УДК 519.6

Влацкая И.В., Сормов С.И.
Оренбургский государственный университет
E-mail: mois@mail.osu.ru

УПРАВЛЕНИЕ И ОБРАБОТКА ИНФОРМАЦИИ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ

В настоящей статье рассмотрены распределенные системы информации, описаны принципы, концепции и технологии этих систем: связь, процессы, синхронизация, целостность и репликация, защита от сбоев и безопасность. Особое внимание уделено спецификации CORBA, а также World Wide Web, развитие которой и послужило толчком к резкому повышению интереса к распределенным системам.

Ключевые слова: распределенная система, Word Wide Web, СУБД, асинхронная связь, база данных, транзакция, обработка информации.

Распределенная система – набор независимых компьютеров, представляющихся их пользователям единой системой. Сокрытие различий между компьютерами и способов связи между ними, приложения единообразно работают в распределенных системах. Решение задачи облегчения доступа к удаленным ресурсам и контроля их совместного использования. Прозрачность (доступа к данным, сохранности данных, поломки системы или ее части, местоположения, смены местоположения, динамической смены местоположения, репликации, параллельного использования). Открытость (интерфейсы, способность к взаимодействию, гибкость, отделение описаний от определений) [1].

Полностью скрыть распределенность не удается. Открытость – использование синтаксических и семантических правил, основанных на стандартах. Правильный интерфейс обеспечивает возможность правильной совместной работы одного произвольного процесса, нуждающегося в интерфейсе, с другим произвольным процессом, представляющим интерфейс. Переносимость характеризует, насколько приложение, сделанное для одной системы, может работать в составе другой. Способность к взаимодействию характеризует, насколько две разные реализации системы в состоянии работать совместно.

Гибкость – это легкость конфигурирования системы, состоящей из различных компонентов, и легкость подключения новых компонентов. Масштабируемость по отношению к размеру (подключение дополнительных пользователей и ресурсов), к географическому положению (пользователи и ресурсы в пространстве), к административному устройству (управление в административно независимых организациях). Проблемы: узкие места по обслуживанию (один сервер для множества клиентов), по данным (один файл с общей информацией), по алгоритмам (централизованный алгоритм и перегрузка коммуникационной сети). Если переносимость не является целью, системы создают на языках ассемблеров. В таких системах минимизированы расходы на переключение контекста, а также на преобразования данных [2].

Двухъярусные архитектуры появились после возникновения ПЭВМ, на которых можно не только отображать информацию, но и обрабатывать ее, снимая часть нагрузки с других слоев. Преимущества перед одноярусными системами: отсутствие переключения контекста, переносимость. Проблемы: ограниченность связи со многими клиентами одновременно, которая требует переключать контекст, трудность работы с разными серверами, когда клиент начинает зависеть от нескольких систем, а изменения в него надо вносить при изменениях любого из серверов. За интеграцию начинает отвечать клиент, который должен комбинировать данные от нескольких серверов, отрабатывать все исключительные ситуации от всех своих серверов, координировать доступ ко всем серверам[2].

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

отдельной ЭВМ. Прикладной слой может быть распределен по разным компьютерам или кластерам. Прикладная логика более независима от управления ресурсами, переносимость и переиспользуемость возрастает. Управление ресурсами трехъярусной системы должно подчиняться четким интерфейсам, которыми должны пользоваться программы прикладной логики, находящиеся в промежуточном слое. Трехъярусные системы привели к стандартизации интерфейсов слоя управления ресурсами. Преимущества трехъярусной архитектуры проявляются при интеграции разнородных ресурсов [2].

Современные программы промежуточных слоев содержат функциональность, необходимую для введения в эти слои дополнительных свойств: транзакционных гарантий для различных видов ресурсов, балансировки загрузки оборудования, возможностей по регистрации событий, репликации, сохранности данных и многого другого. Стандартизованы глобальные свойства и интерфейсы между разными системными платформами на основе объектно ориентированного подхода (пример: спецификация CORBA) [2].

Основной характеристикой способа взаимодействия в распределенных системах является их синхронность или асинхронность. Блокирующим называется взаимодействие, если вовлеченные стороны прежде, чем переходить к следующим работам, должны дожидаться окончания взаимодействия. Параллельная работа фрагментов систем не имеет никакого отношения к асинхронности и деблокирующему взаимодействию. Сервер, получив запрос от одного из своих клиентов, может, обрабатывая запрос, работать параллельно другим своим клиентам, синхронность же относится к работе запрашивающего обслуживания клиента и того процесса в сервере, который этот запрос обрабатывает. Синхронное взаимодействие проще асинхронного. Сервер уверен, что состояние клиента не изменится. Программы обращения к серверу и обработки его ответа сильно связаны друг с другом и обычно размещаются рядом. Синхронное взаимодействие применяется в большинстве систем. Синхронное взаимодействие приводит к существенным потерям времени, а значит, и производительности. Поддерживать синхронность в разнородных системах трудно. Пример асинхронной связи – электронная почта. Письма накапливаются в почтовых ящиках, откуда получатель забирает их, когда хочет. Отправитель не ждет ответа. Это позволяет программе выполнять другие работы и делает ненужной координацию между сторонами взаимодействия. Асинхронная связь наиболее всего эффективна, если взаимодействие не имеет форму запрос/ответ. Взаимодействие может быть построено на регистрации событий и сигналов либо на модели публикация/подписка, когда одни компоненты извещают систему о постоянно доступной информации (публикуют), а другие – о своей потребности в ней (подписываются). Для асинхронного взаимодействия сообщения должны запоминаться в некотором промежуточном месте, откуда они впоследствии могут извлекаться сервером. Построенные так современные брокеры сообщений способны реализовывать сложные стратегии распространения, обрабатывая форматы или содержание сообщений, пока те находятся в очередях. Это важно для многоярусных систем, которые могут помещать обработчики сообщений в очереди, а не в адаптеры и не в сами компоненты, что дает возможность менять способы преобразования и доставки сообщений без изменения самих компонентов, генерирующих и получающих их. В системах сохранной связи сообщения не пропадают, если компьютер пользователя выключен, а хранятся в коммуникационной системе до тех пор, пока его не передадут получателю. Кроме сохранной связи существует связь без сохранения сообщений. В таких системах сообщения сохраняются только в период работы приложений, которые их отправляют и получают. На практике применяются различные комбинации взаимодействия. В случае сохранной асинхронной связи сообщение сохраняется в буфере либо локального прикладного комплекса, либо коммуникационного сервера. В случае сохранной синхронной связи сообщения хранятся только на принимающем прикладном комплексе. Отправитель блокируется на все время, пока сообщение не попадет в этот буфер. Усеченный вариант сохранной синхронной связи заключается в том, что блокировка отправителя осуществляется до доставки сообщения коммуникационному серверу получателя. Асинхронная несохранная связь характерна для транспортного протокола UDP. Если получатель в момент прихода сообщения на принимающий комплекс этого получателя не активен, передача обрывается. Другой пример этого вида связи – асинхронный удаленный вызов процедуры. Синхронная несохранная связь существует в нескольких вариантах. В самой сла-

бой форме, основанной на подтверждениях приема сообщений, отправитель блокируется до тех пор, пока сообщение не окажется в буфере принимающего комплекса. После получения подтверждения отправитель продолжает свою работу. Другая форма этой связи, ориентированная на доставку, предполагает, что отправитель должен быть заблокирован до момента доставки сообщения самому получателю, который продолжит свою часть работы по его обработке.

После возникновения объектно ориентированного подхода на базе систем удаленного вызова процедур появились брокеры объектов. Они более развиты, чем системы удаленного вызова, но, посуществу, от них мало отличаются. Наиболее известны брокеры объектов, построенные на основе подхода CORBA, предложенного группой OMG [1].

Большая часть функциональности брокеров объектов уже была реализована в транзакционных мониторах. Однако возникла необходимость перевести транзакционные мониторы, основанные на процедурных языках, на объектно ориентированные языки программирования. Эти факторы привели к сближению транзакционных мониторов и брокеров объектов, новые системы стали называться мониторами объектов.

Модель удаленного вызова процедур (RPC): клиент вызывает процедуру, работающую на сервере. Для клиента вызов не отличим от вызова локальной процедуры. При применении RPC синхронизация, установление связи, передача параметров и результата делаются скрытно от клиента (прозрачность). Сложности процесса: разные адресные пространства клиента и сервера, необходимость передачи параметров и результатов, необходимость обработки сбоев и отказов оборудования, передача параметров по значению и по ссылке [3].

Транзакции призваны защитить общие ресурсы от одновременного доступа. Транзакции превращают процессы доступа и модификации множества элементов, данных в одну операцию. Если в процессе выполнения транзакции будет определено, что дальнейшее ее выполнение невозможно (по любой причине), все данные восстанавливаются с теми значениями и в том состоянии, в котором они были до начала транзакции. Модель транзакции: процесс объявляет, что намерен начать транзакцию с одним или несколькими другими процессами, происходит согласование условий, выполняются операции, инициатор предлагает всем подтвердить, что работа завершена. Если подтверждение выдают все процессы-партнеры, результаты утверждаются и становятся постоянными. Если хотя бы один процесс-партнер отказывается выдать подтверждение, ситуация возвращается к тому состоянию, которое имело место до начала транзакции. Это свойство называется «все или ничего».

Распределенное подтверждение в распределенных системах реализуется при помощи координаторов. В простейшей схеме координатор запрашивает все остальные процессы, участвующие в транзакции, в состоянии ли они подтвердить запрашиваемую операцию, а те отвечают на запрос. Эта схема известна под названием однофазного подтверждения. Недостаток: если один из участников на самом деле не может осуществить операцию, он не в состоянии сообщить об этом координатору. Двухфазное подтверждение (Two-Phase Commit, 2PC) строится из двух фаз, каждая из которых включает в себя два шага. Первая фаза называется фазой голосования, вторая фаза есть фаза решения. Сначала координатор рассылает всем участникам запрос голосования, а участники возвращают координатору свои голоса «за» или «против» подтверждения своей части транзакции. Собрав ответы всех участников, координатор посылает всем участникам глобальное подтверждение. Если хотя бы один участник отказался подтвердить свою часть транзакции, координатор вместо подтверждения рассылает всем указание отмены транзакции. Если участник, голосовавший за подтверждение, получает подтверждение от координатора, он подтверждает транзакцию. В случае же получения отмены выполнение транзакции прекращается. Проблемы возникают, если при использовании протокола двухфазного подтверждения в системе возникают ошибки. Для исправления ситуации вводится механизм тайм-аута [2].

Отказоустойчивость алгоритма 2PC достигается ведением журналов состояний. Чтобы гарантировать, что процесс действительно восстановился, нужно, чтобы он сохранял свое состояние при помощи средств длительного хранения данных. Выбор содержания и момента времени записи в журнал являются очень важными с точки зрения производительности, поскольку выполняются для каждой транзакции и поскольку ведение журнала предполагает запись на диск (а это длительная операция). Большинство систем используют алгоритм 2PC: протокол 3PC более дорог и в практических ситуациях может оказаться блокирующим [3].

Трехфазное подтверждение необходимо сопровождать системами сборки мусора, иначе информация о многочисленных состояниях протокола слишком быстро разрастается. Многие системы соглашаются с риском блокировки подтверждения при возникновении ошибок, другие соглашаются со снижением непротиворечивости, которое возникает при сбоях в системе.

Для реализации транзакций применяются транзакционные мониторы, разработанные для обеспечения надежного мультиплексного доступа к большому количеству ресурсов для большого количества параллельных пользователей. Транзакционный монитор есть альтернативная операционная система. Облегченный вариант процесса, используемый транзакционными мониторами, работает по одной (разделяемой) программе и видит все данные, которые были видны в точке создания. Цель транзакционного монитора – поддержка выполнения распределенных транзакций на основе транзакционного RPC.

Функциональность транзакционных мониторов достаточна для разработки, выполнения, управления и сопровождения транзакционных распределенных систем. Эта функциональность включает в себя язык IDL, именование, безопасность и аутентификацию, компиляторы переходников, поддержку работы с транзакционными вызовами (транзакционные скобки, обратные вызовы), ведение журнальных записей, восстановление, блокировку, управление процессами и приоритетами, балансировку нагрузки, репликацию, управление ресурсами. Архитектура транзакционных мониторов включает клиентский интерфейсный компонент и поддержку прямого доступа через терминал.

Программный поток исполняет процедуры, написанные на языке данного монитора, которые содержат операции над именованными логическими ресурсами. Маршрутизатор сопоставляет операции и вызовы, которые могут относиться к ресурсам (например, БД) или локальным службам самого монитора. В состав маршрутизатора входит специализированная база данных, содержащая определения соответствий между именами логических ресурсов и физическими устройствами. При изменении системы администратор исправляет это соответствие: клиентское приложение модифицировать не требуется, клиент знает только логические имена. Взаимодействие с ресурсами осуществляется через менеджер взаимодействия.

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

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

та в адресное пространство клиента. Удаленное обращение к методам (RMI) в части маршалинга и передачи параметров напоминает RPC, описание интерфейсов объектов проводится на языке определения интерфейсов. Обращение к методу может быть статическим (интерфейсы известны при разработке) или динамическим (параметры собираются перед обращением к методу). При обращении к методам в качестве параметров используются ссылки на объекты, которые передаются по значению. Привязка к объекту выполняется, как только это понадобится [3].

Параметры обращений преобразуются в общее представление, не зависящее от языка программирования и ОС, а затем преобразуются назад в скелетоне еще до обращения к методам, зависящим от языка. Это позволяет менять логику реализации и язык клиентских и серверных программ. Другое проявление инкапсуляции – независимость от размещения объектов. Когда нужно обратиться к службе, клиент обращается к ядру системы, которое выдает ссылку на объект (непрозрачный идентификатор), то есть логическое имя объекта, приписываемое ему в момент создания. Ссылка действительна до момента, когда объект уничтожается, даже если в течение жизни объект меняет свое местоположение. Объекты могут не только менять свои физические адреса, они могут оказываться доступными посредством разных брокеров объектов. На основе модели RMI создано множество реализаций брокеров объектов, облегчивших создание объектно ориентированных распределенных приложений.

В настоящее время выработано несколько версий стандарта CORBA. Спецификацией не определяются ни языки программирования разрабатываемых объектно ориентированных приложений, ни операционные системы, в которых они должны работать. Другими известными примерами являются брокер объектов Distributed Component Object Model (DCOM) и появившийся позднее COM+, инкорпорированный в операционные системы фирмы Microsoft. Известны платформы. NET (Microsoft) и Java 2 Enterprise Edition (J2EE). Брокер объектов содержит базовые функции взаимодействия объектов. Службы предоставляют функции сохранности, управления жизненным циклом, безопасности и многие другие [2].

Стандарт CORBA 3 поддерживает IDL для Си, Си++, Java, Smalltalk, Ады, Кобола, Лиспа, PL/1 и других языков. Чтобы клиенты могли идентифицировать нужную им службу, в спецификации CORBA ссылки на сервисные объекты выдаются только службой именования и справочной службой. Одна проблема заключается в том, что для поиска службы клиентский объект должен понимать смысл свойств службы, что в свою очередь требует распространения знаний среди клиентов и поставщиков служб. Если клиент не был заранее реализован с возможностью взаимодействовать с конкретной службой, трудно придать ему возможность выяснять, какие операции может выполнять вновь обнаруженная служба, каков подлинный смысл ее параметров, в каком порядке надо обращаться к ним, чтобы добиться нужной функциональности.

Список использованной литературы:

1. Малюк, А.А. Введение в защиту информации в автоматизированных системах / А.А. Малюк, С.В. Пазизин, Н.С. Погожин. – М.: Горячая Линия – Телеком, 2001. – 148 с.
2. Конахович Г.Ф. Защита информации в телекоммуникационных системах / Конахович Г.Ф. – М.: МК-Пресс, 2005. – 288 с.
3. Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju. «Web Services. Concepts, Architectures and Applications». Springer-Verlag, 2004. – 340 p.

Влацкая Ирина Валерьевна, зав. кафедрой математическое обеспечение информационных систем Оренбургского государственного университета, канд. техн. наук, доцент, e-mail: mois@mail.osu.ru
Сормов Сергей Игоревич, ст. преподаватель кафедры математическое обеспечение информационных систем Оренбургского государственного университета, канд. техн. наук, e-mail: astralx@mail.ru
460018, г. Оренбург, пр-т Победы, 13, каб. 2131,

Vlatskaya I.V., Sormov S.I.
Management and processing of information in the distributed systems
In the present grant the distributed systems of the information are considered, principles, concepts and technologies of these systems: communication, processes, synchronisation, integrity and replication, protection against failures and safety are described. The special attention is given to specification of CORBA, and also World Wide Web which development was an incitement to sharp increase of the interest to the distributed systems.
Key words: distributed system, World Wide Web, SUBD, asynchronous connection, database, transaction, processing of information.