на задачи, которые в них нуждаются.

Решения группы Unicenter Job Management обеспечивают планирование заданий для всех типов информационных платформ, средства визуализации потоков работ, выявление оптимальных систем для выполнения тех или иных заданий на основе анализа использования ресурсов и параметров уровня обслуживания. Третий компонент оперативного управления в Unicenter — это развитые средства предоставления информации о ходе и результатах операций в ИТ-среде: запатентованный механизм отображения зависимостей бизнес-процессов от объектов управления, новые средства визуализации данных о состоянии систем на ролевой основе и портал Unicenter Management Portal для консолидации доступа к различным процессам управления.

8.4.2.2 Управление ИТ-ресурсами

Решения Unicenter по управлению ИТ-ресурсами реализуют традиционные задачи по управлению аппаратными и программными активами и инвентаризации (Unicenter Asset Management, Unicenter Argis Portfolio Management), включая управление конфигурациями серверов и сетевой среды, развертывание операционных систем и приложений и управление лицензиями (Unicenter Software Delivery, Unicenter Software License Management), а также удаленный контроль за Windows-серверами и настольными системами

(Unicenter Remote Control). Для реализации стратегии управления вычислениями по требованию существенное значение имеет ряд усовершенствований, сделанных для этой группы систем в семействе Unicenter. Управление ресурсами расширено возможностями контроля за настольными системами, серверами, сетевыми устройствами, приложениями на протяжении всего жизненного цикла, от планирования и приобретения до развертывания, сопровождения и утилизации. Кроме того, ресурсы ИТ-инфраструктуры рассматриваются теперь в работе, а не как статические объекты, и анализируется их вклад в поддержку бизнес-процессов и реализацию ИТ-услуг. Поэтому предметом анализа являются не только начальные вложения в развертывание тех или иных ресурсов, но и общая стоимость владения, а также такие показатели, как среднее время восстановления (mean-time-to-repair) и среднее время между сбоями (mean-time-between-failure).

8.4.2.3 Управление услугами

Управление услугами связывает между собой дисциплины оперативного управления и управления ИТ-ресурсами, задачи повседневного контроля за функционированием ИТ-среды и поддержку бизнес-инвестиций в ИТ. Управление услугами позволяет представить работу ИТ-инфраструктуры как совокупность услуг для бизнеса, которые формируются, предоставляются, контролируются и оцениваются на основе данных о конфигурации, производительности и стоимости активов ИТ-среды.

Решения CA по управлению ИТ-услугами группируются по двум категориям — предоставление услуг (Service Delivery) и поддержка услуг (Service Support), что соответствует постулатам признанного стандарта управления ИТ-услугами IT Infrustructure Library (ITIL). Однако к ним добавлена еще одна категория — системы обеспечения услуг (Service Provisioning), реализующие традиционные для поставщиков услуг задачи. Решения этой группы позволяют сформировать среду, в которой создаются новые услуги, новым пользователям предоставляется доступ к существующим сервисам, и решаются задачи удаления или добавления ресурсов при реализации услуг для существующих пользователей. Исходя из этих задач, модули обеспечения услуг должны быть тесно связаны с управлением системными и сетевыми конфигурациями,

управлением жизненным циклом ИТ-ресурсов, средствами инвентаризации, развертывания приложений, а также системой управления SLA для динамического выделения и оптимизации ИТ-ресурсов в соответствии с заданными параметрами уровня обслуживания.

В предоставлении услуг (Service Delivery) центральным является процесс контроля за уровнем обслуживания, направленный на то, чтобы обеспечить не просто реализацию ИТ-услуги, но ее полное соответствие требованиям оговоренным в контракте. Система Unicenter ServiceLevel Management, используя оперативные данные по доступности и производительности ресурсов, отслеживает выполнение формальных определений SLA по отдельным бизнес-процессам и отдельным элементам ИТ-инфраструктуры, поддерживает многоуровневые SLA и соглашения с множеством взаимозависимых компонентов.

Поддержка услуг (Service Support) — это прежде всего служба help desk/service desk для оперативного реагирования на проблемы. Система Unicenter Service Plus Service Desk может быть автоматически сконфигурирована в соответствии с принципами, сформулированными в ITIL. В ее функции входит поддержка пользователей с помощью механизмов удаленного контроля, собственной базы знаний и других средств, поиск наиболее эффективных способов разрешения проблем, а также автоматическая модернизация программного обеспечения в случае необходимости. Развитие средств help desk в Unicenter пойдет по пути их объединения с центром оперативного управления, которое должно основываться на интеграции источников данных и специальной настройке представления сведений о работе информационной системы — как исторических, так и в реальном времени.

8.4.3 ЦОД в стиле SONA

8.4.3.1 Cisco практические методы реализации архитектуры, обеспечивающей оптимизацию инфраструктуру центров обработки данных

Поводом для встречи, организованной для европейских журналистов компанией Cisco Systems в Амстердаме в начале апреля, стал анонс новой стратегии построения центров обработки данных, в основу которой положена фирменная сетевая архитектура SONA. Несколько месяцев назад аналогичная акция была устроена в Америке. Формирование данной стратегии можно рассматривать как новое направление в деятельности компании, точнее, как ее реакцию на положение дел, сложившееся области центров обработки данных.

До сих пор в подавляющем большинстве случаев эти «фабрики» обработки данных строились экстенсивно, путем последовательного наращивания существующих вычислительных ресурсов, без предварительно разработанного «генерального плана». В результате типичный центр обработки данных представляет собой сложный массив гетерогенного оборудования, составленный из логически изолированных монолитных образований, где оборудование находится в жесткой связке с выполняемыми приложениями. Как известно, расплатой за любую архитектурную непредусмотрительность становится перегруженность инфраструктуры (примеры этого мы можем наблюдать в наших городах), а как следствие — неоправданные издержки. Не стали исключением и монолитные центры обработки данных; из-за их сложности — не столько логической, сколько структурной — 70% общего ИТ-бюджета уходит на поддержание существующей инфраструктуры и лишь 30% — на развитие, что не может не волновать руководителей информационных служб предприятий.

В качестве выхода из положения в Cisco предлагают осуществить переход от монолитных вычислительных центров к сервис-ориентированным архитектурам. В основе этого подхода лежат проверенные технологии виртуализации и консолидации ресурсов; как и в других применениях, они позволяют перераспределять собранные в пул ресурсы между приложениями в динамическом режиме. Но, как свидетельствует весь опыт развития компьютерных систем, для реструктуризации прежде всего должна быть построена многоуровневая модель. В данном случае строится модель интеллектуальной сети, способная предоставлять доступ к ресурсам посредством сервисов. Интеллектуальная сеть отличается от обычной сети тем,

что не просто служит средой для передачи сигналов, но объединяет вычислительные ресурсы и ресурсы хранения в своего рода каркас, или остов.

В таком случае сама сеть обеспечивает пользователю безопасный доступ к распределяемым ресурсам центров обработки данных, том числе приложениям, серверам, системам хранения и т. д. Главное достоинство этого подхода заключается в том, что он открывает возможность для разработки сбалансированного сетевого проекта, оптимизированного по готовности, по производительности приложений, по другим параметрам и в то же время допускающего модернизацию, которая может быть вызвана изменением рыночных условий, приоритетов в бизнесе, появлением новых технологий и т. д.

Характеризуя новый взгляд на вещи, генеральный директор Cisco Джон Чамберс сказал: «Сеть становится платформой, предоставляющей приложения пользователям».

Простейший пример такого подхода: сеть может самостоятельно проверять потоки передаваемых по ней данных на наличие вирусов, и тогда подключенные к ней компьютеры могут быть освобождены от этой заботы.

В качестве технологического базиса для интеллектуальных сетей в Cisco предлагают архитектуру, получившую наименование SONA (Service Oriented Network Architecture — «сервис-ориентированная сетевая архитектура»). Название SONA вызывает очевидные аллюзии с сервис-ориентированной архитектурой SOA (Service Oriented Architecture). Но в отличие от уже привычной SOA, где сервисы служат средством обмена между приложениями, SONA обеспечивает выполнение сетевых прикладных сервисов. Проводя во время встречи с журналистами аналогию между SONA и SOA, специалисты Cisco подчеркивали, что было бы неверно представлять SOA только как программную архитектуру: на тех же принципах может быть организовано и аппаратное обеспечение. Технологии, лежащие в основе SONA, разделяются на две составляющие. Одна из них вполне традиционна для Cisco — оптимизация доставки данных по сети, то есть ее ускорение, компрессия, балансировка нагрузки и все остальное, что требуется предоставить приложениям для более производительной работы. Вторая часть, получившая

название Application Oriented Networking, была впервые представлена компанией в июне 2005 года; теперь аббревиатура AON стала «фирменной». Технология AON наделяет сеть способностью «читать и интерпретировать» (to read and interpret) передаваемые сообщения, она может выполнять функции «универсального системного переводчика». Основное достоинство SONA состоит в том, что, опираясь на нее, удастся создавать такие центры обработки данных, которые не потребуют существенных затрат на администрирование и управление.

Как утверждают в компании, SONA становится одним из важнейших технологических направлений в деятельности Cisco.

8.4.3.2 Это тоже AON

Application-Oriented Networking — подход к построению сети, ориентированный в большей степени не на передачу данных, а на организацию взаимодействия между приложениями. Данная технология возникла в результате развития обмена XML-сообщениями для связи между приложениями, источниками данных и компьютерных ресурсов. Целый ряд операций, требующих посредничества между участниками процесса обмена данными или мониторинга этими процессами, могут быть перенесены в сетевые устройства, специально предназначенные для этой цели. Правила и политики для выполнения эти операций могут быть также выражены на XML и загружены в эти устройства по мере необходимости. Перенос части функции программного обеспечения промежуточного слоя в сеть может привести к ускорению работы.

По поводу того, как сложится соотношения ролей AON и программного инструментария промежуточного слоя в будущем, единства мнений пока нет. Ряд экспертов весьма оптимистичны в оценках перспективы AON; между тем Евгений Кузнецов, один из пионеров данного направления и основатель компании DataPower, недавно купленной корпорацией IBM, проявляет большее здравомыслие. Он считает, что «тяжелые» приложения, где велика логика процессов, в любом случае потребуют полный стек средств промежуточного слоя наподобие J2EE, точно так же, как в языках программирования, где есть место XML, но и Кобол продолжает жить.

8.4.4 Три уровня SONA

Специалисты Cisco Systems выделяют в модели SONA три уровня.

- **Уровень сетевой инфраструктуры (networked infrastructure layer)** объединяет все сетевые ресурсы. На уровне сетевой инфраструктуры архитектура SONA предоставляет рекомендации для того, как построить сеть в виде полностью интегрированной системы.

- **Уровень интерактивных сервисов (interactive services layer)** обеспечивает эффективное резервирование ресурсов, доставляемых средствами сетевой инфраструктуры. Для этого предоставляется полный набор сервисов, придающих сети качество, названное интеллектуальностью, сервисы позволяют сделать взаимоотношения бизнеса и приложений более предсказуемыми и надежными.

- **Прикладной уровень (applications layer)** объединяет взаимодействующие между собой приложения таким образом, что исключается необходимость в инсталляции приложений и обеспечивается возможность модернизации приложений, не прерывая их работы, сохраняя безопасность.

8.4.5 Стратегия N1: управление большими системами

Рано или поздно это должно было случиться — информационные системы выросли до таких размеров, что администраторы уже не справляются с управлением десятками серверов, сотнями гигабайт данных и тысячами сетевых портов. И если совсем недавно еще можно было найти «гуру», которые были бы в состоянии удержать в голове все конфигурации, то сейчас стало понятно, что таких высококлассных специалистов на всех не хватит, да и не всем пользователям хочется попасть в зависимость от гуру. Начались поиски более «индустриального» решения по управлению крупными системами, к которым, в частности, относится стратегия N1.

Обычное явление в наших информационных системах — «зоопарк». Иногда об этом сообщают с сожалением — «ну что же тут поделаешь, раз так сложилось». Иногда — с гордостью: «мы что угодно и с чем угодно умеем соединить».

Причина возникновения такого зоопарка, как правило проста — бизнес требует применения тех или иных прикладных систем, которые тянут за собой конкретные решения. Например, для системы управления персоналом требовался дисковый массив EMC, подключенных к серверам Sun; банковская система, которая внедрялась другим интегратором, работает на серверах HP с массивами от Compaq, а документооборот на базе Lotus Notes потребовал оборудования IBM. Когда пришла мода на intranet, то к одной системе подключили Web-сервер на MS IIS, к другой — на Linux, а к третьей — на Solaris. Кроме того, разные части сети строились в разное время, где-то на Cisco, где-то на Nortel, где-то на 3Com.

И вот все это досталось в наследство начальнику отдела эксплуатации информационных систем, от которого потребовали обеспечить бесперебойную работу, быстрое внедрение очередных приложений и производительность «как было обещано в тестах, продемонстрированных поставщиками».

Получив такое задание системный администратор получает сильнейшую головную боль и ему остается только мечтать о получении адекватного инструмента управления зоопарком: «Что, если бы можно было все виртуализировать и управлять не коробками, а ресурсами? Еще лучше было бы автоматизировать все настолько, чтобы требовалось только назначать каждому приложению необходимый объем ресурсов, а система сама бы их находила и распределяла.»

8.4.5.1 N1 — стратегия

Впервые компания Sun Microsystems заявила о стратегии N1 осенью 2002 года. Тогда это вызвало бурные дискуссии, главным вопросом которых было: «Это очередной маркетинговый ход или реальная стратегия?» Компания нарисовала привлекательную картину построения и эксплуатации вычислительных центров, на основе технологий виртуализации ресурсов, управлении системными и прикладными сервисами и автоматизации самого процесса управления путем применения политик. Спустя год сформировалась вполне реальная стратегия и тактика внедрения, подкрепленная набором конкретных продуктов.

Стратегия развития N1 состоит из трех этапов:

- управление инфраструктурой;
- управление прикладными сервисами;
- автоматизация управления.

8.4.5.2 Управление инфраструктурой

Управление инфраструктурой основывается на концепции виртуализации системных ресурсов и создания «пулов» ресурсов. Известно, что массу времени отнимают рутинные процедуры подключения, перекоммутации, конфигурирования, запуска оборудования и системных сервисов. Каждый администратор стремится минимизировать эти действия путем автоматизации, стандартизации, унификации платформ но не всегда это удается. Разработчики N1 поставили перед собой задачу сделать так, чтобы все оборудование физически устанавливалось и подключалось лишь однажды, а все дальнейшие операции по изменению архитектуры, перекоммутации и переконфигурированию проводились программно.

Научившись программно управлять параметрами оборудования и архитектурой системы, можно поставить перед собой следующую задачу — создание промежуточного системного слоя, позволяющего перейти к виртуальному представлению ресурсов путем сведения всего многообразия конкретных ресурсов к обобщенным. В таком случае их можно было характеризовать унифицированными измеримыми параметрами, позволяющими перейти от конкретных «Pentium/4, 2 Ггц» или «UltraSCSI-III, 73 Гбайт» к обобщенным единицам измерения вычислительной мощности, емкости и производительности дисковых подсистем. Разумеется, не все можно унифицировать. Например, очевидно, что имеются существенные различия в использовании ресурсов между вычислительными узлами с вертикальной масштабируемостью (большие SMP машины) и горизонтальной масштабируемостью (одно-двух процессорные серверы и серверы-»лезвия») — это будут различные типы ресурсов.

Когда архитектура построена, ресурсы измерены и сведены к универсальным единицам, то для их полнофункционального управления необходимо наладить

обратную связь — получение информации об используемых ресурсах. Наличие такой информации необходимо как для ручного управления размещением приложений на имеющихся ресурсах, так и для динамической оптимизации производительности. Динамическая оптимизация позволит производить перераспределение ресурсов в случае пиковых загрузок или в соответствии с заданными политиками эксплуатации. Кроме того, мониторинг загрузки ресурсов будет использоваться и для биллинга, когда вычислительный центр начнет предоставлять услуги своим пользователям на основе SLA.

8.4.5.3 Управление прикладными сервисами

Система с виртуализированными ресурсами готова к тому, чтобы разместить на ней прикладные сервисы — современные архитектуры прикладных систем имеют достаточно выраженную модульную, многоуровневую структуру, в которой присутствуют следующие уровни (рис. 8.9):

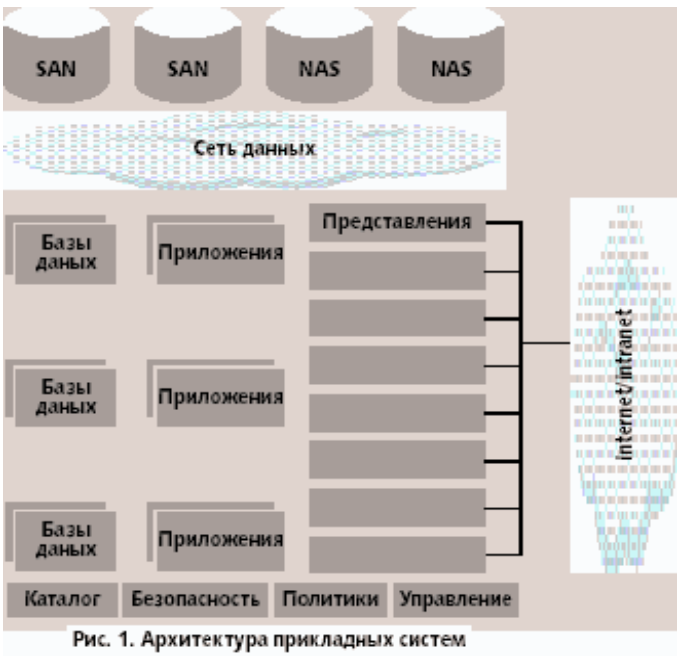


Рис. 1. Архитектура прикладных систем

- Уровень доступа пользователей и устройств — шлюз, через который проходят все запросы к прикладным сервисам и выполняющий функции идентификации, аутентификации и авторизации пользователей и устройств в соответствии с политиками безопасности.
- Уровень представления — уровень Web-серверов и порталов, на котором формируется универсальный пользовательский интерфейс к информации и приложениям. Здесь происходит консолидация и индивидуализация пользовательского интерфейса, поддержка сеансов, поддержка динамического информационного наполнения.
- Уровень приложений — выполнение бизнес-логики прикладных сервисов. Важно, что этот уровень отделен и от уровня представления и от уровня доступа к данным. Только такое решение позволит обеспечить требуемую гибкость управления.
- Уровень доступа к данным — предоставление стандартных интерфейсов доступа к консолидированному хранилищу данных. Здесь используется стандартный доступ к структурированным данным (СУБД) и к неструктурированным данным (файловые системы).
- Сеть хранения данных — разделяемый доступ к ресурсам систем хранения данных и гибкость конфигурирования, что в сочетании со средствами виртуализации, поставляемыми производителями дисковых массивов, дает возможность сделать единое хранилище данных для всех уровней системной архитектуры.
- Инфраструктура управления и системных политик — набор служб, пронизывающий все уровни архитектуры и обеспечивающий выполнение заданных политик безопасности, администрирования, управления ресурсами.

Разложив приложения по таким составляющим, администратор может передать информацию о них в N1, чтобы разместить эти составляющие по существующим виртуализированным ресурсам.

Предположим, что системный администратор, имея пулы ресурсов, хочет разместить на них некоторое количество приложений. Тогда, взяв Приложение 1, он анализирует его компоненты и описывает требующиеся ему ресурсы и политики: Приложение 1 требует: M единиц вычислительных ресурсов с вертикальной масштабируемостью (как правило для СУБД или аналитики); N единиц вычислительных ресурсов с горизонтальной масштабируемостью (для уровня приложений или уровня доступа — Web-серверы и порталы); S Гбайт

дискового пространства с уровнем производительности L и уровнем защищенности (готовности) P; K единиц сетевой пропускной способности. Общий уровень готовности приложения должен составлять B.

Сформулировав эти требования, системный администратор передает их «шине управления N1». Шина анализирует доступные ресурсы и резервирует их в соответствии с заданными политиками и требованиями. После определения необходимых аппаратных ресурсов шина производит размещение компонентов Приложения 1 на зарезервированном оборудовании и задает режимы их использования. Так, например, будут заданы предельно допустимые уровни загрузки процессорной мощности, определены режимы и приоритеты перехода на резервные мощности в случае сбоя.

Идеальным случаем является набор приложений, разработанных для стандартной платформы J2EE, для которой вопросы зависимости от ОС и аппаратной платформы решаются на уровне промежуточного слоя. Однако в реальной жизни встречаются самые различные приложения, и эти варианты тоже должны быть учтены при разработке протоколов управления N1.

8.4.5.4 Автоматизация управления

На следующем этапе администратор задает политики и правила для автоматизированного управления прикладными сервисами. Одна из задач — перераспределение ресурсов в соответствии с требованиями приложений. Например, если сервер баз данных не справляется с возросшей нагрузкой, то система перемещает его на более мощный сервер или расширяет домен, на котором работает СУБД, путем добавления системных плат.

Еще одна не менее важная задача — автоматическое поддержание требуемого уровня готовности прикладных сервисов. Вся система, построенная на технологиях N1, пронизана кластерными связями, обеспечивающими резервирование ресурсов и процедуры запуска приложений в случае сбоя. В идеале, каждое приложение выполняется в кластерной конфигурации с несколькими узлами, которая гарантирует непрерывность функционирования при любых условиях.

8.4.5.5 N1 — продукты и решения

На сегодняшний день в N1 реализованы средства управления инфраструктурой, включающие: средства виртуализации и управления системами хранения данных, средства управления ресурсами больших SMP-серверов, средства управления группировками серверов. Совсем недавно появился продукт, позволяющий говорить о начале перехода ко второй фазе реализации — управлению прикладными сервисами.

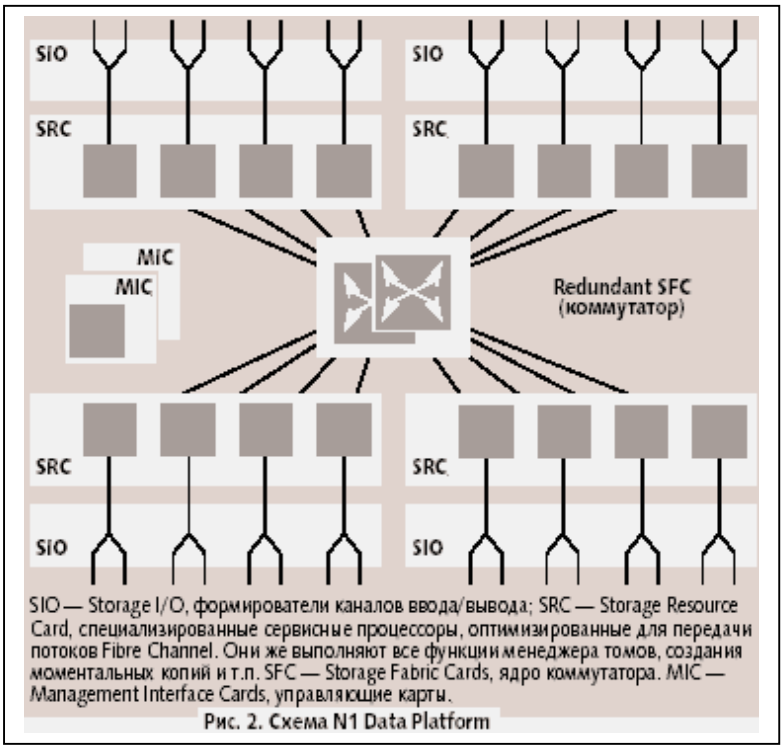
8.4.5.5.1 Системы хранения данных

Еще недавно системный администратор для того, чтобы работать с дисками, был вынужден пользоваться заклинаниями типа «newfs -b 4096 /dev/rdsk/c0t3d0s6» (и так для каждой новой файловой системы на каждом разделе диска). Рост объемов данных и необходимость упрощения администрирования привели к появлению менеджеров томов, позволяющих лишь однажды вспомнить о физических дисках и разделах при создании логического тома и впоследствии работать уже на более высоком уровне абстракции. Следующим шагом к упрощению жизни администратора стало появление технологий виртуализации в дисковых массивах, которая позволила избавиться даже от этого первого шага.

Для того, чтобы получить настоящую виртуализацию систем хранения, не хватает одного — централизованного управления. Для управления гетерогенной сетью хранения данных имеется продукт N1 Data Platform, который позволяет сформировать «промежуточный слой» между сетью серверов и сетью данных. Этот продукт представляет собой коммутатор потоков данных между серверной частью вычислительного центра и сетью хранения данных. Коммутатор имеет до 32 портов протокола Fibre Channel и до 16 процессоров данных. Помимо коммутации система выполняет функции менеджера томов, разгружая от этой задачи процессоры подключенных хостов. Таким образом, мы получаем консолидированный менеджер томов, позволяющий объединить все ресурсы хранения в единый виртуализированный пул.

N1 Data Platform также выполняет функции управления ресурсами хранения, обеспечивая учет использования и управление производительностью за счет организации томов (задание уровней RAID и т.п.). В устройстве предприняты специальные меры по обеспечению защиты данных и изоляции друг от друга томов, используемых различными приложениями. Устройство имеет встроенную функцию создания моментальных копий в сети хранения данных, что также помогает разгружать центральные процессоры серверов.

Внутренняя архитектура N1 Data Platform приведена на рис. 8.10. Видно, что по существу система представляет собой коммутатор сети хранения данных, насыщенный интеллектуальными функциями, которые позволяют возложить на него значительную часть обязанностей по управлению сетью хранения данных.



SIO — Storage I/O, формирователи каналов ввода/вывода; SRC — Storage Resource Card, специализированные сервисные процессоры, оптимизированные для передачи потоков Fibre Channel. Они же выполняют все функции менеджера томов, создания моментальных копий и т.п. SFC — Storage Fabric Cards, ядро коммутатора. MIC — Management Interface Cards, управляющие карты.

Рис. 2. Схема N1 Data Platform

Рис. 8.10

8.4.5.5.2 Управление SMP-серверами

Разработчики из Sun Microsystems — активные приверженцы архитектуры симметричной многопроцессорной обработки, которые компания выпускает с 1990 года. В середине 90-х годов высказывались серьезные сомнения в целесообразности развития этой архитектуры — многие аналитики и производители вынесли приговор «не больше 10-12 процессоров в SMP-системе, после этого наступает деградация». Однако в Sun продолжается разработка и внедрение 64-процессорных серверов и совершенствование ОС Solaris для лучшей масштабируемости, что, в результате, позволило сделать первые шаги в сторону лучшей управляемости ресурсами, заложившими основу N1.

Динамическая реконфигурация компонентов — возможность добавлять или удалять системные платы (процессор/память) или платы ввода/вывода в работающей системе. При нехватке ресурсов в сервере баз данных (например, в период закрытия квартала) их можно позаимствовать в соседнем сервере. По окончании отчетного периода их можно вернуть на место.

Динамические системные домены — технология динамической реконфигурации сделала возможным создание динамических системных доменов, когда один большой сервер разбивается на несколько серверов поменьше, полностью изолированных друг от друга. Каждый сервер имеет свою конфигурацию, операционную систему, набор приложений — сбой в одном домене никак не влияет на работу других.

Контейнеры Solaris — средство управления вычислительными ресурсами внутри домена. Системный администратор может выделять определенные доли ресурсов отдельным приложениям, пользователям или группам. Начиная с девятой версии Solaris эти возможности включены в базовую поставку операционной системы.

Одно из важных направлений развития перечисленных технологий — унификация и упрощение администрирования с тем, чтобы обеспечить возможности автоматического перераспределения ресурсов в соответствии с заданными политиками. Сегодня уже доступны возможности автоматического

изменения конфигураций доменов с помощью скриптов (например, перевод OLTP-ресурсов по окончании рабочего дня в домен, занятый выполнением пакетных аналитических запросов).

8.4.5.5.3 Управление серверными группировками

Работа с большим количеством сетевых Unix-систем в Sun была налажена давно — уместно вспомнить, что компания начинала с разработок сетевых рабочих станций, которые поставлялись и запускались десятками и сотнями. Для таких массовых установок и были разработаны технологии JumpStart и Solaris Flash, позволяющие производить массовую установку и конфигурирование рабочих станций. С появлением blade-серверов ситуация стала критической — теперь только в одну стандартную стойку можно поместить до 160 самостоятельных серверов. Очевидно, что без специальных средств управления такими серверными фермами их эксплуатация не представляется возможной. Именно поэтому сейчас эти системы поставляются в комплекте с программным обеспечением N1 Provisioning Server.

Этот пакет позволяет системному администратору проектировать серверную ферму методами, очень похожими на рисование диаграмм в продукте Visio. Администратор проектирует архитектуру комплекса, подбирает необходимые компоненты (Web-серверы, межсетевые экраны, балансировщики загрузки, почтовые серверы) и соединяет их между собой. Отличие состоит в том, что N1 Provisioning Server обладает возможностью анализировать имеющееся в наличии оборудование и производить программную перекоммутацию, установку и конфигурирование системного программного обеспечения. В результате получается то, что можно было бы назвать мечтой администратора, который только чертит проект, а всю рутинную работу выполняет Provisioning Server.

В схему могут вноситься изменения: добавление серверов, изменение конфигураций, замена моделей и т.п. В этом случае администратор просто загружает сохраненную схему комплекса, производит требуемые изменения и дает команду на переконфигурирование.

8.4.6 N1 Provisioning Server изнутри

Архитектура системы на базе N1 Provisioning Server включает несколько уровней

- Уровень управления
- Уровень связей (для логических комплексов)
- Уровень ресурсов (для серверов с виртуализацией и SAN)

Уровень ресурсов включает в себя установленное и готовое к использованию серверное оборудование и системы хранения данных.

Уровень связей — сетевая инфраструктура (как локальная сеть, так и сеть хранения), позволяющая организовывать виртуальные сети и имеющая возможности программной коммутации.

Уровень управления отвечает за создание логических схем серверных комплексов и включает визуальное средство проектирования и языки логического описания создаваемых архитектур: Farm Markup Language (FML), Monitoring Markup Language (MML) и Wiring Markup Language (WML), основанные на XML. Ядром этого уровеня (и всего пакета N1 Provisioning Server) является программный инструментарий Infrastructure Director, обеспечивающее управление использованием ресурсов, виртуализацию и мониторинг, систему безопасности и верификацию схем соединений.

Развитием этой подсистемы будет ее интеграция со средствами управления ресурсами больших SMP-серверов и создание N1 Datacenter Provisioning Server.

8.4.7 Управление прикладными сервисами

В 2005 году в арсенале программных средств, входящих в состав N1, появилось средство управления прикладными сервисами N1 CenterRun.

Автоматизация достигла больших успехов в области проектирования и разработки — широкий спектр средств систем визуального проектирования с последующей генерацией кода позволяет значительно сократить время разработки сетевых сервисов. Существуют развитые средства групповой разработки, контроля версий и документирования. Немаловажно и то, что существует также набор инструментов для автоматизации тестирования программного обеспечения. Однако когда дело доходит до ввода разработок в промышленную эксплуатацию, приходится возвращаться к ручной работе. Разнообразные установочные скрипты и программы, системы управления пакетами, дистрибутивы разных форматов, системы контроля заплат «ложатся» на плечи администратора без всякой надежды на возможность стандартизации и автоматизации. И если с одним или несколькими серверами эта работа еще может быть выполнена вручную, то при переходе к эксплуатации десятков и сотен серверов задача становится нереальной без средств автоматического управления программными пакетами на большом количестве серверов. Прибавьте к этому резко возрастающую вероятность ошибок, проблемы с безопасностью и правами доступа, невозможность учета и аудита установленных пакетов.

N1 CenterRun нацелен на выполнение задач именно такого масштаба. Главная цель — на индустриальном уровне обеспечить унифицированный контроль за установкой, запуском, изменением программных систем. Важно отметить, что продукт специально разрабатывался в расчете на эксплуатацию в системах с десятками и сотнями серверов и позволяет автоматизировать масштабные административные работы.

Первым этапом работы с пакетом является сбор информации о приложениях (рисунок). Администратор указывает сервер, на котором размещена эталонная версия приложения, и параметры этого приложения. Следует отметить, что приложения, с которыми работает N1 CenterRun, должны быть построены по объектной модели (J2EE, COM/COM+, .NET) и работать в рамках одного из существующих серверов приложений (Sun Java System, BEA WebLogic, IBM WebSphere, Microsoft COM+).

На следующем шаге система производит автоматическое размещение приложений. Администратор задает серверы, на которых должны размещаться

компоненты, а система производит анализ зависимостей, при необходимости делает пробное размещение, и после этого размещает компоненты приложения по заданным узлам сети.

В процессе эксплуатации система поддерживает контроль версий и заплат, позволяет производить откат к предыдущей версии, если после перехода на новую обнаруживаются проблемы и многие другие функции жизненного цикла программных систем.

Интересной особенностью является возможность периодического сравнения установленных систем между собой или с эталонной установкой. Такая функция позволяет выявлять изменения, сделанные администратором помимо централизованной системы, и либо вносить их в единую базу данных, либо возвращать установку к эталонной модели.

Информационная безопасность в N1 CenterRun обеспечивается системой пользователей и групп с различными ролями и правами. Такая организация позволяет избежать выполнения всех действий от имени суперпользователя, которому все позволено.

Информация о приложениях хранится в виде метаданных в формате XML, что позволяет администратору расширять систему за счет добавления своих типов метаданных, скриптов и вызовов Java-процедур.

На сегодняшний день система поддерживает платформы Solaris 8 и 9, Windows 2003, RedHat 7.2, 7.3, Advanced Server 2.1 и AIX 4.3.3, 5.1, 5.2. n.

8.5 *Новая технология интеграции ресурсов на базе виртуальных «Облаков»*

«Облака» объединяют под один зонтик множество областей ИТ, которые раньше, имея много общего, разделялись. Из «облаков» подразумевается получать платформы для вычислений (серверы и виртуальные машины) или приложений (MS Azure), а также сами приложения, доставляемые в рамках концепции SaaS (Salesforce.com и сервисы от Google, Facebook или Yandex). Работа «облака» обеспечивается неким комплексом аппаратных и программных средств – назовем его операционной системой «облака» (ОСО), поддерживающей работу с клиентом, опираясь на сеть крупных центров хранения и обработки данных. Как устроена ОСО, где расположены центры, на каком оборудовании и под управлением какого ПО они функционируют – все эти вопросы для пользователя представляют интерес только из любопытства, а сама структура «облака» для него не важна.

Действительно, с какой целью обычно используются компьютеры? Для работы с данными и приложениями. Однако для решения таких достаточно простых задач компаниям приходится приобретать ИТ-инфраструктуру и нанимать обслуживающий персонал. «Облака» позволят сделать это ненужным. По этой логике именно хостинг-провайдеры по сути стали первыми поставщиками «облачных» сервисов.

Сегодня бизнес хостинг-провайдеров значительно расширяется за счет охвата корпоративного сектора, начинающего предъявлять спрос на «облачные» сервисы, самыми распространенными из которых являются SaaS и внутренние «облака». Компании переходят на потребление программного обеспечения по модели SaaS, что представляет собой потребление уже бизнес-, а не ИТ-услуг, как раньше. Методы хостинга начинают применяться не только в рамках ауторсинга, но и для организации внутрикорпоративных ИТ-систем, превращающихся во внутренние «облака». Появление спроса привлекло внимание крупных ИТ-игроков, которые стали теперь позиционировать себя в роли хостинг-провайдеров «облачных» сервисов.

Можно предложить следующую классификацию поставщиков «облаков» (см. рис.), в которой выделяются две большие группы: поставщики платформных «облаков» (ИТ-компании, предлагающие хостинг на базе собственного ПО) и поставщики «облаков» услуг, использующие для создания сервисов ПО других компаний. Первая группа поставщиков делится на три подгруппы: Google, Microsoft и другие крупные компании (IBM, Apple, а также Yahoo!, EMC, HP/EDS, Amazon, Facebook, Adobe и т.д.). Ко второй группе поставщиков «облачных» вычислений можно отнести независимых сервис-провайдеров, предоставляющих услуги внешним клиентам, а также внутрикорпоративных провайдеров, обслуживающих филиалы, отделы и дочерние подразделения, а также сотрудников и партнеров.

Рис. 8.11 Типы Облаков в ИТ-технологиях

Между платформными и сервисными поставщиками «облачных» вычислений обостряется конкуренция – несмотря на большой потенциал ИТ-гигантов, услуги хостеров скоро будут востребованы и в контексте «облачных» вычислений. Этому способствует нынешняя рецессия – средний и малый бизнес в поисках снижения издержек на ИТ переходит к использованию хостинга вычислительных ресурсов и при этом ищет наиболее дешевые предложения. При таком сценарии развития рынка стоимость услуг станет не единственным преимуществом независимых сервис-провайдеров – важно то, что хостеры будут проявлять более высокую гибкость и оперативность в работе с клиентами. Ведь задача заключается не столько в предоставлении оптимальной вычислительной среды потребителям «облаков» при помощи виртуализации, средств балансировки нагрузки – всего того, что предлагают сегодня представители первых трех групп провайдеров облачных

услуг, сколько в автоматизации бизнес-процессов клиентов: управление планами счетов, биллингом и т.п.

Постепенно уходят в прошлое и другие факторы, тормозящие развитие «облачного сервиса». Так, наряду с весьма существенным повышением надежности сетей сегодня уменьшаются и опасения компаний, связанные с потенциальной возможностью контроля провайдером услуг «облака» приватных данных предприятий.

Рост «облачных» сервисов генерируется, в том числе в денежном выражении, за счет активности компаний малого и среднего бизнеса, которые заинтересованы в фокусировании внимания на ключевых бизнес-процессах и потому передают поддержку своих ИТ третьим лицам. Именно этот корпоративный эшелон начинает приобретать ПО как услугу, оценивая преимущества оплаты за время пользования, а в некоторых случаях в кредит. Более того, по оценкам аналитиков, компании, выходящие сейчас на рынок, в большинстве случаев даже не планируют обзаводиться собственной ИТ-инфраструктурой, сразу приобретая ее по модели аутсорсинга или, в новой терминологии, переходя к «облачным» вычислениям. Такие компании преимущественно нуждаются в приложениях для совместной работы и обмена сообщениями, в файловых сервисах, а также в офисных приложениях, бухгалтерских системах, программах безопасности – все это уже сейчас можно потреблять из «облаков». И выиграет в конкурентной гонке тот провайдер «облачных» сервисов, которому удастся предложить все наиболее востребованные программы и ИТ-услуги из «одних рук», на своей собственной, но предельно отрытой для интеграции любых внешних сервисов платформе. Сегодня на рынке пока еще нет поставщика, который консолидировал бы все востребованные компаниями малого и среднего бизнеса ИТ-сервисы и предлагал бы их «пакетами».

Среди крупного бизнеса наиболее распространенным сценарием приобщения к «облакам» является создание инфраструктуры виртуальных компьютеров, в которой рабочие места пользователей превращаются в средства доступа, а вся программная «начинка» хранится в «облаке». Для доступа к своему рабочему месту пользователю нужен лишь канал в Internet, а все необходимое он может взять из корпоративного «облака». Однако сегодня еще далеко не все

приложения предоставляются с помощью тонкого клиента: они не поддерживаются используемыми технологиями, требуют административного доступа, не совместимы с другими приложениями, нуждаются в тщательной изоляции, не обеспечивают возможности полного контроля рабочей среды. Все эти проблемы решаемы, и для продвижения инфраструктуры виртуальных компьютеров в «облака» нет объективных преград.

Практически все ведущие производители заявляют сегодня о готовности помочь компаниям и пользователям в создании своих собственных «облаков», например Microsoft анонсировала «облачную» платформу Azure Services Platform, VMware позиционирует vSphere как первую ОС для «облачных» вычислений, и т.д. Однако, если разобраться, во всех случаях рабочей модели «облака» пока нет, а происходит перепозиционирование имеющегося набора продуктов с целью «застолбить» место на перспективном направлении и, как обычно, заранее по максимуму «привязать» потребителя к своей проприетарной платформе.

Все необходимые составляющие собственного «облака» есть у Microsoft – Windows Azure включает инструменты для разработки сервисов и сайтов, центр обработки данных, исполняющий код созданного решения, масштабируемое хранилище, эмуляцию, позволяющую полноценно отлаживать сервис на локальной машине, и др. VMware перепозиционировала свое решение VMware Virtual Infrastructure под иную парадигму. Система vSphere 4, в силу исторических особенностей работы компании VMware, будет работать в «облаках» (собственных виртуальных ЦОД) корпоративных заказчиков, создавая частные группы виртуальных машин на базе физических серверов, взятых из пула ресурсов поставщиков услуг «облачных» вычислений. В реальности все эти платформы еще только начинают тестироваться, и незрелость подобных технологических предложений пока ограничивает полноценный переход ИТ в «облака» как для предприятий, так и даже для домашних пользователей. Но вместе с тем много разных задач для разных видов бизнеса и конечных пользователей уже сейчас эффективно решаются с помощью «облачных» вычислений – кто-то на этом зарабатывает, а кто-то существенно экономит, а это гарантирует, что рано или поздно все окажутся в «облаках».

Для обозначения компьютера, собранного из компьютеров, стали использовать слово «облако». Сначала выдвигалась утопическая идея, будто весь мир будет накрыт несколькими гигантскими облаками, как тут было не вспомнить высказывание Томаса Ватсона, легендарного председателя совета директоров корпорации IBM, сделанное в 40-е годы, относительно того, что на весь мир хватит дюжины компьютеров. Но очень скоро стало ясно, что малым числом облаков не обойтись и облака могут быть разными по своим размерам.

8.5.1 Исторические последствия

За сменой технологической парадигмы следует еще одна. С середины сороковых годов и до середины первого десятилетия XXI века внимание было сосредоточено на программном и аппаратном обеспечении, позволяющем получать желаемые результаты, то есть в большей мере на инструментах, а не на том, для чего они предназначены. Скажем, если сравнивать с музыкой, то больше внимания уделялось тому, на чем ее исполнять, как записывать мелодии, на чем хранить записи, а не музыке как таковой. Изменения, связанные с распространением сервисной модели от первых скромных шагов SOA до EaaS (Everything as a Service), привели к тому, что теперь важны потребительские качества сервисов, а не то, где и как они выполняются.

На первое место сегодня вышли вопросы экономичности, безопасности и надежности сервисов, а их решение ищется в централизации компьютинга, сначала на базе все более виртуализированных ЦОД, а затем в облачном компьютинге в его разных формах. Из централизации непосредственным образом вытекает виртуализация клиентских рабочих мест, то есть абстрагирование собственно рабочего места от прибора, на котором оно реализовано, и трансформация приложений в виртуальные устройства (virtual appliance), способные доставлять приложения и инфраструктурное ПО к местам их использования. Эти явления не существуют сами по себе, а взаимодополняют друг друга (рис.8.11). По определению экономиста Пола Рамера, совместно они образуют «одновременное совершенствование дополнительных благ», так он назвал компоненты, суммарный эффект работы которых выше, чем каждого по отдельности.

Таким образом завершился виток эволюционной спирали, начавшийся с централизованной модели вычислений и ею же завершившийся: централизованная модель с мэйнфреймами и тупыми зелеными терминалами, мини-ЭВМ, распределенная модель и снова центры обработки данных. Однако централизованная модель с тонкими клиентами не исчезала вообще, она продолжала существовать, хотя и в ограниченных масштабах, и вот сейчас происходит возврат к ней, но на принципиально ином уровне – вычисления и хранение данных уходят в облака, а в руках пользователя остаются виртуальные устройства доступа. Как именно будет выглядеть новый централизованный компьютерный мир, пока сказать сложно, но уже сейчас ясно, что картина будет совершенно иной – материальные формы компьютинга будут глубоко скрыты и станут невидимы для пользователя, а на поверхности останутся только сервисы. Обидно, но, по всей видимости, в будущем пополнять экспозиции музеев больше станет нечем, маловероятно, что однообразные стойки, набитые однообразным оборудованием, смогут вызвать интерес у посетителей. Но на все нужно время, известно, что период от зарождения новой компьютерной технологии до ее серьезного выхода на рынок занимает лет восемь-десять, сейчас прошел примерно такой срок, и наметившиеся изменения принимают массовый характер.

8.5.2 Приватизация облаков

Как бы ни ругали централизованную модель, особенно лет десять-пятнадцать назад, когда ее противники предрекали смерть мэйнфреймам, она не исчезала и продолжила свое нишевое существование, поскольку обладает рядом превосходных качеств. Обеспечивая более эффективное, чем децентрализованная, использование ресурсов, высокую степень безопасности и надежности, удобство с точки зрения управления, в итоге она является носителем еще одного свойства, которому можно дать название «доверительность» (trust), поэтому не случайно, что самые ответственные приложения по-прежнему реализуют на мэйнфреймах. Осознание преимуществ централизованной модели подтолкнуло к идее создания центров обработки данных, и поначалу все казалось замечательно и весьма просто – есть дешевые

серверы, есть сети, достаточно собрать все вместе под одной крышей. Но тривиальная сборка ЦОД из стандартных серверов отличается чрезвычайно низкой экономической и энергетической эффективностью, особенно если принять во внимание паровозный коэффициент полезного действия серверов на платформе x86. Оказалось, что такими ЦОД, иногда их еще называют серверными фермами, «в лоб» заменить мэйнфреймы невозможно.

Выход был найден в создании масштабируемых облачных инфраструктур. Облако, в отличие от ЦОД, мэйнфреймов или мощных Unix-серверов (их еще называли мидфреймами) со встроенными – как аппаратными, так и программными – механизмами виртуализации, является динамической структурой. Все ресурсы облака могут использоваться с большей эффективностью, здесь не требуется заранее для каждого приложения резервировать серверную нагрузку, область хранения и пропускную способность сети – они могут быть выделены «по требованию», а приложение может использовать ровно столько, сколько ему нужно, из общего пула ресурсов. Совершенно естественно желание совместить преимущества централизованной модели с достоинствами облака, собрать вместе доверительность и эффективность использования ресурсов. Возможности для такого объединения есть и обеспечиваются двумя группами технологий: виртуализации и интеграции информации; совмещая их, можно в произвольных масштабах обеспечить динамическое управление ресурсами (Dynamic Resource Scheduling, DRS).

Результатом конвергенции всех облачных новаций может стать корпоративная информационная система, построенная как частное облако (private cloud). В такой системе сочетаются преимущества централизованной модели и облака. Виртуальные устройства, в зависимости от необходимости, могут находиться во внешнем или во внутреннем облаке, а место выполнения приложения диктуется наличием ресурсов и требованиями безопасности. Избранное для работы устройство доступа тоже не имеет значения, это может быть тонкий или толстый клиент, мобильное устройство или настольный компьютера – критична лишь возможность подключения к частному облаку.1

Внутреннее облако есть не что иное, как корпоративная информационная система, построенная на базе динамического ЦОД, которая, кроме использования собственных ресурсов, может заимствовать их из внешнего облака, создавая частное облако. Внутреннее облако можно рассматривать как апофеоз виртуализации, оно позволяет собрать в единый пул разнородные информационные ресурсы предприятия, чтобы потом распределять их «на ходу», по мере возникновения потребностей, что в конечном итоге приводит к более разумному использованию ресурсов и повышению экономической эффективности. Внутренние облака, скорее всего, будут создаваться крупными предприятиями, они будут ровно того размера, какой на самом деле нужен предприятию, если же этих ресурсов окажется недостаточно, то всегда будет возможность «взять в кредит» у провайдера внешнего облака. Что же касается мелких предприятий, то они, вероятно, откажутся от собственных ИТ-подразделений и целиком «уйдут в облака».

8.5.3 Операционная система для облаков

Осенью прошлого года, после сенсационного увольнения генерального директора VMware Дайаны Грин, многие задавались вопросом, чем удивит компания после ее ухода. Новое руководство не заставило себя ждать и в качестве следующей цели назвало операционную систему, способную виртуализировать серверы и системы хранения всего ЦОД в целом. Тогда впервые появилось название Virtual Datacenter Operating System (VDC-OS), эта система не предполагалась в качестве замены традиционным операционным системам – ее функция должна была быть аналогичной, но в масштабах всего центра обработки данных. VDC-OS призвана оперировать всем ЦОД как единым пулом оборудования, динамически распределяя его ресурсы и обеспечивая бесперебойность работы. В апреле 2009 года вышла vSphere4 с уточнением, что – это операционная система для облаков. О значимости этого шага для отрасли можно судить хотя бы по тому, что на мероприятии VMware, посвященном объявлению vSphere 4, присутствовали Джон Чемберс, генеральный директор и президент Cisco, технический руководитель Intel Патрик Гелсингер, вице-президент по стандартным серверам HP Джеймс Моутон и генеральный директор Dell Майкл Делл. Если принять во внимание,

насколько трудно представить такой синклит на одной сцене, значит, событие и в самом деле сверхординарное.

Понять интерес отрасли к vSphere несложно, помимо общих слов имеется грубая реальность – виртуализация серверов может поднять их КПД в несколько раз, но если учесть, что исходная цифра 5%, то это совсем немного. Требуется дальнейший рост, но если оставаться в рамках одного сервера, он невозможен, следовательно, нужно сливать вместе ресурсы всего серверного пула. Только так можно справиться с врожденными дефектами архитектуры x86. Предлагая vSphere 4, в VMware надеются, что с помощью этой системы ИТ-подразделениям удастся создавать самоуправляемые безотказные и самообслуживаемые внутренние облака, как у Amazon или Google, но существующие внутри периметра, охраняемого межсетевыми экранами. В последующем VMware обещает выпустить апгрейды, позволяющие использовать ресурсы и сервисы, предоставляемые из внешних облаков провайдерами этих услуг.

Со структурной точки зрения вместе с vSphere 4 вводится еще один дополнительный уровень между приложениями и железом, своего рода подложка для виртуальных машин. Ядром vSphere остается старый гипервизор VMware ESX, адаптированный для 64-разрядных процессоров. Он приспособлен к работе с процессором Intel Xeon 5500, используя его встроенные возможности для виртуализации и расширенный доступ к памяти. Вполне очевидно, что работа в облаке потребовала расширения функции Virtual SMP, и теперь виртуальная машина может поддерживать до восьми виртуальных процессоров, причем виртуальные процессоры могут работатьна том же самом физическом процессоре, что и виртуальная машина, а в случае нехватки ресурсов – на других процессорах. Наличие этой функции повышает эффективность использования процессорного пула. Увеличены и количественные параметры, память виртуальной машины выросла до 256 Гбайт, число виртуальных сетевых контроллеров возросло с четырех до десяти, соответственно полоса пропускания увеличилась до 30 Гбит/с, а количество операций ввода/вывода в секунду превышает 300 тыс. Благодаря этим возможностям VMware vSphere 4 может агрегировать существенно большее количество виртуальных машин и физических инфраструктурных ресурсов,

образуя огромный логический пул ресурсов, необходимый для создания облака. Это пул можно охарактеризовать следующими количественными параметрами: 32 физических сервера с 2048 ядрами, на которых может работать 1280 виртуальных машин, использующих 32 Тбайт оперативной памяти, 16 Пбайт памяти на системах хранения и 8000 сетевых портов. Анонсируя vSphere 4, VMware демонстрирует, что стремится навести мосты между традиционными и виртуальными ЦОД, а также облаками. Компания утверждает, что возможностей vSphere 4 достаточно для того, чтобы трансформировать существующие и будущие инфраструктуры во внутренние облака, объединить их в частные облака и обеспечить федеративный союз с внешними облаками.

С организационной точки зрения VSphere позволит разрешить одну из важнейших проблем ИТ – оценки их экономической эффективности, а это позволит превратить в реальность деятельность по предоставлению «услуг, оплачиваемых по мере потребления» (pay-as-you-go service) через Web-портал. В целом компания видит свою роль в предоставлении пользователю возможностей для создания внутренних облаков.

Пока у vSphere в облачной области конкурентов не много, и ни один из них не обладает сравнимой полнотой. Microsoft намеревается к концу 2009 года предложить облачную инфраструктуру Windows Azure, которая изначально называлась Windows Cloud, но функционально это совсем иное, не уровень или подложка в понимании VMware. Скорее, Azure можно рассматривать как платформу для разработки Web-приложений, исполняемых во внешнем облаке типа Amazon EC2. С ее появлением разработчик остается с привычными ему средствами, например Visual Studio, добавив к традиционному приложению для десктопа качества, необходимые для работы в облаке. Операционную систему Solaris иногда рассматривают как облачную, поскольку в ней есть гипервизор xVM Server, технологии Solaris Containers и Zones, файловая система ZFS, но это утверждение выглядит не слишком убедительно. Что же касается gOS Linux от компании Good OS, которая назвала свой продукт просто – cloud operating system, то в данном случае имеет место терминологическое совпадение. Эта операционная система предназначена для установки на нетбуки, имеет ограниченную функциональность и служит для доступа во внешнее облако, а не для создания внутренних облаков.

8.5.4 Идеальное Облако и реальные прототипы

Идеальное внутреннее облако (рис. 8.12) состоит из четырех пулов виртуализованных ресурсов: процессоров, памяти, систем хранения и интерконнекта. На таком облаке могут выполняться любые приложения, оно поддерживает доступ к ним с использованием произвольных типов терминалов: толстых и тонких, стационарных и мобильных.

Рис. 8.12 Структура внутреннего облака

Для того чтобы сложилось нечто подобное этому облаку, потребуется немало времени, в первую очередь предстоят большие работы по стандартизации, а пока даже на серверы-лезвия нет каких-либо общих стандартов и невозможно совместить в одном шасси или в одной стойке изделия от разных производителей, что явно противоречит идее создания облаков как универсальных сред.

Самое упорное сопротивление стандартизация в сфере лезвий испытывает со стороны наиболее крупных производителей, HP и IBM. Несколько лет назад IBM все же пыталась предложить собственную спецификацию в качестве стандарта, но серьезной поддержки эта инициатива не получила. В 2009 году свои намерения относительно стандартизации лезвий выражают участники организации Server Systems Infrastructure Forum, но среди них нет основных

игроков серверного рнынка – HP, IBM, Dell и Sun Microsystems; новое движение возглавляют American Megatrends, Samsung и Supermicro. Они считают возможным создать такие условия, когда каждый сможет собирать необходимые конфигурации из покупных лезвий, корзин, мезонинных карт, коммутаторов и компонентов ввода/вывода. Складывающаяся ситуация очень напоминает события начала восьмидесятых, когда массовое производство персональных компьютеров началось с отрытого стандарта IBM PC. Повторение неизбежно, но пока быстрорастущий рынок серверов-лезвий (33% в год) контролируется пятью компаниями – HP, IBM, Sun, Dell и Fujitsu, ситуация сохранится.

Еще одной особенностью, определяющей движение к облакам, является исключительное положение архитектуры x86. Она далеко не оптимальна, даже если принять во внимание все внесенные в нее усовершенствования, но огромное количество готовых под нее приложений, а также многочисленные рыночные аргументы делают ее единственно возможной при создании массивных вычислительных инфраструктур.

8.5.5 Egenera, Dell и Fujitsu

Компания Egenera формально не входит в число лидирующих производителей лезвий, но по своему технологическому статусу она была и остается признанным авторитетом. Egenera вышла на рынок лезвий второй, непосредственно после RLX Technologies, создавшей в свое время лезвия как класс серверов. Впоследствии HP купила RLX, что оказалось предпосылкой для нынешнего лидирующего положения HP, а Egenera продолжает независимое существование. В отличие от RLX, сосредоточенной непосредственно на вычислительных возможностях серверов-лезвий, во главу угла производственной программы Egenera были положены более крупные структуры, эта программа сохранилась до настоящего времени.

В сентябре 2001 года Egenera представила абсолютно оригинальную, не похожую ни на что компьютерную систему BladeFrame, представляющую собой пул из 96 процессоров со свободным сетевым соединением. В ней можно было

обнаружить схожесть с архитектурой NUMA и с менее популярной SAN (System Area Network), но от них система BladeFrame отличалась применением стандартных узлов, в качестве которых использовались двух- или четырехпроцессорные ультратонкие и, что очень существенно, бездисковые серверы. Отсутствие дисков в сочетании с развитой системой управления обеспечивало BladeFrame те качества, которые сегодня дают технологии виртуализации.

В самой первой стойке BladeFrame могло быть смонтировано до 24 двух- или четырехпроцессорных лезвий. Для их объединения в единую систему было разработано оригинальное техническое решение, получившее название PAN (Processing Area Network), в состав которого входят интегрированные коммутаторы, контроллеры и шины для объединения в кластеры. PAN Manager позволяет управлять системой из одной точки, а SAN и PAN, собранные в одной стойке, могут образовать готовую инфраструктуру предприятия или провайдеров различных информационных услуг, в том числе услуг независимого хранения данных.

Сегодня Egenera выпускает два основных продукта – BladeFrame EX и BladeFrame ES, различающиеся между собой размерами. В них устанавливаются двух- или четырехпроцессорные вычислительные лезвия Processing Blade на архитектуре x86 от AMD или Intel. От своих предшественников Processing Blade унаследовали главные черты, прежде всего отсутствие дисков. Для управления в EX и ES устанавливают по два управляющих модуля – соответственно Control Blade EX и Switch Blade EX, которые различаются количеством портов Gigabit Ethernet. Системой управляет PAN Manager, который реализует «виртуализацию без виртуализации», то есть преобразует фиксированную физическую серверную инфраструктуру в пул гибко распределяемых ресурсов. В Egenera эту процедуру называют «оркестровкой инфраструктуры» (Infrastructure Orchestration).

Для региона EMEA, к которому относится Россия, продукция Egenera выпускается под торговой маркой Fujitsu Technology Solutions и известна как Primergy BladeFrame. Изделия Primergy BladeFrame BF200 и BF400 идентичны и взаимозаменяемы с Egenera BladeFrame ES и EX. Вторым лицензиатом является

Dell PAN System, представляющий собой предварительно сконфигурированный набор серверов Dell с предустановленной системой управления PAN Manager.

8.5.6 Cisco

Не удивительно, что комплексная вычислительная система от Cisco, предназначенная для динамического ЦОД, – Unified Computing System – интересна не собственно лезвиями, а тем, как они коммутируются, точнее сочетанием коммутирующей инфраструктуры с виртуальным распределенным коммутатором.

Система Cisco UCS собирается из набора, состоящего из пяти основных компонентов. Ее логическим центром служит устройство Cisco UCS 6100 Series Fabric Interconnects, которое осуществляет физическую коммутацию 10 Gigabit Ethernet и FcoE (Fibre Channel over Ethernet). Функции коммутации могут быть расширены устройством (экстендером) Cisco UCS 2100 Series Fabric Extenders, осуществляющим связь с лезвиями. Сами лезвия Cisco UCS B-Series Blade Servers ничем не отличаются и выпускаются в двух версиях: Cisco UCS B200-M1 Blade Server (до двух процессоров Intel Xeon 5500, до 96 Гбайт памяти DDR3, два диска SAS в разных вариантах с контроллером RAID и одна двухпортовая мезонинная карта) и Cisco UCS B250-M1 Extended Memory Blade Server с увеличенной до 384 Гбайт памятью и двумя мезонинными картами, повышающими скорость обмена с лезвием до 40 Гбит/с. Как и в большинстве подобных конструкций, лезвия устанавливаются в корзину-шасси Cisco UCS 5100 Series Blade Server Chassis, вмещающую до восьми лезвий плюс к ним два экстендера. В мезонины лезвий могут устанавливаться сетевые адаптеры разных типов. Эти небольшие устройства дают возможность приспособить лезвия к разным типам нагрузок, к работе в виртуальной среде, обеспечить их лучшую «подгонку» к сетевой среде, позволяют добиться более высокой производительности.

Изюминка UCS – не в этом железе, а в виртуализации сетевых функций, обеспечиваемых сочетанием возможностей VMware vNetwork Distributed Switch и Cisco Nexus 1000V Series Switch. Хотя в обоих названиях используется термин «коммутатор», оба решения являются программными продуктами. Как же могут быть коммутаторы программными и к тому же распределенными? Появление

гипервизора VMware ESX вызвало к жизни необходимость коммутировать виртуальные машины, и до последнего времени эти действия выполнялись вручную. В 2009 году в VMware vNetwork появились автоматизирующие эти функции виртуальные коммутаторы – стандартный VMware vNetwork Standard Switch и распределенный VMware vNetwork Distributed Switch. Первый действует в пределах одного физического сервера, второй позволяет распространить настройку коммутации виртуальных машин на более чем один физический сервер. Коммутатор Cisco Nexus 1000V позволяет сделать следующий шаг.

8.5.7 Sun Microsystems

Премьера модульного семейства Open Network Systems состоялась в апреле 2009 года, причем его представляли Энди Бехтольсхайм (один из основателей Sun, еще студентом разработавший рабочую станцию Sun 1, а ныне руководящий разработкой серверов стандартной архитектуры), разработчик файловой системы ZFS Джефф Бонвик и главный специалист по флэш-накопителям Майкл Корнуэлл. Из того, что они говорили, совершенно очевидно, что семейство Open Network Systems главным образом нацелено на высокопроизводительные вычисления. Sun не первой начала поставлять лезвия на процессорах Intel Xeon 5500, но ей первой удалось сделать несколько вещей. Прежде всего, упаковать 96 двухсокетных узлов в стандартную стойку высотой 24 дюйма, затем связать эти узлы межсоединением QDR InfiniBand, обеспечивающим скорость передачи 40 Гбит/с, и, наконец, установить в лезвиях ускорители на твердотельных накопителях, где одна плата обеспечивает скорость обмена, равную той, что дают 100 дисков, работающих параллельно. Очевидно, что все перечисленное будет использовано при сборке новой версии суперкомпьютера Sun Constellation System.

Однако потенциал Open Network Systems не сводится лишь к высокопроизводительным вычислениям. Начнем с того, что у Sun существует не один, как у большинства, а два основных конструктива.

Традиционную корзину Sun Blade 6000 дополняют стойки Sun Blade 6048, в них могут собираться системы для HPC, но они же могут использоваться для консолидации и виртуализации серверов, для работы с СУБД и корпоративными приложениями, а также для поддержки инфраструктур Web. В стойке появилась новая система охлаждения с активной дверью Sun Cooling Door Systems, предназначенная для Sun Blade 6048, она способна отбирать до 35 кВт тепла, причем непосредственно из наиболее горячих зон, что особенно критично для систем HPC.

Второй конструктив – собственные коммутаторы. Еще недавно при анализе продуктовой линейки Sun удивляло то, что в ней давно были вычислительные лезвия, в прошлом году к ним появились лезвия для хранения данных Sun Blade 6000 Disk Module, но до сих пор не было того, что объединяет отдельные лезвия в единую инфраструктуру, в то, что называют динамическим ЦОД. Наконец недостающие части появились – коммутаторы Sun Blade 6000 Virtualized NEM и Sun Blade 6048 QDR InfiniBand NEM. Аббревиатура NEM расшифровывается как Network Express Module, а цифры в названии указывают, какому именно конструктиву они предназначены. Младшая модель, Sun Blade 6000 Virtualized NEM, более традиционна, она служит для объединения лезвий в пределах корзины посредством 10Gt Ethernet, а старшая – реальный шаг вперед. Три буквы QDR в ее названии означают Quad Data Rate InfiniBand, то есть передача данных по интерфейсу InfiniBand с учетверенной скоростью. Появление этого коммутатора есть плод многолетнего сотрудничества с израильской компанией Mellanox. В устройстве сочетаются 30 портов 4xQDR, обеспечивающих передачу со скоростью 40 Гбит/с, и 24 порта Gigabit Ethernet.

Еще две новинки – серверы-лезвия Sun Blade X6270 и Sun Blade X6275, построенные на процессорах Quad-Core Intel Xeon, но X6270 ориентирован на корпоративные приложения и на задачи виртуализации и консолидации, поэтому у него большой запас по памяти (18 слотов DIMM, до 144 Гбайт) и четыре диска с горячей заменой, а X6275 специально спроектирован для работы в составе высокопроизводительных HPC-кластеров. Он состоит из двух независимых узлов и имеет интегрированный контроллер QDR InfiniBand, который в сочетании с технологией Intel QuickPath обеспечивает высокую

скорость передачи данных и низкую задержку. Качественная новизна X6275 состоит в поддержке твердотельных дисков SSD, которые смогут использовать преимущества пропускной способности QDR InfiniBand. Модуль размером со спичечную коробку и толщиной несколько миллиметров, вставляемый в материнскую плату, хранит 24 Гбайт данных, используя кэш 64 Мбайт. Скорость работы таких дисков в 50-70 раз выше, чем механических, и это обстоятельство радикально изменяет подходы к проектированию. На представлении Open Network Systems Бехтольсхайм отметил, что X6275 – только первый шаг по части освоения возможностей SSD.

8.5.8 Hewlett-Packard

В конце апреля 2009 года было представлено решение HP BladeSystem Matrix, из названия которого следует, что в нем вся ИТ-система собирается в матрицу, состоящую из лезвий. В отличие от Sun здесь делался упор на экономические преимущества модульных систем и на то, что они являются плодом развития фирменной идеологии, называемой адаптивной инфраструктурой (Adaptive Infrastructure). Главный козырь BladeSystem Matrix – качество интеграции. Здесь все собрано в единое целое: программное обеспечение, серверы, системы хранения, сетевое оборудование. В багаже компании не просто набор необходимых строительных блоков, но еще и система управления HP Matrix Orchestration Environment, обеспечивающая универсальный интерфейс ко всем процедурам проектирования, внедрения и последующей оптимизации среды выполнения приложений. Опираясь на HP MOE, пользователь сможет собрать до тысячи физических или виртуальных серверов в единую адаптивную инфраструктуру.

Базовый конструктивный элемент – «корзина» BladeSystem c7000, обеспечивающая питание, охлаждение, инфраструктуру ввода/вывода и межсоединение для установки 16 обычных серверных лезвий или специализированных лезвий-накопителей. Сейчас «матрица» комплектуется серверами HP ProLiant G6 на четырехъядерных процессорах Intel Xeon 5500 или AMD Opteron (Shanghai). В перспективе можно будет использовать лезвия на Itanium, в том числе на будущих двухъядерных Itanium 9100. По способности к

горизонтальному масштабированию Matrix заметно превосходит Cisco UCS, использование в последней коммутационной структуры с 40 портами ограничивает общее число серверов до 320 против 1000 в BladeSystem Matrix. Для коммутации используются модули, поддерживающие соединения 10 Gigabit Ethernet и Fibre Channel с пропускной способностью 8 Гбит/с; допускается использование высоконадежной системы коммутации HP NonStop и нескольких вариантов системы питания – однофазной, трехфазной и постоянного тока.

Вторая составляющая BladeSystem Matrix – интегрированный центр управления HP Insight Dynamics – Virtual Server Environment, предназначение которого в непрерывном анализе и оптимизации состояния всей адаптивной инфраструктуры. Именно благодаря наличию этого интеллектуального центра BladeSystem Matrix приобретает качества, которые позволяют назвать эту структуру «облаком в миниатюре».

8.5.9 emc

Накопитель старшего уровня V-Max (Virtual Matrix Architecture), как следует из названия, построен с использованием виртуальной матричной архитектуры. При его объявлении было указано на то, что этот массив предназначен для частных или корпоративных облачных инфраструктур. Он назван виртуальным, поскольку предполагается возможность построения мощных и даже сверхмощных систем хранения из относительно простых модулей.

Основным компонентом для сборки виртуальной модели Symmetrix V-Max является модуль V-Max Engine, представляющий собой управляющий узел, построенный на четырех процессорах Intel Xeon Quad-core, объем оперативной памяти одного V-Max Engine составляет 128 Гбайт, в нем установлены две группы интерфейсов по 16 каналов для подключения дисков и хостов. Скорость обмена данными с периферией равна 24 Гбайт/с. Кроме всего прочего, «на борту» имеется специализированная микросхема для управления доступом к памяти и интерконнектом RapidIO.

Механизм виртуализации позволяет объединить до восьми модулей V-Max Engine в единый системный образ, в котором все характеристики суммируются,

а это значит, что размер общей памяти становится равным 1 Тбайт, а виртуальная скорость обмена достигнет 192 Гбайт/с. Количественный рост обеспечивает качественное повышение показателей. Общее число портов для подключения дисков в таком случае составит 128, к ним можно подключаться практически по любым протоколам – Fibre Channel, FICON и iSCSI. Емкость дисков может варьироваться от 146 Гбайт до 1 Тбайт, а флэш-накопителей – от 200 до 400 Гбайт. В максимальной конфигурации система будет состоять из одной управляющей стойки и восьми стоек с накопителями.

Если говорить об объемах хранения, то в системе может быть установлено от 96 до 2400 дисков. В зависимости от типов дисков и RAID полезный объем хранения колеблется от 6 до 2026 Тбайт. Существенно то, что можно сочетать диски разных типов, а значит, оптимизировать состав в зависимости от решаемой задачи, подбирая соответственно флэш-диски – быстрые, но малые по объему или же большие, но более медленные. Для распределения данных по уровням в Symmetrix V-Max имеется средство FAST (Fully Automated Storage Tiering), оно поддерживает эту функцию автоматически в непрерывном режиме.

Главное преимущество виртуализованной системы хранения – в неограниченном масштабировании, оно может быть вертикальным (scale in) и горизонтальным (scale out). Массив Symmetrix V-Max может расти в процессе своего жизненного цикла, может увеличиваться за счет наращивания числа стоек, а по мере появления более производительных V-Max Engine – к тому же за счет замены старых на новые. Вовсе не обязательно, чтобы все строительные блоки были одинаковыми, на то она и виртуализация. Таким образом создается эволюционирующая среда, не вызывающая перерывов в обслуживании.

С технической точки зрения виртуализация V-Max Engine стала возможной во многом благодаря тому, что для обеспечения соединений между модулями и для прямого доступа в память использована коммутационная инфраструктура со стеком протоколов RapidIO. Этот относительно малоизвестный альтернативный протокол изначально был предложен для встроенных систем, но неожиданно, как иногда бывает, стал осваивать ту площадку, где сейчас господствует InfiniBand. Обеспечивающий обмен данными на скоростях до 60 Гбайт/с,

протокол RapidIO, разработанный специалистами компаний Mercury Computer Systems и Motorola, построен на принципах коммутации пакетов и существует в последовательной и параллельной версиях. RapidIO позволяет разносить узлы на расстояние в несколько десятков метров. Помимо скорости, от встроенных систем RapidIO унаследовал естественный для них прямой доступ к памяти, без которого невозможно создать систему, работающую в режиме реального времени. В приложении к управлению системами хранения принципиально важно то, что использование RapidIO открывает возможность для создания архитектуры с когерентным доступом к памяти (coherent-memory architecture). Это, собственно, и дает все те преимущества виртуализации, о которых уже шла речь. Когерентный доступ уравнивает в правах доступа к памяти вызовы от локальных и удаленных процессоров, что в конечном итоге обеспечивает виртуализацию, а по существу, объединение отдельных модулей в единый системный образ. Теоретически количество V-Max Engine может быть существенно больше нынешних восьми (в некоторых источниках называлась цифра 256), что позволит создавать неограниченные по размеру системы хранения, предназначенные не только для крупных облачных ЦОД, но, возможно, и глобального характера.

Заключение

Общие проблемы современных ИТ-сред: избыточность вычислительных ресурсов и неэффективное их использование, сложность управления инфраструктурой, слабая связь между технологиями и бизнес-задачами. парадигма вычислений по требованию нацелена на разрешение этих проблем, но она в свою очередь требует управления на совсем ином уровне. Зачинатели направления on-demand — компании IBM и HP — являются лидерами рынка ИТ-управления, и в их системах Tivoli и OpenView, естественно, уже появляется много новых средств по управлению адаптивными возможностями информационных инфраструктур.

Литература

1. Michael Champion, SOA: One acronym to bind them all? weblogs.java.net.
2. Jonathan Sapir, Will Web services and SOA change the development world? www.TechRepublic.com, August 2003.
3. Laws of Evolution: A Pragmatic Analysis of the Emerging Web Services Market. Stencil Group, April 2002.
4. Introduction to Web-services Architecture. Systinet.
5. Schutle, Predicts 2003: SOA Is Changing Software, Gartner Group, December 2002.
6. Enterprise Service Bus Will Revolutionize Information Technology, IDC, March 2003.
7. Service-oriented architecture (SOA), definition. Barry&Associates, www.service-architecture.com.
8. Gregor Hohpe, Web Services: Pathway to Service-oriented architecture, ThoughtWorks.
9. Michael Pallos, Service-oriented architecture: A Primer. EAI Journal, December 2001.
10. Don Roeder, The Next Wave of Integration Platforms. EAI Journal, September 2002.
11. Леонид Черняк, "Следующая волна технической революции: от баз данных к распределению данных". // *Открытые системы*, № 3, 2003.
12. Dion Wiggins, Net vs. Java: Competition or Coopetition? Gartner Group.
13. Web Services Architecture Working Group, *http://www.w3.org/2002/01/ws-arch-charter*.
14. Web Services Glossary, *http://www.w3.org/TR/2003/WD-ws-gloss-20030808*.
15. Web Services Architecture, W3C Working Draft 8, August 2003, *www.w3.org/TR/2003/WD-ws-arch-20030808*.
16. Chris Peltz, Web services orchestration, a review of emerging technologies, tools, and standards. Hewlett-Packard, January 2003.
17. Леонид Черняк, MOM, второе рождение. // *Открытые системы, 2001, № 10*
18. Валерий Коржов, Почтальон для приложений. // *Открытые системы, 2001, № 10*

19. Stuart J. Johnston, Web Services Wars Take Artistic Turn. Choreography or orchestration? XML Magazine, 2002, № 10/11

20. David E. Bakken, «Middleware», глава из книги Encyclopedia of Distributed Computing, Kluwer Academic Press, 2001, *www.eecs.wsu.edu/~bakken/middleware.pdf*.

21. Scott Grant, Michael P. Kovacs, Meeraj Kunnumpurath, Silvano Maffeis, K. Scott Morrison, Gopalan Suresh Raj, Paul Giotta, «Professional JMS». (Информацию об этой книге и отдельные главы из нее можно найти в Web по адресу *http://www.execpc.com/ ~gopalan/jms/ jms.html*

Список литературы

Основная:

1. К. Дж. Дейт «Введение в системы баз данных». 6-е изд.,М.-1998г.
2. Дейл Роджерсон. «Основы COM». М.1997г.- 376с.
3. Д.Чеппел. Технологии ActiveX и OLE. /Microsoft Press. Lern Java Now/,1997,376с.
4. Д. Васкевич. Стратегии “клиент/сервер”. 2-е изд. – К. “Диалектика”, 1996г., 396с.
5. С.Дунаев. "Интранет технологии”. – М. “Диалог – МИФИ”, 1997 - 287с.
6. С.Дунаев. "Доступ к базам данных и техника работы в сети”. – М. “Диалог – МИФИ”, 1999 - 416с.
7. Гери Хансен, Джеймс Хансен. Базы данных: разработка и управление.- М. ЗАО «Издательство БИНОМ», 1999. – 704с.
8. Г.Н. Калянов. «CASE – технологии. Консалтинг в автоматизации бизнес-процессов». – М. «Горячая линия-Телеком», 2000, 317с.
9. Саймон А.Р. Стратегические технологии баз данных: менеджмент на 2000 год: Пер. с англ. /Под ред. и с предисл. М.Р. Когаловского.- М.: Финансы и статистика, 1999.- 479с.:ил.

Дополнительная:

10. ЖУРНАЛ «Мир ПК» (1999-2002р.р.).
11. ЖУРНАЛ «Компьютерные сети и системы» (1997-2002).
12. ЖУРНАЛ «СУБД» (1998-2002)
13. ЖУРНАЛ «Компьютеры + программы» (1998-2002)
14. ЖУРНАЛ «Открытые системы» (1999-2003) - http://www.osmag.ru
15. www.osp.ru/system
16. www.lanit.ru
17. http://master.cmc.msu.ru
18. Л. Черняк. Стоит ли бежать впереди паровоза.- М.: журн. "Открытые системы". 2002, №11
19. И. Гринберг, Ли Гарбер. Разработка новых технологий информационного поиска.- М.: журн. "Открытые системы", 1999, № 09-10