Архитектура клиент-сервер

Появление архитектуры клиент-сервер, как и многих других новых компьютерных технологий, сопровождалось рождением новой терминологии. В табл. 17.1 перечислены некоторые термины, часто встречающиеся в описаниях продуктов и приложений клиент-сервер.

- **Прикладной программный интерфейс (Application Programming Interface, API)**
  Набор функций и подпрограмм, обеспечивающих взаимодействие клиентов и серверов
- **Клиент**
  Объект, запрашивающий информацию по сети. Как правило, это персональный компьютер или рабочая станция, запрашивающая информацию у сервера
- **Промежуточное программное обеспечение**
  Набор драйверов, прикладных программных интерфейсов и прочего программного обеспечения, позволяющего улучшить взаимодействие между клиентским приложением и сервером
- **Реляционная база данных**
  База данных, в которой доступ к информации ограничен выбором строк, удовлетворяющих определенным критериям поиска
- **Сервер**
  Компьютер (как правило, высокопроизводительная рабочая станция, мини-компьютер или мэйнфрейм), хранящий информацию, с которой работают сетевые клиенты
- **Язык структурированных запросов (Structured Query Language, SQL)**
  Разработанный корпорацией IBM и стандартизованный институтом ANSI язык для создания, управления и изменения баз данных

Знакомство с архитектурой клиент-сервер

Вычислительная модель клиент-сервер заняла прочное место среди методов распределенных вычислений. И хотя разные производители предлагают разное программное обеспечение, о том, что такое архитектура клиент-сервер, вполне сложилось единое мнение. В табл. 17.2 процитированы некоторые определения понятия «клиент-сервер». Внимательно прочитайте их. На этих определениях основаны некоторые основные темы этой главы.

Вычислительная модель клиент-сервер

Разделение приложения на отдельные задачи, размещаемые на различных платформах для большей эффективности. Как правило, это означает, что программа представления данных находится на машине пользователя (на клиенте), а программа управления данными и сами данные — на сервере. В зависимости от приложения и используемого программного обеспечения вся обработка данных может осуществляться на клиентской машине или распределяться между клиентом и сервером. Сервер соединяется со своими клиентами по сети. Серверное программное обеспечение принимает запросы от клиентского программного обеспечения и возвращает ему результаты [28]

Архитектура клиент-сервер

Сетевое окружение, в котором управление данными осуществляется на серверном узле, а другим узлам предоставляется доступ к данным [59]

Вычислительная модель клиент-сервер

Совместная с клиентом обработка запросов клиента сервером и возвращение

результатов клиенту. В этой модели обработка данных приложением распределена (не обязательно поровну) между клиентом и сервером. Обработка данных инициируется и частично управляется клиентом, но не в режиме «ведущий-ведомый», а, скорее, в режиме сотрудничества [5]

Вычислительная модель клиент-сервер

Модель взаимодействия между одновременно выполняемыми программными процессами. Клиентские процессы посылают запросы серверному процессу, посылающему обратно результаты этих запросов. Как предполагает название, серверные процессы предоставляют услуги своим клиентам, как правило, выполняя специфическую обработку, которую могут выполнить только они. Клиентский процесс, освобожденный от выполнения сложной обработки транзакции, может выполнять другую полезную работу. Взаимодействие между клиентским и серверным процессами представляет собой совместный транзакционный обмен, в котором активность исходит от клиента, а сервер реагирует на эту активность [90]

Приложение клиент-сервер

Любое приложение, в котором инициатор действия находится в одной системе, а исполнитель действия — в другой. Кроме того, в большинстве приложений клиент-сервер один сервер обслуживает запросы нескольких клиентов [75]

На рис. 17.1 мы попытались проиллюстрировать суть этих определений. Как предполагает термин, окружение клиент-сервер состоит из клиентов и серверов. Клиентские машины, как правило, представляют собой однопользовательские персональные компьютеры или рабочие станции, предоставляющие конечным пользователям дружественный интерфейс. Клиентская станция обычно имеет наиболее удобный графический интерфейс пользователя, предполагающий наличие окон и мыши. Наиболее известные примеры подобных интерфейсов — интерфейсы операционных систем Microsoft Windows и Macintosh. Клиентские приложения предполагают простоту использования и знакомые инструментальные средства, например, электронные таблицы.

Каждый сервер в окружении клиент-сервер предоставляет клиентам набор услуг. Наиболее распространенным типом сервера в архитектуре клиент-сервер является сервер баз данных, как правило, управляющий реляционной базой данных. Высокопроизводительный сервер обеспечивает коллективный доступ нескольких клиентов к одной и той же базе данных.

Помимо клиентов и серверов в окружение клиент-сервер входит сеть. Вычислительная модель клиент-сервер по определению является распределенной. Пользователи, приложения и ресурсы располагаются на разных компьютерах и соединены общей локальной, глобальной или составной сетью.

В чем отличие конфигурации клиент-сервер от других распределенных решений? Есть несколько характеристик, отличающих вычислительную модель клиент-сервер от обычных схем распределенных вычислений.

- В приложениях клиент-сервер большое внимание уделяется созданию на клиентской машине дружественного пользователю интерфейса. Таким образом, пользователь получает полный контроль над расписанием и режимом работы компьютера, а менеджеры уровня отделов получают возможность реагировать на локальные проблемы.
- Хотя приложения являются распределенными, в архитектуре клиент-сервер, как правило, используются централизованные корпоративные базы данных. Это позволяет руководству корпорации сохранять полный контроль над инвестициями в информационные системы, а также обеспечивать полную связность всех систем. В то же время такая конфигурация избавляет различные отделы компании от накладных расходов по управлению сложными вычислительными системами, но позволяет им выбирать типы машин и интерфейсы, которые им необходимы для доступа к данным. С

подобным сочетанием централизованного и распределенного управления мы уже знакомы (см. раздел «Конкретный пример — компания Levi Strauss» в главе 15).

- Как корпоративные пользователи, так и производители отдают предпочтение открытым и модульным системам. Это означает, что пользователю предоставляется более широкий выбор продуктов и большая свобода в объединении оборудования от различных производителей.
- Компьютерная сеть является ключевым звеном данной архитектуры. Поэтому вопросы сетевого администрирования и сетевой безопасности при работе с информационными системами данного типа имеют приоритет.

С одной стороны, архитектура клиент-сервер представляет собой естественное решение с точки зрения производителя, так как в ней используются все более доступные микрокомпьютеры и сети. С другой стороны, архитектура клиент-сервер, возможно, является идеальным выбором для поддержки выбранного организацией направления бизнеса.

Последнее утверждение требует пояснений. Успех архитектуры клиент-сервер на рынке объясняется не новыми названиями старых решений. Вычислительная модель клиент-сервер действительно представляет собой новый технический метод распределенных вычислений. Но помимо этого, архитектура клиент-сервер создает условия для новых методов организации бизнеса. Рассмотрим две важные тенденции в промышленности, иллюстрирующие этот факт.

Первая тенденция заключается в том, что компании постоянно пытаются снижать трудовые затраты и избавляться от лишних рабочих мест, что вызвано жесткой рыночной конкуренцией. Почему компаниям необходимо сокращать рабочие места, чтобы сохранить конкурентоспособность, и как им удается увеличивать производительность, добиваясь роста продаж без увеличения числа сотрудников? Стоимость каждого рабочего места стремительно растет, и этот рост сопровождается ростом заработной платы. В то же время стоимость рабочего оборудования, особенно компьютерного и сетевого, а также сетевого обслуживания увеличивается гораздо более скромными темпами. Все это привело, как и следовало ожидать, к существенному росту капиталовложений в компьютеры и информационные технологии, чтобы компенсировать сокращение штата сотрудников.

Эта тенденция просматривается как в малом, так и в крупном бизнесе и затрагивает управленцев среднего звена, а также конторских служащих. Архитектура клиент-сервер предоставляет средство автоматизации задач и устранения барьеров для информации, что позволяет компаниям удалять лишние управленческие звенья и увеличивать производительность, не раздувая штаты.

Другой тенденцией, иллюстрирующей эффективность архитектуры клиент-сервер, является так называемое движение внутреннего рынка. Это движение затрагивает, в первую очередь, крупный бизнес, пытающийся сочетать предпринимательское рвение с корпоративной мощью, чтобы получить лучшее и от того, и от другого: экономию, обусловленную крупными масштабами большого бизнеса, и гибкость малого бизнеса. В эпоху быстрых технологических и рыночных изменений многие крупные компании отказались от традиционной функциональной иерархии, заменив ее набором относительно независимых организационных единиц. Эти организационные единицы должны конкурировать с внешними компаниями. На внутреннем рынке каждая организационная единица функционирует как независимая компания. Каждая организационная единица сама решает, что и у кого ей покупать (независимо от того, является поставщик подразделением той же корпорации или же внешней компанией). Даже такие традиционные непроизводительные подразделения, как бухгалтерия, информационные системы и юридический отдел, должны продавать свои услуги другим подразделениям и конкурировать с внешними поставщиками.

Внутренняя конкуренция призвана исправить недостатки традиционного метода ведения бизнеса. В [93] отмечается, что «Американские корпорации являются одними из самых крупных социалистических бюрократий в мире. Они характеризуются централизованным планированием, централизованным владением капитала, централизованным распределением ресурсов, субъективной оценкой труда, отсутствием внутренней конкуренции и склонностью принимать решения в ответ на политическое давление».

Движение внутренних рынков уже трансформировало некоторые компании, и обещает оказать существенное влияние на другие. Однако до недавних пор на пути реализации подобной схемы было одно труднопреодолимое препятствие. В крупной компании наличие внутреннего рынка может привести к тому, что тысячам отделов придется постоянно договариваться друг с другом и с внешними организациями. Проанализировав эту ситуацию, можно предположить, что стоимость и сложность бухгалтерского учета всех транзакций превысит пользу от введения внутреннего рынка. Это препятствие было преодолено благодаря развитию вычислительных технологий. Сегодня ряд транснациональных корпораций пользуются новейшими системами управления базами данных, работающими в сетях с архитектурой клиент-сервер, что позволяет им внедрять идею внутреннего рынка [93].

Эти и другие тенденции в мире бизнеса послужили стимулом к увеличению инвестиций в технологию клиент-сервер. На рис. 17.2 показаны функциональные области, в которых активно применяются приложения клиент-сервер. График основан на результатах опроса 350 менеджеров информационных систем [31].

Разумеется, как и любое кардинальное изменение компьютерной конфигурации, переход на архитектуру клиент-сервер не является ни безопасным, ни безболезненным. В табл. 17.3 показано, что пользователи при переходе на архитектуру клиент-сервер, помимо получаемых преимуществ, сообщают о множестве проблем. Тем не менее благодаря снижению стоимости и росте популярности персональных компьютеров, а также благодаря растущей конкуренции в промышленности, архитектура клиент-сервер в обозримом будущем будет, скорее всего, доминировать в бизнесе.

Таблица 17.3. Достоинства и недостатки архитектуры клиент-сервер

Системная характеристика	**Значение**
Достоинства	
Сеть небольших мощных машин	Если одна машина выйдет из строя, ваша компания все равно сможет продолжать работу
Мощные объединения компьютеров	Система предоставляет мощность, позволяющую выполнять работу без монополизации ресурсов. У конечных пользователей достаточно мощностей для локальной работы
Некоторые рабочие станции столь же мощны, как мэйнфреймы, но их стоимость на порядок ниже	Предоставляя вычислительные мощности за меньшие деньги, система позволяет вам тратить сэкономленные средства на другие приобретения или на увеличение ваших доходов
Открытые системы	Аппаратуру, программы и услуги можно приобретать у разных поставщиков
Легкость наращивания системы	Вашу систему нетрудно модернизировать, как только ваши потребности изменятся
Индивидуальная рабочая среда клиента	Вы можете объединять компьютерные платформы, подбирая их под конкретные нужды подразделений и пользователей
Недостатки	

Слабая поддержка	Отдельные части системы не всегда корректно работают вместе. Бывает довольно трудно найти причину неисправности
Недостаток инструментальных средств обслуживания	При использовании архитектуры клиент-сервер часто приходится искать инструментальные средства на рынке или разрабатывать их самостоятельно
Необходимость переобучения	Философия программирования для Mae или Windows существенно отличается от философии программирования на языке COBOL или C

Эволюция архитектуры клиент-сервер

Идея организовать вычислительные ресурсы в соответствии с архитектурой клиент-сервер возникла на уровне рабочих групп и подразделений компаний. Менеджеры отделов обнаружили, что использование централизованных приложений, работающих на мэйнфрейме, не дает им достаточно быстро реагировать на требования бизнеса. Развертывание приложений центральным отделом информационного обслуживания требовало очень много времени, а результаты не всегда соответствовали нуждам подразделений. С развитием персональных компьютеров работники получили возможность хранить и обрабатывать данные прямо на своих рабочих местах, а менеджеры отделов — быстро находить нужные приложения.

Однако в окружении, состоящем исключительно из персональных компьютеров, пользователи при совместной работе сталкивались с определенными трудностями. Даже на уровне одного подразделения компании было необходимо поддерживать базу данных, а также форматы и стандарты их использования. Решение проблем дает архитектура клиент-сервер, развернутая на уровне подразделения. Как правило, подобная архитектура включает одну локальную сеть, несколько персональных компьютеров и один или два сервера.

Успех систем клиент-сервер уровня подразделения проложил дорогу системам клиент-сервер уровня предприятия. В идеальном случае подобная архитектура позволяет интегрировать ресурсы подразделений компании и отдела информационного обслуживания, а также запускать приложения, предоставляющие пользователям контролируемый доступ к корпоративным базам данных. Важной чертой подобных архитектур является то, что центральный отдел информационного обслуживания вновь возвращает себе полный контроль над данными, но уже в контексте распределенной вычислительной системы.

Приложения клиент-сервер

Важнейшей особенностью вычислительной модели клиент-сервер является распределение прикладных задач между клиентами и серверами. Иллюстрация общего случая приведена на рис. 17.3. Как на клиенте, так и на сервере базовым программным обеспечением является, разумеется, операционная система. Аппаратные платформы и операционные системы клиентов и серверов могут отличаться. В самом деле, в едином окружении могут использоваться разные типы клиентских и серверных платформ и операционных систем. Однако эти различия не имеют значения, если сервер и клиент используют одни и те же коммуникационные протоколы и поддерживают одинаковые приложения.

Взаимодействие клиента и сервера обеспечивается коммуникационным программным обеспечением. Примерами такого программного обеспечения являются набор протоколов TCP/IP,- протоколы OSI, а также различные фирменные архитектуры, вроде SNA.

Разумеется, назначение всего этого программного обеспечения поддержки (протоколов и операционной системы) заключается в предоставлении базы для распределенных приложений. В идеальном случае выполняемая приложением функция должна быть распределена между клиентом и сервером таким образом, чтобы вычислительные и сетевые ресурсы использовались оптимально, а пользователи получили оптимальные возможности для выполнения различных задач и совместной работы. В некоторых случаях для этого может быть необходимо, чтобы большая часть программного обеспечения выполнялась на сервере, тогда как в других случаях большая часть логики может быть реализована на клиенте.

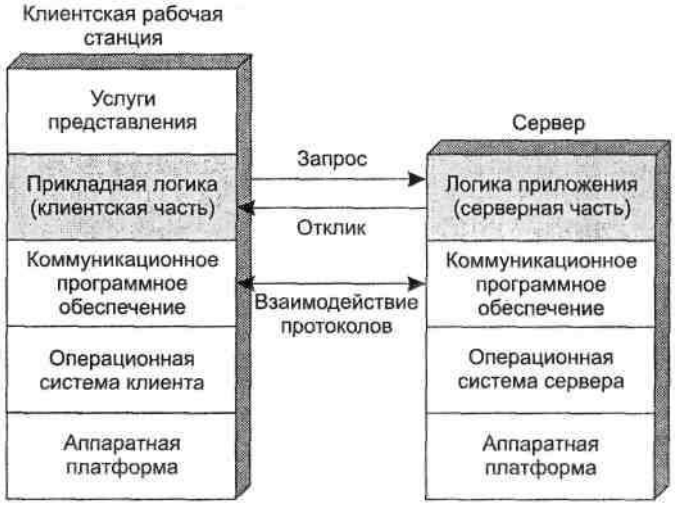

Рис. 17.3. Общая архитектура клиент-сервер

Наконец, существенным фактором успеха является метод взаимодействия пользователя с системой, то есть большое значение имеет пользовательский интерфейс клиентской машины. В большинстве систем клиент-сервер *графическому интерфейсу пользователя* (Graphical User Interface, GUI) уделяется очень серьезное внимание — он должен быть простым и удобным, но одновременно мощным и гибким. Таким образом, модуль услуг представления на рабочей станции можно считать ответственным за дружественный интерфейс с распределенными приложениями.

Модуль услуг представления не следует путать с уровнем представления эталонной модели OSI. Уровень представления занимается форматированием данных для их корректной интерпретации каждым из двух взаимодействующих компьютеров. Модуль услуг представления имеет дело с взаимодействием пользователя и приложения, а также с функциональностью графического интерфейса пользователя.

Базы данных

Рассмотрим концепцию распределенной между клиентом и сервером логики приложения на примере реляционной базы данных. В этой среде сервер является сервером баз данных. Взаимодействие между клиентом и сервером осуществляется в форме транзакций, в которых клиент посылает серверу запрос и получает ответ на него.

Архитектуру этой системы иллюстрирует рис. 17.4. Сервер отвечает за управление базой данных. На клиентских машинах могут располагаться различные приложения, пользующиеся базой данных. Специальное программное обеспечение связывает клиента и сервера, позволяя клиенту выполнять запросы и получать доступ к базе данных. Популярным примером такой логики является язык структурированных запросов (Structured Query Language, SQL).

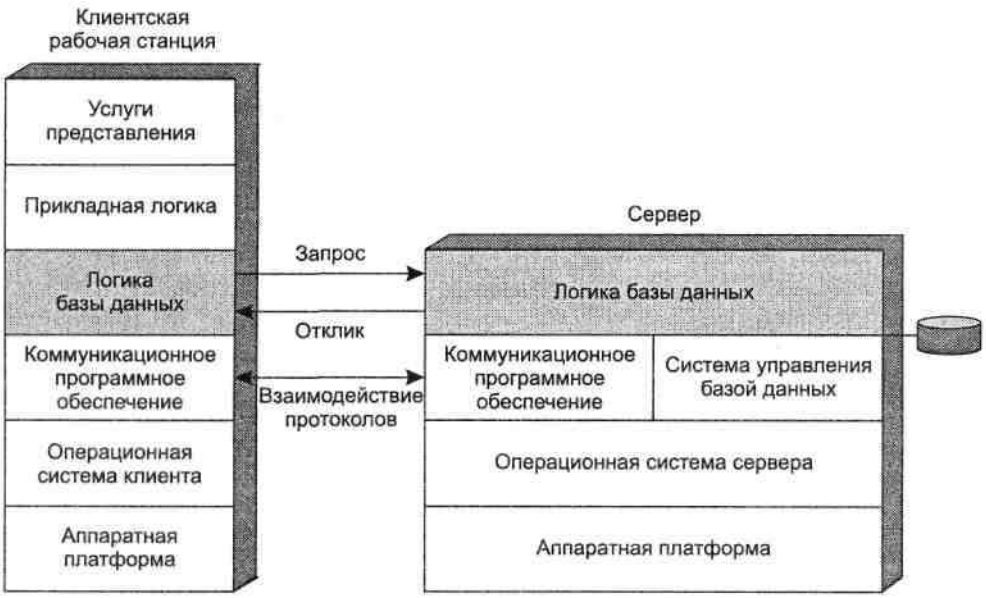

Рис. 17.4. Архитектура клиент-сервер для баз данных

На рис. 17.4 предполагается, что вся прикладная логика — программы для обработки и анализа данных — располагается на клиентской стороне, тогда как сервер занимается только управлением базой данных. Приемлемость такой конфигурации зависит от стиля и задачи конкретного приложения. Предположим, например, что основная цель приложения заключается в обеспечении доступа для поиска записей в режиме подключения (on-line). Пример работы такой схемы показан на рис. 17.5, *а*. Предположим, что сервер управляет базой данных, содержащей один миллион записей (на жаргоне реляционных баз данных называемых строками), и пользователь хочет выполнить операцию поиска, результатом которой может быть нуль записей, одна запись или небольшое количество записей. Пользователь может искать эти записи по нескольким критериям поиска (например, записи, сделанные ранее 1992 года; записи, касающиеся жителей штата Огайо; записи, относящиеся к специфическим событиям или характеристикам, и т. д.). Начальный запрос клиента вызывает ответ сервера о том, что в базе данных содержится 100 000 записей, удовлетворяющих критериям поиска.

При этом пользователь задает дополнительные критерии, и посылает новый запрос. На этот раз сервер отвечает, что в базе данных содержится 1000 записей, удовлетворяющих критериям поиска. Наконец, клиент посылает третий запрос с дополнительными критериями. Результатом поиска на этот раз является одна запись, которая возвращается клиенту.

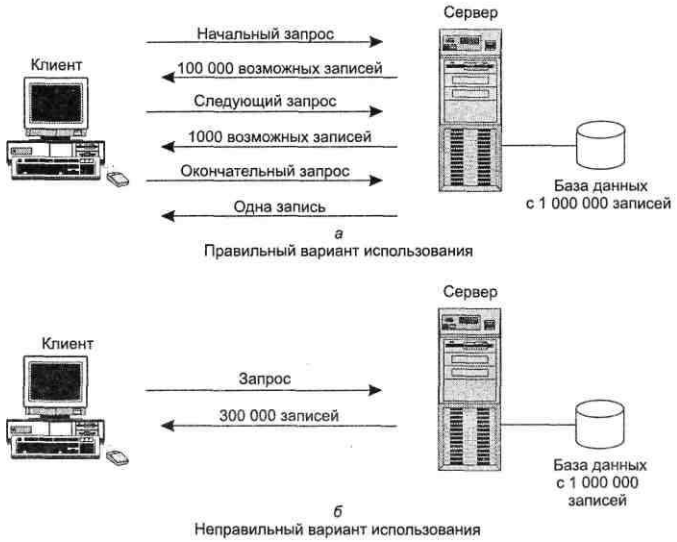

Рис. 17.5. Использование базы данных в архитектуре клиент-сервер

Данное приложение хорошо соответствует архитектуре клиент-сервер по двум причинам.

- С базой данных производится большой объем работ по сортировке и поиску данных. Для этого необходим большой диск или массив дисков, высокоскоростной центральный процессор и высокоскоростная архитектура ввода-вывода. Такие мощности не нужны на однопользовательской рабочей станции или на персональном компьютере.
- Перемещение на клиентскую машину файла с миллионом записей для поиска явилось бы слишком тяжелым бременем для сети. Таким образом, серверу недостаточно просто получать доступ к записям от имени клиента. Сервер должен обладать логикой базы данных, позволяющей ему выполнять операции поиска от имени клиента.

Рассмотрим теперь рис. 17.5, *б*, который иллюстрирует сценарий работы с той же самой базой данных, содержащей миллион записей. В этом случае в результа-

те одного запроса пользователю по сети пересылается 300 000 записей. Такое может случиться, если, например, пользователь захочет определить сумму или среднее значение определенного поля в большом множестве записей или даже во всей базе данных.

Разумеется, подобные действия являются недопустимыми. Одно из возможных решений данной проблемы состоит в том, чтобы переместить часть логики приложения на сервер. То есть помимо функций доступа к записям базы данных и функций поиска записей, сервер может быть оснащен прикладной логикой анализа данных.

Классы приложений клиент-сервер

В рамках общей структуры приложений клиент-сервер выполняемая работа может быть разделена между клиентом и сервером по-разному. Точная доля исполняемых операций и объем передаваемых по сети данных зависят от природы информации, содержащейся в базе данных, поддерживаемых типов приложений, доступности оборудования, которое может работать совместно, а также от характера использования данных в организации.

Схемы некоторых основных вариантов приложений баз данных показаны на рис 17.6. Возможны также другие варианты распределения задач между сервером и клиентом. В любом случае стоит изучить этот рисунок, чтобы получить представление о возможных компромиссах.

На рисунке изображены четыре класса приложений с разными вариантами распределения задач между сервером и клиентом.

♦ *Обработка данных на базе хоста.* Данная схема не является настоящим приложением клиент-сервер, а относится к традиционному окружению мэйнфрейма, когда вся или практически вся обработка данных осуществляется на главной вычислительной машине. Зачастую в подобной вычислительной среде интерфейс пользователя предоставляет примитивный терминал. Но даже если пользователь сидит за персональным компьютером, роль персонального компьютера в этом случае ограничивается эмуляцией терминала.

♦ *Обработка данных на базе сервера.* Простейшим классом конфигурации клиент-сервер является схема, в которой клиент отвечает лишь за предоставление графического интерфейса пользователя[1], тогда как практически вся обработка данных осуществляется на сервере.

♦ *Обработка данных па базе клиента.* Данная схема представляет собой другую крайность. Практически вся обработка данных осуществляется на клиенте, за исключением процедур проверки целостности данных и прочей логики, относящейся к обслуживанию базы данных, которые лучше исполнять на сервере. Как правило, наиболее

сложные функции для работы с базой данных располагаются на клиентской стороне. На сегодняшний день подобная схема, возможно, является наиболее распространенной реализацией архитектуры клиент-сервер. Она позволяет пользователю работать с приложениями, соответствующими его локальным потребностям.

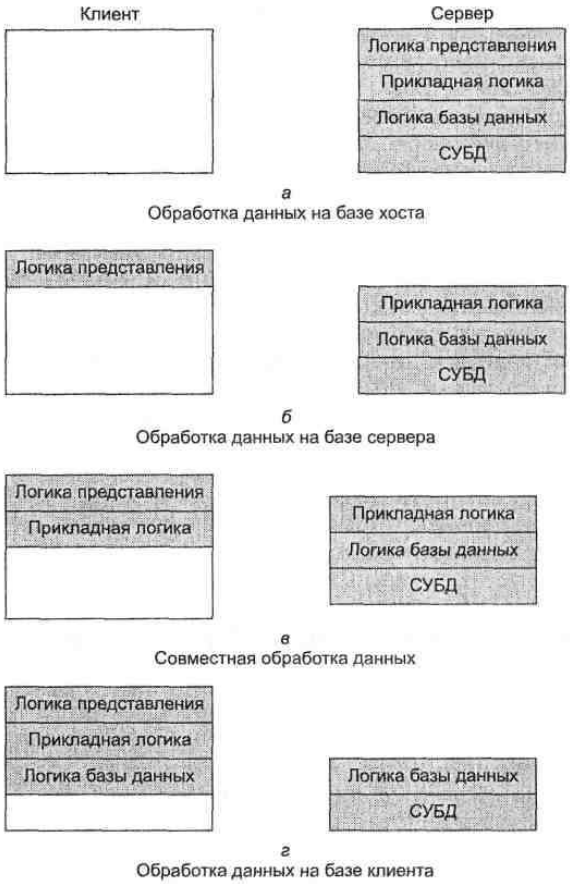

Рис. 17.6. Классы приложений клиент-сервер

♦ *Совместная обработка данных.* В данной конфигурации обработка данных оптимизирована таким образом, чтобы использовать сильные стороны как клиента, так и сервера, а также самого факта распределения данных. Подобные конфигурации гораздо сложнее в установке и обслуживании, но в долговременной перспективе они позволяют обеспечить лучшие показатели производительности и эффективности использования сетевых ресурсов, чем другие методы реализации архитектуры клиент-сервер.

Рисунок 17.6, в и г соответствует конфигурациям, в которых существенная нагрузка ложится на клиента. Эта так называемая модель «толстого» клиента стала популярной благодаря появлению таких инструментальных средств как PowerBuilder корпорации Powersoft и SQL Windows корпорации Gupta. Данные инструментальные средства рассчитаны на создание приложений уровня подразделения с поддержкой от 25 до 150 пользователей [34]. Главное достоинство модели «толстого» клиента заключается в том, что она максимально опирается на вычислительные возможности настольного компьютера, освобождая сервер от излишней нагрузки по обработке данных и, таким образом, повышая их эффективность и снижая вероятность того, что сервер станет узким местом системы.

Однако у стратегии «толстого» клиента имеются и недостатки. Оснащение настольных компьютеров новыми функциями быстро истощает их вычислительные мощности, заставляя компании проводить модернизацию. Если в работу приложения вовлекаются множество пользователей из разных подразделений, компании приходится устанавливать локальную сеть с высокой пропускной способностью для поддержки огромных объемов данных, передаваемых между «тонкими» серверами и «толстыми» клиентами. Наконец, приложения, распределенные по нескольким десяткам и сотням

настольных машин, трудно поддерживать, обновлять и заменять.

Рисунок 17.6, *б* иллюстрирует вариант с «толстым» сервером. Этот подход почти не отличается от традиционного варианта вычислений на базе хоста и часто используется как промежуточный этап при переходе от мэйнфрейма к распределенному вычислительному окружению.

Трехзвенная архитектура клиент-сервер

Традиционная архитектура клиент-сервер состоит из двух уровней, или звеньев: клиентского и серверного. В последние годы все большее распространение получает трехзвенная архитектура клиент-сервер (рис. 17.7). В данной архитектуре прикладное программное обеспечение распределено между машинами трех типов: пользовательской машиной, промежуточным сервером и сервером базы данных. Машина пользователя представляет собой клиента, и в трехзвенной модели это, как правило, «тонкий» клиент. Промежуточные машины являются, по существу, шлюзами между «тонкими» клиентами и разнообразными серверами баз данных. Промежуточные машины должны уметь преобразовывать протоколы и отображать одни типы запросов к базам данных на другие. Вдобавок промежуточные машины должны объединять результаты запросов от различных источников данных. Наконец, эти машины должны играть роль шлюзов между настольными приложениями и оставшимися у организации со старых времен системами управления базами данных, таким образом, являясь посредниками между двумя «мирами». По существу, такая архитектура означает интеграцию приложений предприятия (Enterprise Application Integration, EAI), о которой рассказывалось в одноименном разделе главы 16.

Взаимодействие между промежуточным сервером и сервером баз данных также соответствует модели клиент-сервер. Таким образом, промежуточная система функционирует одновременно и как клиент, и как сервер.

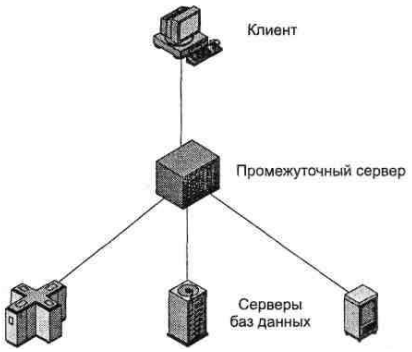

Рис. 17.7. Трехзвенная архитектура клиент-сервер

Промежуточное программное обеспечение

Процессы разработки и развертывания программных продуктов, относящихся к архитектуре клиент-сервер, значительно опередили процессы стандартизации всех аспектов распределенных вычислений от физического до прикладного уровня. Отсутствие стандартов затрудняет реализацию интегрированной конфигурации, состоящей из устройств различных производителей. Поскольку основные достоинства архитектуры клиент-сервер связаны с ее модульностью, а также с возможностью объединять различные платформы и приложения, необходимо решить вопросы совместной работы этих платформ и приложений.

Чтобы добиться максимальной эффективности применения архитектуры клиент-сервер, разработчики должны обладать набором инструментальных средств, предоставляющих единообразные средства и обеспечивающие единый стиль доступа к системным ресурсам на всех платформах. Это позволит программистам создавать приложения, которые не только одинаково выглядят на различных персональных

компьютерах и рабочих станциях, но и используют один и тот же метод доступа к данным, независимо от их расположения.

Наиболее общий подход к удовлетворению этих требований заключается в использовании стандартных интерфейсов и протоколов между приложением, с одной стороны, и коммуникационным программным обеспечением и операционной системой, с другой. Эти стандартизованные интерфейсы и протоколы получили название *промежуточного программного обеспечения*. Наличие стандартных программных интерфейсов облегчает развертывание одного и того же приложения на серверах и рабочих станциях разных типов. Это, очевидно, является достоинством не только для потребителя, но и для производителя. Это связано с тем, что потребители приобретают приложения, а не серверы. Потребители только выбирают серверные продукты, на которых работают нужные им приложения. Стандартизованные протоколы необходимы для взаимодействия различных серверных интерфейсов с клиентами.

Существуют разные пакеты промежуточного программного обеспечения от очень простых до очень сложных. Всех их объединяет способность скрывать от пользователя сложности и несоответствия различных сетевых протоколов и операционных систем. Производители клиентов и серверов, как правило, предлагают на выбор ряд популярных пакетов промежуточного программного обеспечения. Таким образом, пользователь может выбрать определенную стратегию развертывания промежуточного программного обеспечения, а затем заняться приобретением оборудования от разных производителей, поддерживающих данную стратегию.

Архитектура промежуточного программного обеспечения

На рис. 17.8 демонстрируется возможное применение промежуточного программного обеспечения в архитектуре клиент-сервер. Точная роль промежуточного программного обеспечения зависит от вычислительной модели клиент-сервер. Как было показано на рис. 17.6, существует несколько классов приложений клиент-сервер, зависящих от способа разделения функций приложения. В любом случае рис. 17.8 дает хорошее представление о данной архитектуре.

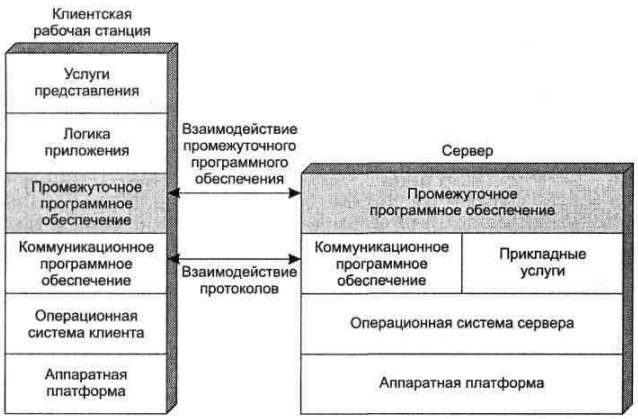

Рис. 17.8. Роль промежуточного программного обеспечения в архитектуре клиент-сервер

Обратите внимание, что промежуточное программное обеспечение состоит из двух компонентов — клиентского и серверного. Основное назначение промежуточного программного обеспечения заключается в обеспечении доступа приложения или пользователя к различным услугам, предоставляемым на серверах, причем так, чтобы не беспокоиться о различиях серверов. Стандартным средством доступа к реляционной базе данных как для локального, так и для удаленного пользователя или приложения является язык структурированных запросов (SQL). Однако многие производители реляционных баз данных, хотя и поддерживают язык SQL, добавили к нему свои собственные расширения. Это, с одной стороны, позволяет производителям различать свою продукцию, но, с другой

стороны, приводит к несовместимости.

Для примера рассмотрим распределенную систему, обслуживающую, помимо прочего, отдел кадров. Основные сведения о сотрудниках, такие как имена, адреса и т. д., могли бы храниться в базе данных от Gupta, тогда как информация о зарплате — в базе данных от Oracle. Когда пользователь в отделе кадров запрашивает доступ к определенным записям, он не должен думать, в которой из баз данных хранятся требуемые ему записи и кто является производителем этой базы данных. Единообразный доступ к этим системам должно предоставлять промежуточное программное обеспечение.

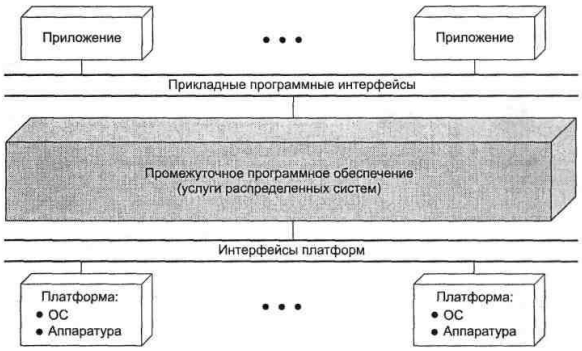

Рис. 17.9. Промежуточное программное обеспечение с точки зрения логики

На роль промежуточного программного обеспечения полезно взглянуть с точки зрения логики, а не реализации (рис. 17.9). Вся распределенная система может рассматриваться как набор приложений и ресурсов, доступных пользователю. Пользователям не нужно беспокоиться о расположении данных или приложений. Все приложения работают поверх унифицированного *прикладного программного интерфейса* (Application Programming Interface, API). За маршрутизацию запросов клиента к соответствующему серверу отвечает промежуточное программное обеспечение, присутствующее на всех клиентских и серверных платформах.

Пример применения промежуточного программного обеспечения для интеграции несовместимых продуктов иллюстрирует рис. 17.10. В этом случае промежуточное программное обеспечение используется для решения проблем несовместимости сетей и операционных систем. Сети DECnet, Novell и TCP/IP соединены магистральной сетью. Промежуточное программное обеспечение, работающее в каждой сети, гарантирует, что у всех сетевых пользователей будет прозрачный доступ к приложениям и ресурсам во всех трех сетях.

Несмотря на разнообразие предлагаемого на рынке промежуточного программного обеспечения, в основе всех этих продуктов лежит, как правило, один из трех механизмов: передачи сообщений, вызова удаленных процедур и объектно-ориентированный механизм. Мы рассмотрим эти три механизма последовательно.

Передача сообщений

Распределенную передачу сообщений, реализующую функциональность архитектуры клиент-сервер, иллюстрирует на рис. 17.11, *а*. Клиентский процесс запрашивает некую услугу (например, прочитать или распечатать на принтере файл). Для этого он посылает сообщение с запросом данной услуги серверному процессу. Серверный процесс принимает сообщение, обрабатывает запрос и посылает ответное сообщение. В простейшем случае требуются только две функции: Send (отправить) и Receive (принять). Функции Send на входе необходимо указать получателя, а также содержимое сообщения. Функции Receive нужно указать, от кого ожидается сообщение (включая вариант «все») и адрес буфера, в который следует поместить принятое сообщение.

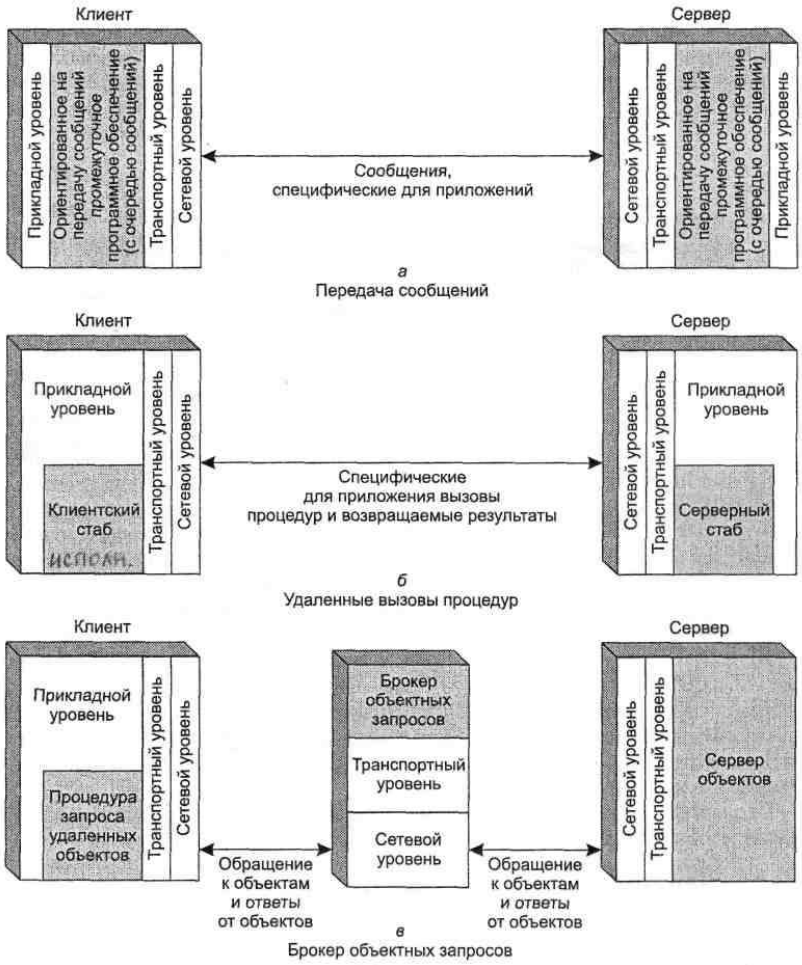

Рис. 17.11. Механизмы реализации промежуточного программного обеспечения

На рис. 17.12 показана схема возможной реализации механизма передачи сообщений. Процессы пользуются услугами модуля передачи сообщений. Запросы на эти услуги могут иметь вид примитивов и параметров. Примитив соответствует исполняемой функции, в параметры используются для передачи данных и контроля над информацией. Конкретная форма примитивов зависит от программного обеспечения передачи сообщений. Это может быть вызов процедуры или передача сообщения процессу операционной системы.

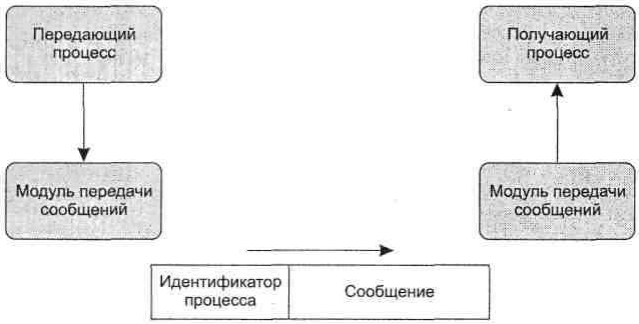

Рис. 17.12. Базовые примитивы передачи сообщений

Примитив Send используется процессом, желающим отправить сообщение. Входными параметрами этого примитива являются идентификатор получающего процесса и содержимое сообщения. Модуль передачи сообщений формирует блок данных, включающий эти два элемента. Этот блок данных посылается на машину, на которой работает получающий процесс, при помощи каких-либо сетевых протоколов, например, TCP/IP. Когда адресат получает блок данных, этот блок направляется модулю передачи сообщений. Этот модуль исследует поле идентификатора процесса и сохраняет сообщение

в буфере этого процесса.

В этом сценарии получающий процесс должен объявить о своем желании получать сообщения, выделить буфер для сообщений и информировать модуль передачи сообщений при помощи примитива Receive. Альтернативный метод не требует подобного объявления. Просто когда модуль передачи сообщений получает сообщение, он сигнализирует процессу, которому оно адресовано, а затем помещает сообщение в общий буфер.

Существует несколько вариантов реализации механизма распределенной передачи сообщений, о которых будет говориться далее.

Надежная передача сообщений

Надежной называется такая система передачи сообщений, которая гарантирует доставку сообщения, если такая доставка возможна. В подобной системе используется надежный транспортный протокол или сходная логика, а также должны поддерживаться контроль ошибок, квитирование, повторная передача и восстановление порядка следования сообщений. Поскольку доставка гарантирована, нет необходимости уведомлять передающий процесс о доставке сообщения. Тем не менее может быть полезно возвращать передающему процессу подтверждения, чтобы уведомить его о том, что сообщение доставлено. В любом случае, если системе не удается доставить сообщение (например, из-за неисправности сети или выходе из строя получателя), передающий процесс уведомляется о неудаче. Другую крайность представляет собой система передачи сообщений, которая просто посылает сообщения в компьютерную сеть, но не сообщает ни об успехе, ни о неудаче. Эта альтернатива позволяет значительно упростить систему передачи сообщений и накладные расходы на обработку и взаимодействие. Если приложению требуется подтверждение доставки сообщения, это может быть реализовано на прикладном уровне.

Синхронные и асинхронные системы передачи сообщений

При обращении к асинхронному (также называемому неблокирующим) примитиву, процесс не приостанавливается. Таким образом, после того как процесс вызывает примитив Send, операционная система возвращает процессу управление сразу после установки сообщения в очередь на передачу или после создания копии сообщения. Когда сообщение передано или скопировано в безопасное место для последующей передачи, передающий процесс прерывается и, таким образом, информируется о том, что буфером сообщения можно пользоваться снова. Если копии сообщения не создается, то любые изменения сообщения, производимые передающим процессом уже после обращения к примитиву Send, но до отправки сообщения, являются рискованными. Аналогично, после обращения к асинхронному примитиву Receive процесс продолжает работу. Когда сообщение прибывает, процесс информируется об этом событии путем прерывания или периодического опроса.

Асинхронные примитивы обеспечивают эффективное и гибкое обращение процессов к системе передачи сообщений. Недостаток этого подхода заключается в том, что программы, использующие подобные примитивы, трудно тестировать и отлаживать. События, которые зависят от времени и которые невозможно воспроизвести, могут стать источником трудноразрешимых проблем.

Альтернатива заключается в использовании синхронных или, как их еще называют, блокирующих примитивов. При вызове синхронного примитива Send управление не возвращается передающему процессу до тех пор, пока сообщение не будет передано (ненадежное обслуживание), или до тех пор, пока не будет получено подтверждение о доставке сообщения (надежное обслуживание). Блокирующий примитив Receive не возвратит управление, пока сообщение не окажется в выделенном для него буфере.

Уделенные вызовы процедур

Базовые сведения

Уделенный вызов процедуры (Remote Procedure Call, RPC) представляет собой вариант базовой модели передачи сообщения. Сегодня уделенные вызовы процедур являются общим и широко применяемым методом инкапсуляции взаимодействия в распределенной системе. Суть этой техники состоит в том, чтобы позволить программам на разных машинах взаимодействовать друг с другом путем простого вызова процедур, как если бы они работали на одной машине. Таким образом, механизм вызова процедур используется для доступа к услугам, предлагаемым удаленной машиной. Популярность этого подхода связана со следующими преимуществами.

- Вызов процедуры представляет собой широко распространенную и понятную абстракцию.
- Уделенные вызовы процедур позволяют специфицировать удаленный интерфейс в виде множества именованных операций с объектами указанных типов. Таким образом, интерфейс может быть четко и ясно документирован, а в распределенной программе можно выполнить статический контроль типов.
- Поскольку интерфейс стандартизован и точно определен, коммуникационная программа приложения может быть сгенерирована автоматически.
- Поскольку интерфейс стандартизован и точно определен, разработчики могут написать клиентский и серверный модули, для перемещения которых на другие платформы и операционные системы потребуется лишь небольшая модификация исходного текста программы.

Механизм удаленного вызова процедур может рассматриваться как усовершенствованная система надежной синхронной передачи сообщений. Общую архитектуру иллюстрирует рис. 17.11, б, а на рис. 17.13 показана более детальная схема. Вызывающая программа выполняет на своей машине обычный вызов процедуры с параметрами. Например:

CALL P(X. Y) Здесь

- ♦ P — имя процедуры;
- ♦ X — передаваемые аргументы;
- ♦ Y — возвращаемые значения.

То, что на самом деле происходит удаленный вызов процедуры на какой-то другой машине, может быть прозрачным или непрозрачным для пользователя. Так называемый исполнитель процедуры P, или стаб, должен быть включен в адресное пространство вызывающего процесса или динамически скомпонован во время вызова. Стаб создает сообщение, идентифицирующее вызываемую процедуру и содержащее ее параметры. Затем он посылает это сообщение удаленной системе и ждет ответа. Когда ответ получен, стаб возвращает управление вызвавшей ее программе и передает ей возвращаемые значения.

На удаленной машине с вызываемой процедурой ассоциируется другой стаб. Когда приходит сообщение, стаб исследует его и на основе полученных имени процедуры и параметров формирует обычное локальное обращение CALL P(X, Y). То есть удаленная процедура вызывается локально, при этом выполняется стандартная передача параметров через стек.

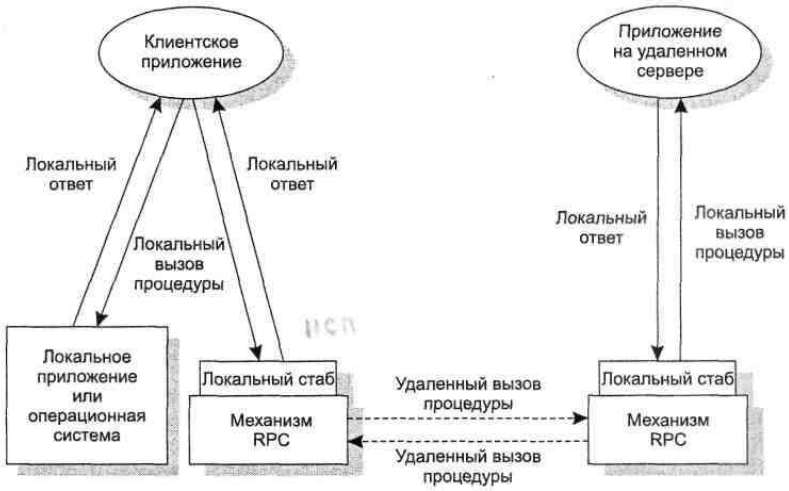

Рис. 17.13. Механизм удаленного вызова процедур

Привязка клиента и сервера

Привязка позволяет специфицировать отношения между удаленной процедурой и вызывающей ее программой. Привязка формируется после того, как два приложения установили логическое соединение и готовы обмениваться командами и данными.

Привязка может быть постоянной и непостоянной. При непостоянной привязке логическое соединение между двумя процессами устанавливается только на время удаленного вызова процедуры. Как только удаленная процедура возвращает значения, соединение разрывается. Активное соединение потребляет ресурсы, так как обе стороны должны хранить информацию о его состоянии. Использование временных соединений позволяет сберечь эти ресурсы. С другой стороны, на установку соединения требуется время и обмен служебными данными по сети, поэтому данный подход неприемлем для удаленных процедур, часто вызываемых одним и тем же процессом.

При постоянной привязке соединение, устанавливаемое для удаленного вызова процедуры, сохраняется и после того, как удаленная процедура возвращает управление. Это соединение может использоваться для последующих удаленных вызовов процедуры. Если в течение определенного интервала соединение не используется, оно разрывается. Такая схема удобна для приложений, делающих много удаленных вызовов процедур. В этом случае постоянная привязка позволяет делать вызовы и получать их результаты через одно и то же логическое соединение.

Объектно-ориентированные механизмы

После того как объектно-ориентированные технологии стали доминирующими в системном программировании, разработчики приложений клиент-сервер также начали применять этот подход. В этом случае объекты на клиентах и серверах обмениваются сообщениями. Взаимодействие между объектами может быть основано на обмене сообщениями, на удаленном вызове процедур или непосредственно на объектно-ориентированных возможностях операционной системы.

Клиент, которому требуется услуга, посылает запрос брокеру объектных запросов, действующему как каталог всех доступных в сети удаленных услуг (см. рис. 17.11, *в)*. Брокер вызывает соответствующий объект и передает ему необходимые данные. Затем удаленный объект обслуживает запрос и отвечает брокеру, который возвращает ответ клиенту.

Успех объектно-ориентированного подхода зависит от того, насколько хорошо стандартизирован объектный механизм. К сожалению, в этой области сосуществуют сразу несколько конкурирующих схем. Одной из них является модель COM (Component Object Model — модель компонентных объектов) компании Microsoft, являющаяся основой технологии OLE (Object Linking and Embedding — связывание и внедрение объектов).

Этот метод получил поддержку со стороны компании Digital Equipment Corporation, разработавшей механизм COM для операционной системы UNIX. С этим подходом конкурирует получившая широкую промышленную поддержку технология CORBA (Common Object Request Broker Architecture — обобщенная архитектура брокера объектных запросов), разработанная компанией Object Management Group. Архитектуру CORBA поддерживают компании IBM, Apple, Sun, а также многие другие.,

Интранет

Интранетом называют реализацию технологий Интернета во внутренней корпоративной сети. Появление этой концепции, еще несколько лет назад неизвестной, привело к самым быстрым за всю историю изменениям в области передачи данных в бизнесе. Сети интранет проникли в корпоративное сознание быстрее, чем персональные компьютеры, архитектура клиент-сервер или даже Интернет и Всемирная паутина, о чем свидетельствуют рекламные объявления производителей, планы потребителей, фактическое распространение этой технологии и книги на полках магазинов.

- ***ИНТРАНЕТ***
  Корпоративная составная сеть, предоставляющая ключевые приложения Интернета, в особенности Всемирную паутину (WWW). Интранет обслуживает внутренние потребности организации. Он может соединяться с Интернетом или существовать как изолированная составная сеть. Наиболее характерным примером интранета является корпоративная сеть, работающая по протоколам TCP/IP, имеющая собственный веб-сервер (или несколько вебсерверов) и предназначенная для обмена данными в пределах компании.

Росту популярности интранета способствовал длинный список привлекательных черт и достоинств этой технологии в области корпоративных вычислений, в том числе:

- быстрое создание прототипов и развертывание новых служб (время может измеряться днями или даже часами);
- простота масштабирования;
- обучения пользователей практически не требуется, а разработчики обучаются очень быстро, так как все службы и пользовательские интерфейсы знакомы по Интернету;
- возможность развертывания практически на всех платформах с полной поддержкой совместной работы;
- открытая архитектура означает большое и продолжающее расти количество приложений;
- поддержка широкого спектра распределенных компьютерных архитектур (несколько центральных серверов или множество распределенных серверов);
- поддержка оставшихся с прежних времен информационных ресурсов (баз данных, текстовых документов в различных форматах, хранилищ информации коллективного пользования);
- поддержка мультимедиа (аудио, видео, интерактивные приложения);
- для установки и наращивания инфраструктуры достаточно небольших капиталовложений.

Использование технологий, ориентированных на интранет, становится возможным благодаря высокой скорости обработки данных современных персональных компьютеров и высокой скорости передачи данных в локальных сетях.

Хотя термин *интранет* относится ко всему спектру приложений для Интернета, включая новости, gopher и FTP, своей популярности сети интранет обязаны, в первую очередь, технологии Всемирной паутины. Поэтому основная часть этого раздела

посвящена веб-системам, а о других приложениях интранета мы лишь кратко упомянем.

Всемирная паутина в интранете

В последние годы веб-браузер стал универсальным информационным интерфейсом. Все больше пользователей самых разных специальностей, познакомившихся в Интернете с Всемирной паутиной, комфортно себя чувствуют при работе с предлагаемой ею моделью доступа. Именно этот опыт позволяет широко использовать технологию Всемирной паутины в интранете.

Веб-страницы

Организация может использовать технологию Всемирной паутины в интранете для улучшения качества взаимодействия между администрацией и сотрудниками, а также для быстрого и удобного предоставления рабочей информации.

На рис. 17.14 показан пример структуры веб-страниц корпорации. Как правило, у корпорации имеется внутренняя домашняя страница, служащая точкой входа для сотрудников в корпоративную сеть. На домашней странице располагаются ссылки на другие страницы, посвященные разным темам, представляющим интерес всем сотрудникам или большим группам сотрудников.

Помимо этих услуг, технология Всемирной паутины идеально подходит для предоставления в интранете информации и услуг на уровне отделов и проектов. Группа может создать собственную веб-страницу для распространения информации и обработки проектных данных. Широкое распространение таких инструментальных средств, как Adobe Pagemill, облегчает создание собственных вебстраниц сотрудникам самых разных отделов.

Приложения баз данных

Несмотря на то, что Всемирная паутина представляет собой мощный и гибкий инструмент для обслуживания потребностей корпорации, язык HTML, используемый для создания веб-страниц, предлагает весьма ограниченный набор функций для обслуживания больших и изменяющихся баз данных. Для повышения эффективности интранета во многих организациях имеется потребность объединить услуги, предлагаемые технологией Всемирной паутины, с собственной системой управления базами данных.

На рис. 17.15 показана общая схема объединения технологий Всемирной паутины и баз данных. Сначала веб-браузер на клиентской машине генерирует запрос на информацию в форме обращения по URL-адресу. Это обращение запускает программу на веб-сервере, который посылает команду на сервер баз данных. Ответ сервера баз данных возвращается на веб-сервер, преобразуется в формат HTML и отправляется веб-браузеру.

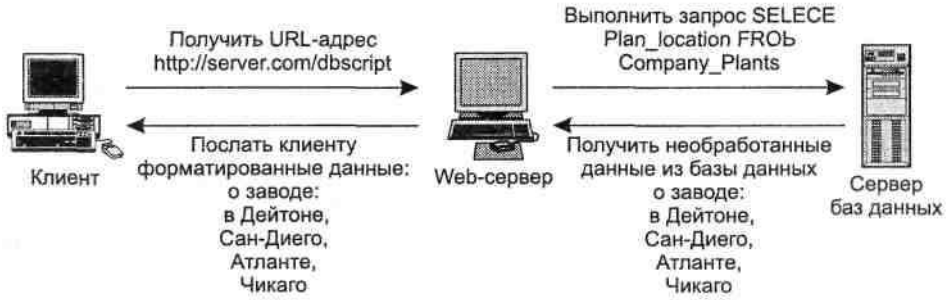

Рис. 17.15. Объединение технологий Всемирной паутины и баз данных

В [123] перечисляются некоторые преимущества интегрированной подобным образом системы по сравнению с традиционными методами доступа к базам данных.

♦ *Простота администрирования.* С сервером баз данных соединен только веб-сервер. Для добавления нового типа сервера баз данных не требуется изменять конфигурацию всех драйверов и интерфейсов на каждой клиентской машине. Достаточно наделить веб-сервер способностью преобразовывать HTML-запросы в команды к базе данных.

♦ *Установка.* Браузеры установлены практически на всех платформах, что избавляет разработчика от необходимости создания графических интерфейсов пользователя для разных машин и операционных систем. То есть разработчики могут считать, что у пользователя уже есть веб-браузер и он сумеет им воспользоваться, когда веб-сервер в интранете будет доступен. Таким образом, разработчику не придется заниматься установкой клиентской части программы и синхронизацией клиентской и серверной частей.

♦ *Скорость разработки.* При работе над традиционным приложением клиент-сервер очень много рабочего времени уходит на разработку клиентской части программы. К веб-проектам это неприменимо. Кроме того, текстовые теги языка HTML обеспечивают возможность быстрой модификации приложения на основе отзывов пользователей. В противоположность этому изменение формы или содержания типичного графического приложения представляет собой серьезную задачу.

♦ *Гибкость представления информации.* Гипермедийная основа Всемирной паутины позволяет разработчику приложений задействовать структуру, которая наилучшим образом подходит для данного приложения, включая применение иерархических форматов, когда пользователю доступны разные уровни детализации.

О недостатках данного подхода написано в [123].

♦ *Функциональность.* По сравнению с возможностями развитого графического интерфейса пользователя, типичный интерфейс веб-браузера является достаточно ограниченным. Например, с помощью языка HTML трудно генерировать формы с кнопками, текстовыми полями и меню, содержимое которых зависит от ввода пользователя.

4- *Операции без фиксации состояния.* Природа языка HTTP такова, что каждое взаимодействие между браузером и сервером представляет собой отдельную транзакцию, независимую от предыдущих и последующих операций. Как правило, между транзакциями веб-сервер не хранит пользовательской информации, хотя подобная информация о ходе взаимодействия с пользователем может быть важной. Например, рассмотрим приложение, позволяющее пользователю обращаться к базе данных, содержащей информацию о запчастях для автомобилей. После того как пользователь указал, что он ищет деталь для определенной марки автомобиля, последующие меню должны содержать названия запчастей только для данной модели. Добиться такой работы приложения, используя интерфейс, разработанный на языке HTML, можно, но весьма непросто.

Сравнение технологий интранет и клиент-сервер

Хотя традиционная архитектура клиент-сервер получила широкое распространение, почти вытеснив прежние компьютерные модели, с ее использованием связаны некоторые проблемы, включая:

- длительный цикл разработки;
- трудность разделения приложений на клиентский и серверный модули, а также проблемы модернизации этой схемы в ответ на отзывы пользователей;
- рассылка обновленных версий приложения клиентам требует определенных усилий;
- сложность масштабирования серверов в ответ на рост нагрузки в распределенном окружении;
- постоянно требуются все более мощные настольные компьютеры.

Большая часть этих проблем связана с природой традиционной модели клиент-сервер, в которой большая доля нагрузки ложится на клиента (стратегия «толстого» клиента, см. рис. 17.6, *в* и рис. 17.6, *г*). Как уже отмечалось, подобная модель плохо масштабируется, поэтому многие компании придерживаются модели «толстого» сервера, а интранет можно рассматривать как разновидность «толстого» сервера.

По сравнению с другими вариантами «толстого» сервера, интранет обладает такими преимуществами, как простота установки, использование небольшого количества популярных стандартов и интеграция с другими приложениями на базе стека протоколов TCP/IP. Однако маловероятно, чтобы интранет полностью или хотя бы частично вытеснил традиционные системы клиент-сервер, по крайней мере, в ближайшие годы. В долгосрочной перспективе интранет может занять доминирующие позиции в корпоративных информационных системах, но может стать и просто популярной альтернативой другим процветающим стратегиям клиент-сервер. Никто не может предсказать этого.

Экстранет

Концепция *экстранета* близка концепции интранета. Как и в интранете, в экстранете используются протоколы TCP/IP и основанные на них приложения, в частности, Всемирная паутина. Отличительной особенностью экстранета является то, что он предоставляет доступ к корпоративным ресурсам внешним клиентам, как правило, корпоративным поставщикам и потребителям. Этот внешний доступ может предоставляться через Интернет или другие коммуникационные сети, но его возможности гораздо шире, чем возможности простого веб-доступа, который сегодня предлагается практически всеми компаниями. Доступ к корпоративным ресурсам через экстранет, как правило, реализуется в соответствии с особой политикой безопасности. Как и в интранете, типичной моделью работы в экстранете является модель клиент-сервер.

- ЭКСТРАНЕТ
  Экспансия корпоративного интранета в Интернет, призванная предоставить избранным заказчикам и поставщикам компании, а также ее мобильным сотрудникам доступ через Всемирную паутину к частным данным и приложениям компании. В отличие от общедоступного веб-сайта компании, экстранет, как правило, предоставляет доступ в режиме реального времени через брандмауэр.

Существенная особенность экстранета состоит в том, что эта технология позволяет компаниям работать с информацией совместно. В [81] перечисляются некоторые достоинства подобного разделения информации.

♦ *Экономия средств.* Процесс формирования информации, к которой должен предоставляться коллективный доступ, отличается очень высоким уровнем автоматизации с минимумом бумаготворчества и участия человека.

♦ *Повышение ликвидности продуктов.* Потребители могут напрямую включаться в процесс проектирования продукта, просматривать спецификации проекта, пользоваться автоматизированными инструментальными средствами сбора пожеланий потребителей, а также другими средствами доступа к данным. Все эти функции помогают фирме определить оптимальное соотношение свойств продукта.

♦ *Более высокое качество продукта.* Жалобы потребителей быстрее достигают поставщиков, и их легче отслеживать, что позволяет быстрее вносить изменения в продукт.

♦ *Рост прибыли поставщиков.* Поминутно поступающая информация о том, что продается, а что нет, помогает поставщикам более четко реагировать на ситуацию на рынке.

♦ *Снижение материалыю-производствеиых запасов.* Управляемая потребителем система оперативной поставки комплектующих позволяет принимать более взвешенные решения по приобретению оборудования.

♦ *Сокращение сроков выхода па рынок.* Продукт достигает рынка быстрее, когда производители, разработчики, продавцы и потребители связаны «электронным партнерством».

В экстранете важное значение имеют вопросы безопасности. Поскольку доступ к

веб-ресурсам и базам данных корпорации предоставляется внешним пользователям, причем разрешается выполнение транзакций с этими ресурсами, необходимо решить проблемы конфиденциальности и аутентификации. Как правило, для этого используется виртуальная частная сеть. Эта тема будет обсуждаться в главе 20, здесь же мы просто перечислим некоторые коммуникационные ресурсы корпоративного интранета, к которым можно предоставить доступ внешним пользователям, создавая, таким образом, экстранет.

+ *Удаленный доступ по телефонным линиям*. Внешние пользователи получают непосредственный доступ в интранет, пройдя стандартную процедуру аутентификации (введя имя пользователя и пароль). Такой подход, возможно, обеспечивает самый низкий уровень безопасности, так как велик риск того, что один пользователь сможет выдать себя за другого, а набор средств противодействия подобным атакам весьма ограничен.

♦ *Безопасный доступ к иптранету из Интернета*. Аутентификация пользователей и шифрование данных, передаваемых между пользователем и интранетом, позволяют повысить уровень безопасности. Шифрование данных защищает от подслушивания, а аутентификация предотвращает несанкционированный доступ. Однако, как и в случае доступа по телефонной линии, если хакер сумеет обмануть систему аутентификации, он может получить доступ ко всем ресурсам корпоративной сети.

♦ *Доступ через Интернет к внешнему серверу, дублирующему некоторые данные интранета*. Такой подход снижает риск проникновения хакера, но также может снизить ценность экстранета для внешних партнеров, так как реального доступа в интранет они не получают.

♦ *Доступ через Интернет к внешнему серверу, генерирующему запросы к базам данных, хранящимся на внутренних серверах*. Внешний сервер играет роль брандмауэра, проводя в жизнь корпоративную политику безопасности. Брандмауэр может шифровать взаимодействие с внешними пользователями, аутентифицировать внешних пользователей и фильтровать поток информации, ограничивая доступ к ней внешних пользователей. Если сам брандмауэр надежно защищен от хакерских атак, такой подход является достаточно надежным.

♦ *Виртуальная частная сеть*. Виртуальная частная сеть представляет собой обобщение концепции брандмауэра. В ней используются встроенные механизмы безопасности протокола IP, которые обеспечивают безопасное взаимодействие между внешними пользователями и корпоративным ин-транетом. Виртуальные частные сети обсуждаются в главе 20.