АС "Спектр" в этих условиях предлагает очень гибкий подход, основанный на модульной архитектуре системы. Некоторые модули существуют в нескольких реализациях, настроенных на те или иные особенности, предприятие имеет возможность выбрать оптимальное решение, некоторые модули могут заменяться пакетами, разработанными на предприятии ранее, и, конечно, существует возможность интеграции АС "Спектр" с любыми уже действующими системами.

Следует отметить, что в настоящее время наметилась тенденция к глобализации бизнеса. Предприятия объединяются в холдинги, корпорации, концерны. При этом руководству холдинга сложно управлять предприятиями с разнородными уникальными производственными системами. Гораздо удобнее иметь корпоративный или отраслевой стандарт построения информационных систем, который регламентировал бы не только логику построения бизнес-процессов предприятия внутри холдинга, но и опирался на единый инструмент – АСУ предприятием. Тогда может идти речь о типовом программном решении для определенного бизнеса. Пока же на рынке существует множество решений, которые позволяют автоматизировать те или иные специфические задачи. В АС "Спектр" включены средства для объединения информации нескольких предприятий и принятия решений на уровне объединения.

Компания "Старт плюс" как разработчик системы, ориентированной на решение задач производства, предлагает на рынок несколько решений по организации производства. Каждое решение может быть дополнительно настроено и адаптировано под специфику конкретного предприятия. Программный продукт фирмы "Старт плюс" используется на предприятиях машиностроительного и приборостроительного профиля с дискретным типом производства. Существуют отдельные решения для опытного, сборочного и механического производств. Система постоянно совершенствуется и развивается по нескольким направлениям.

***Шумский Сергей Леонидович*** *– коммерческий директор компании "Старт плюс".*
*Контактный телефон (0855) 285-310.*

## Технологии создания прикладных программ для распределенных вычислений

**А.Ю. Молчанов (НПП "СпецТек")**

*Показано, что на рынке существуют системы программирования, поддерживающие две самых распространенных технологии создания прикладных программ, функционирующих в условиях распределенных вычислений. Это технология клиент-сервер, уже довольно давно завоевавшая место на рынке, и относительно новая технология создания многоуровневых (multi-tier) приложений. Приводятся особенности, преимущества и недостатки разработки ПО, а также принципы работы приложений в многоуровневой архитектуре.*

Организация распределенных вычислений предусматривает создание соответствующих технологий разработки прикладных программ (приложений), ориентированных на функционирование в ее рамках. Эти технологии должны поддерживаться разработчиками прикладных программ, производителями ОС и средств разработки. На сегодняшний день известны две самых распространенных технологии создания прикладных программ, функционирующих в условиях распределенных вычислений: клиент-серверная и многоуровневая. Материал, посвященный рассмотрению основных принципов, лежащих в основе технологии клиент-сервер, был опубликован в журнале "Автоматизация в промышленности" №1, 2005 г. Данная статья повествует о многоуровневой архитектуре.

### Многоуровневая архитектура: разработка программ и принципы разработки приложений

Трехуровневая, а затем и многоуровневая архитектуры разработки приложений явились логическим продолжением идей, заложенных в архитектуре клиент-сервер [1].

Главным недостатком архитектуры приложений типа клиент-сервер стал тот факт, что в клиентской части приложения совмещались два принципиально разных уровня обработки данных – пользовательского интерфейса и уровень бизнес-логики прикладной программы. Для выполнения этих функций к вычислительной системе должны предъявляться разные требования, и в том случае, когда на уровне бизнес-логики выполняется сложная обработка данных, эти требования со стороны клиентской части приложения могут быть непомерно велики. Кроме того, бизнес-логика приложения, как правило, незначительно изменяется по мере прохождения прикладной программой этапов ее жизненного цикла. В то же время, интерфейсная часть может серьезно видоизменяться и, в предельном случае, подстраиваться под требования конкретного заказчика.

Еще одним фактором, повлиявшим на дальнейшее развитие архитектуры клиент-сервер, стало распространение глобальных сетей и развитие сети Internet. Многие приложения стали нуждаться в предоставлении пользователю возможности доступа к СУБД посредством глобальной сети. Возможностей архитектуры клиент-сервер для этой цели недостаточно, поскольку при работе прикладной программы в глобальной сети клиентская часть может не иметь никаких вычислительных ресурсов, кроме программы навигации по сети (браузера, browser).

Поэтому дальнейшим развитием архитектуры клиент-сервер стало разделение клиентской части еще на две составляющие: сервер приложений или промежуточное ПО (ППО), реализующее прикладную бизнес-логику; и "тонкий клиент", обеспечивающий интерфейс и доступ пользователя к результатам обработки (далее "тонкий клиент" будем называть просто "клиентом" или клиентской частью в тех случаях, когда путаница с архитектурой клиент-сервер исключается). Серверная часть осталась без изменений, но теперь она должна обязательно называться сервер данных или сервер БД, а не просто сервер, чтобы не путать ее с сервером приложений.

Простейшая структура ПО, построенного на основе трехуровневой архитектуры, представлена на рис. 1.

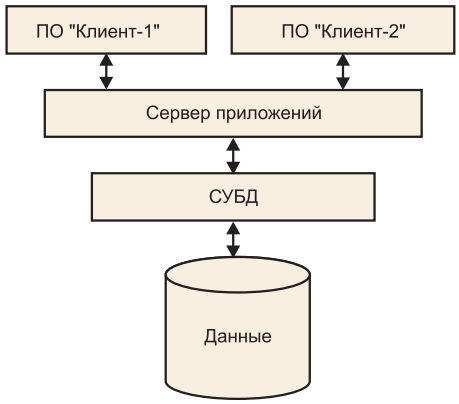


*Рис. 1. Структура приложения, построенного на основе трехуровневой архитектуры*

Таким образом, в состав ПО, построенного на основе трехуровневой архитектуры, входит три основных составляющих: сервер данных (database server); сервер приложений (application server или middleware); тонкий клиент (thin client").

Сервер данных выполняет все операции на уровне БД и на уровне файловых операций. Как и в архитектуре клиент-сервер, в качестве сервера данных в трехуровневой архитектуре чаще всего выступает соответствующая СУБД. Сервер приложений выполняет все функции, связанные с обработкой данных, а также формирует запросы к серверу данных и обеспечивает представление данных для клиентской части в той или иной форме. На клиентскую часть приходятся функции, связанные с отображением данных и организацией интерфейса пользователя.

В общем случае, ПО, построенное на основе трехуровневой архитектуре, может иметь более сложную структуру, чем показана на рис. 1. Например, сервер приложений может обмениваться данными не с одним, а с несколькими серверами данных, или клиентская часть может взаимодействовать с несколькими серверами приложений.

С другой стороны, по своим функциям сам сервер приложений может быть разделен на несколько уровней. В самом простейшем случае первый уровень будет взаимодействовать с сервером данных, второй уровень – выполнять обработку данных, а третий – формировать данные для представления их клиенту. В некоторых случаях такое усложнение системы оправдано. Тогда говорят не о трехуровневой, а о многоуровневой архитектуре ПО.

Несмотря на рассмотренные возможные усложнения трехуровневой архитектуры ПО, приводящие к многоуровневой архитектуре, принципы, лежащие в ее основе, остаются неизменными и могут быть рассмотрены в рамках трехуровневой архитектуры. Увеличение числа серверов данных или серверов приложений в каждом конкретном случае, а также разбиение самого сервера приложений на несколько уровней не вносит принципиальных изменений в организацию механизмов обмена данными между ними. Просто увеличивается число взаимодействующих приложений и могут усложняться протоколы обмена данными.

Если в архитектуре клиент-сервер разработчик прикладной программы создавал только код клиентской части, а в качестве сервера данных использовал, как правило, одну из имеющихся на рынке СУБД, то в трехуровневой архитектуре разработчик должен создавать две составляющих – сервер приложений и клиентскую часть, а также выбрать механизм обмена данными между ними и определить протокол обмена. Это является принципиальным отличием разработки программного обеспечения для трехуровневой архитектуры от архитектуры клиент-сервер.

Далее будут рассмотрены механизмы, лежащие в основе организации обмена данными между сервером приложений и клиентской частью.

## Технологии взаимодействия с сервером приложений

### Технология RPC (Remote Procedure Call)

Технология RPC (Remote Procedure Call, вызов удаленных процедур) возникла для обеспечения взаимодействия двух параллельно выполняющихся процессов в ОС типа UNIX задолго до появления многоуровневых архитектур ПО. Ее основы были заложены в саму архитектуру ОС UNIX [2].

Вызов удаленной процедуры заключается в том, что один из процессов ("процесс-клиент") запрашивает у другого процесса ("процесса-сервера") некоторую услугу (сервис) и не продолжает свое выполнение до тех пор, пока не получит результаты запрошенной услуги. Видно, что по смыслу такой механизм взаимодействия процессов напоминает вызов процедуры или функции в прикладной программе (отсюда и происходит название). Разница заключается в том, что выполнение вызова может происходить не только в рамках разных задач, но и на разных компьютерах.

Кроме того, механизм вызова RPC очень похож на обмен данными между клиентской частью и сервером данных в приложении, построенном на основе архитектуры клиент-сервер. Это не удивительно, поскольку в большинстве случаев ПО в архитектуре клиент-сервер также использует механизмы RPC, просто они остаются закрытыми для разработчика прикладной программы.

Реализация технологии RPC достаточно сложна, поскольку этот механизм должен обеспечить работу

взаимодействующих приложений, возможно, находящихся на разных компьютерах. Если при обращении к процедуре или функции в рамках одного приложения на одном компьютере передача данных выполняется через стек или общие области памяти, то в случае удаленного вызова передача параметров процедуре превращается в задачу синхронизации двух приложений и передачу запроса по сети. Соответственно получение результата также должно использовать сетевые механизмы взаимодействия двух приложений.

Также надо отметить, что одни и те же данные могут иметь разное представление в компьютерах с разной архитектурой. Поэтому еще одной задачей RPC является автоматическое обеспечение преобразования форматов данных при взаимодействии приложений, выполняющихся на разнородных компьютерах.

Удаленный вызов процедур включает следующие шаги:

- клиент осуществляет локальный вызов процедуры, называемой "заглушкой" (stub);
- процедура-заглушка принимает аргументы, преобразует их в стандартную форму и формирует сетевой запрос к серверу, упаковка аргументов и создание сетевого запроса называется "сборкой" (marshalling);
- сетевой запрос пересылается на удаленный компьютер, где соответствующий модуль RPC должен ожидать такой запрос;
- модуль RPC (который, как правило, входит в состав ОС) при получении запроса извлекает параметры вызова (unmarshalling), определяет тип сервера, которому предназначается вызов;
- модуль RPC на удаленном компьютере обращается к серверу, передает ему параметры вызова и ожидает получения результата;
- полученный результат преобразуется в стандартную форму, формируется сетевой запрос (marshalling) и передается процедуре-заглушке;
- процедура-заглушка извлекает результат вызова из полученного сетевого запроса (unmarshalling) и возвращает его клиенту, выполнение вызова RPC закончено.

В общем виде схема выполнения вызова RPC представлена на рис. 2.

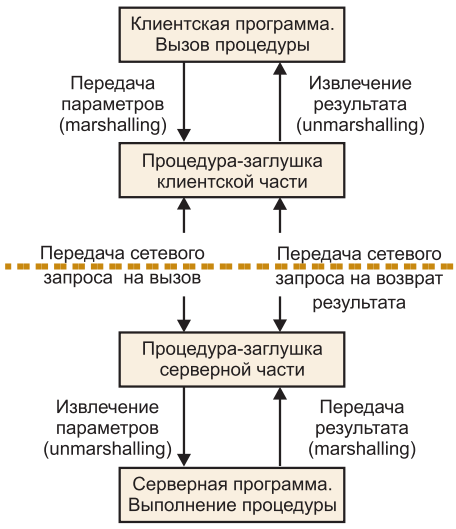


*Рис. 2. Выполнение вызова удаленной процедуры по технологии RPC*

Столь сложный механизм взаимодействия определяет тот факт, что реализация механизма обмена данными между клиентской частью ПО и сервером приложений не может быть полностью возложена на разработчика прикладной программы. Для этой цели должны использоваться функции ОС, а для организации обмена данными между приложениями, выполняющимися на компьютерах с различной архитектурой, должны быть разработаны соответствующие стандарты [2].

Использование RPC накладывает определенные ограничения на тип связи между приложениями. Дело в том, что в RPC применяется синхронный механизм взаимодействия: запрашивающее приложение выдает запрос и ждет ответа. На время ожидания приложение оказывается заблокированным. В связи с этим развертывать основанные на RPC приложения представляется целесообразным в локальных сетях, где время ответа обычно не очень велико. Кроме того, RPC является механизмом взаимодействия, ориентированным на соединение. В связи с этим RPC не может быть применена в глобальных сетях, так как вероятность недоступности приложения в этом случае достаточно велика.

### Технологии MOM (Message Oriented Middleware) и ORB (Object Request Broker)

С развитием ПО, построенного на основе трехуровневой архитектуры, на рынке стали появляться новые технологии организации взаимодействия клиентской части с сервером приложений.

Следующим шагом в разработке ПО для взаимодействия между активными приложениями стали системы передачи сообщений. Принцип их построения достаточно прост, но тем не менее они предоставляют огромные возможности по связыванию взаимодействующих программ.

В основе систем передачи сообщений лежит технология очередей сообщений: приложения обмениваются информацией не непосредственно друг с другом, а используя специальные буферы (очереди). В случае необходимости обмена данными программа пересылает их в принадлежащую ей очередь и продолжает функционирование. Доставку сообщения по назначению и его хранение обеспечивает специальное ПО – MOM (Message Oriented Middleware, ориентированное на сообщения промежуточное ПО). При этом MOM, как правило, может работать на разных программно-аппаратных платформах и с использованием различных сетевых протоколов. Мультиплатформность достигается за счет минимизации функций, выполняемых клиентской частью, а поддержка различных протоколов – за счет использования внутреннего протокола обмена информацией. Разработчику же предоставляется несложный и высокоуровневый API для работы с очередями сообщений.

Наличие очередей сообщений гарантирует доставку информации: в случае сбоев или отказов в сети, а также при отказе серверов будет обеспечено либо сохранение сообщения до восстановления соединения,

либо его повторная передача, или же будет произведен поиск нового пути в обход отказавшего участка.

Системы, построенные на базе MOM, похожи на системы электронной почты, но отличаются от последних тем, что обеспечивают взаимодействие между приложениями, а не между людьми. Соответственно MOM предназначена для передачи структурированной информации преимущественно технологического типа (т. е. порождаемой без участия пользователя), в то время как системы электронной почты осуществляют в большинстве случаев передачу неструктурированной информации.

Технология брокеров запросов к объектам (ORB, Object Request Broker) является наряду с MOM наиболее бурно развивающимся типом ПО для серверов приложений. ORB управляют обменом сообщениями в сети. Подобно обычному, "человеческому" брокеру, ORB выполняет функции интеллектуального посредника, т. е. принимает запросы от клиента (клиентского приложения), осуществляет поиск и активизацию удаленных объектов (серверов приложений), которые принципиально могут ответить на запрос, затем передает запрос одному из найденных серверов, получает ответ и передает его объектам запрашивающего приложения.

ORB, как и RPC и MOM, скрывает от пользователя процесс доступа к удаленным объектам. Запрашивающий клиент должен знать имя активизируемого объекта (сервера приложений) и передать ему некоторые параметры (как правило, это информация об интерфейсе вызываемого объекта – своего рода API для ORB). Интерес к ORB подогревается тем, что это ПО для серверов приложений поддерживает объектную модель, ставшую де-факто стандартом при разработке больших информационных систем [2].

### Организация серверов приложений

Столь сложная организация взаимодействия клиентской части с сервером приложений, с одной стороны, и большой интерес к разработке приложений для трехуровневой архитектуры, с другой стороны, привели к развитию в составе современных ОС соответствующих средств, которые обеспечивают выполнение обращений клиентской части к серверу приложений.

В настоящее время на рынке конкурируют стандарт CORBA и технология COM/DCOM корпорации Microsoft. Семейство стандартов COM/DCOM создано фирмой Microsoft для ОС семейства Microsoft Windows, а семейство стандартов CORBA (Common Object Request Broker Architecture) поддерживается специальным консорциумом, состоящим из широкого круга производителей ОС и систем программирования для большого спектра различных вычислительных архитектур. Этот последний стандарт (CORBA) претендует на то, чтобы обеспечивать переносимость серверов приложений и клиентских частей, соответствующих ему, между вычислительными системами различной архитектуры, в то время как семейство стандартов COM/DCOM ориентировано исключительно на вычислительные системы, построенные на базе ОС типа Windows (которые также производятся фирмой Microsoft).

Современные системы программирования обеспечивают поддержку разработки приложений в трехуровневой архитектуре. Средства поддержки тех или иных стандартов, необходимых для работы клиентской части и сервера приложений в трехуровневой архитектуре, сейчас присутствуют в составе многих систем программирования. Принципы их использования и распространения аналогичны принципам, применяемым для средств поддержки архитектуры типа клиент-сервер. Следует заметить, что существуют системы программирования, ориентированные на разработку либо клиентской части, либо сервера приложений; и в то же время, многие системы программирования стремятся предоставить разработчикам средства для создания обеих частей в трехуровневой архитектуре. Сервер БД в трехуровневой архитектуре, как и в архитектуре клиент-сервер, чаще всего является продуктом стороннего разработчика (как правило, производителя СУБД).

Используя описанные выше технологии и стандарты, разработчик может сам создать сервер приложений, а затем и клиентскую часть для своего программного продукта, разработать протоколы обмена данными между ними и создать соответствующие функции. Однако создание сервера приложений и разработка протоколов взаимодействия с ним – это достаточно сложный и трудоемкий процесс, содержащий много факторов, которые потенциально могут привести к появлению ошибок, которые сложно обнаружить. С другой стороны, в составе сервера приложений "значимую" часть, требующую интеллектуального труда разработчика, составляют только функции, непосредственно связанные с прикладной бизнес-логикой. Программный код, связанный с обеспечением обмена данными между сервером приложений и клиентской частью, как правило, требует кропотливого рутинного труда, который может быть автоматизирован.

На этот факт обратили внимание производители систем программирования, и на рынке средств разработки ПО появились программные продукты, носящие название "сервер приложений" [1]. Эти продукты зачастую входят в состав соответствующих систем программирования, но могут поставляться и отдельно от них. Такой сервер приложений представляет собой своеобразный контейнер, обеспечивающий разработчику взаимодействие с клиентской частью и с сервером данных по определенному стандарту, а также облегчающий написание программного кода, реализующего бизнес-логику. От разработчика требуется только создать код, отвечающий за специализированную обработку данных – все остальное большей частью обеспечит "контейнер" сервера приложений. Единственным ограничением при таком способе создания ПО для трехуровневой архитектуры является

тот факт, что созданный при этом полномасштабный сервер приложений должен поставляться заказчику в комплекте с программным кодом контейнера сервера приложений из состава системы программирования, а это чаще всего требует приобретения дополнительных лицензий от производителя системы программирования. Однако при таком пути достигается существенный выигрыш в объеме трудозатрат на создание сервера приложений для прикладной программы.

Хорошим примером такого рода "контейнера" сервера приложений может служить программный продукт Borland Enterprise Server (первоначально носивший название Borland Application Server) от известного производителя систем программирования – компании Borland (Inprise).

### Возможности многоуровневой архитектуры

По сравнению с архитектурой "клиент-сервер" трехуровневая и многоуровневая архитектуры создания приложений предоставляет разработчику ПО следующие преимущества:

- функции обработки данных возложены на сервер приложений, а на клиентскую часть приходятся только функции отображения данных и организации интерфейса пользователя, что существенно снижает требования к аппаратно-программному обеспечению клиентской части программного обеспечения;
- при необходимости изменить или дополнить логику обработки данных достаточно модифицировать только ПО на сервере приложений, а обновление клиентских приложений на всех рабочих местах необязательно;
- если необходимо изменить только внешний вид интерфейса пользователя, то для этого достаточно изменить простую по своей структуре клиентскую часть на тех рабочих местах, где это необходимо;
- для подключения к серверу данных (который, как правило, поставляется сторонним разработчиком) можно ограничиться одной лицензией (это не всегда справедливо, так как многие фирмы-производители СУБД учли это обстоятельство и требуют специальных лицензий при использовании их СУБД в многоуровневой архитектуре, либо же рассчитывают стоимость лицензии не от числа пользователей, а из параметров архитектуры сервера данных).

Главным недостатком трехуровневой архитектуры следует признать довольно сложный и трудоемкий процесс организации взаимодействия между клиентами и сервером приложений. Именно по этой причине далеко не все современные программные системы, построенные на основе архитектуры "клиент-сервер", быстро переходят на трехуровневую архитектуру, поскольку выделить в составе программного комплекса сервер приложений, описать его функции и организовать взаимодействие клиентских частей с ним – это сложный процесс, требующий усилий от разработчиков. В том случае, если функциональность ПО, построенного на основе архитектуры "клиент-сервер", достаточно стабильна, а обработка данных не содержит сложных ресурсоемких операций, переход на трехуровневую архитектуру не всегда оправдан (следует признать, что развитие аппаратных средств также способствует этому, поскольку появляются достаточно мощные и недорогие рабочие станции).

Кроме того, фирмы-производители ОС и системного ПО, обеспечивающего функционирование приложений в трехуровневой архитектуре, в настоящее время пока не заинтересованы в создании единого стандарта взаимодействия клиентской части с сервером приложений. Не следует ожидать, что в этой области в ближайшее время появится средство, которое сыграет роль, подобную роли языка SQL в архитектуре "клиент-сервер". Поэтому, выбирая тот или иной стандарт, разработчик может ограничить переносимость создаваемых прикладных программ и попасть в зависимость от производителя определенного системного ПО. Этот фактор, который является скорее организационным, чем техническим, также задерживает развитие приложений, построенных на базе трехуровневой архитектуры.

### Список литературы


1. *Молчанов А.Ю.* Системное программное обеспечение: Учебник для вузов / СПб.: Питер. 2003.
2. *Гордеев А.В.* Операционные системы: Учебник для вузов. 2-е изд. / СПб.: Питер. 2004.



***Молчанов Алексей Юрьевич*** *– канд. техн. наук, директор по разработкам НПП "СпецТек".*

*Контактный телефон (812) 329-45-60.*
*E-mail: mill@spectec.ru Http://www.trim.ru*