УДК 004.415.2.031.43

ПРИНЦИПЫ РЕАЛИЗАЦИИ РАСПРЕДЕЛЕННЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

В.М. Яковлев

ООО “Информационные программные системы”,
03680, Киев, ул. Боженко, 15.
Тел.: +380 44 200 8424; факс: +380 44 200 8566.
Victor.Yakovlev@iss.org.ua

Рассматриваются общие принципы разработки и функционирования систем передачи и обработки сообщений в реальном времени, проблемы, возникающие при разработке и сопровождении таких систем и возможные пути их решения. Предлагается подход к масштабированию таких систем, базирующийся на концепции агентной среды.

The common principles of development the real-time message-processing systems are considered. The approach to the implementation of the scalability, based on the concept of the environment and agents, is proposed.

Предметная область

В настоящей работе рассматриваются принципы реализации систем передачи и обработки сообщений в реальном времени. При этом реальное время понимается в смысле (т.н. soft real-time), т.е. когда наряду с корректностью функционирования требуемый результат должен быть получен в заданное («приемлемое») время [1]. Подобные системы обычно представляют собой сложные программно-технические комплексы, и применяются в области управления технологическими процессами, а также в корпоративных системах электронной торговли [2, 3].

Главной отличительной особенностью таких систем является, прежде всего, то, что каждая система создается как уникальный продукт для решения конкретных прикладных задач. Это, в свою очередь, обуславливает высокую стоимость проектирования, разработки и сопровождения.

Системы передачи и обработки сообщений в реальном времени имеют, как правило, закрытый код, поэтому любые доработки программного обеспечения в процессе его эксплуатации также обходятся весьма дорого.

Известные решения, призванные удешевить и ускорить разработку и модификацию подобных систем, условно можно разделить на две категории:

- системы автоматизации проектирования. Здесь речь идет, как правило, о программных инструментах, предназначенных для проектирования структур данных и их взаимосвязей, а также об инструментах облегчающих создание сценариев использования (use cases), которые затем применяются при тестировании системы. Заметим, что инструменты автоматизации проектирования алгоритмов высокого уровня, в частности UML-инструменты, применяются в основном лишь на начальных стадиях проектирования системы, что диктуется требованиями производительности конечного кода;
- разработка «линеек» совместимых продуктов и программных шлюзов, обеспечивающих совместимость с продуктами других производителей.

Настоящая работа описывает еще один подход к разработке систем передачи и обработки сообщений в реальном времени – использование концепции агентной среды. Данный подход, в сочетании с вышеперечисленными, может дать существенное ускорение и удешевление разработки и модификации программного обеспечения за счет стандартизации алгоритмов взаимодействия различных блоков системы. Чтобы обосновать перспективность данного подхода, вначале рассмотрим требования к системам передачи и обработки сообщений в реальном времени и типовую архитектуру таких систем.

Требования к системе

Системы передачи и обработки сообщений в реальном времени обычно относятся к категории non-stop, т.е. таких систем, которые должны функционировать постоянно. К подобным системам обычно предъявляют повышенные требования, касающиеся надежности функционирования, защиты от сбоев, поддержки целостности информации, дублирования (резервирования) компонент и т.п. – наряду с жесткими требованиями относительно производительности и времени реакции [4].

Безусловно, реализация большей части требований сильно зависит от используемых аппаратных средств и линий передачи информации. В настоящей работе рассматриваются только вопросы программной реализации в сочетании с предположением, что все аппаратные средства функционируют надлежащим образом.

Типовая архитектура

В состав систем передачи и обработки сообщений входят следующие основные компоненты

- подсистема хранения данных (СУБД);
- подсистема обработки сообщений в реальном времени;
- подсистемы информационных служб;
- подсистема приема/передачи данных;
- подсистема ведения протоколов;
- терминальные компоненты (устройства, рабочие места, терминалы), являющиеся источниками и потребителями сообщений.

Первые пять подсистем относятся к серверной части системы, а терминальные компоненты обычно являются клиентскими частями. Серверная часть системы поддерживает постоянные соединения с терминальными компонентами. Взаимодействие с терминальными компонентами обычно параллельное и асинхронное, т.е. данные могут быть переданы или посланы от/к терминальной компоненте в любой момент времени, независимо от работы других компонент. Кроме того, результаты обработки запроса, поступившего от одной терминальной компоненты, могут быть разосланы многим терминалам.

В качестве примера приведем схему архитектуры торгово-информационной системы Люблинской биржи (Словения), размещенную на официальном сайте http://www.ljse.si (Рис. 1). На этой схеме можно увидеть все вышеперечисленные компоненты (сервер BorzaNet представляет собой подсистему поддержки информационных служб), кроме подсистемы ведения протоколов, поскольку последняя не всегда выделяется как отдельная подсистема.

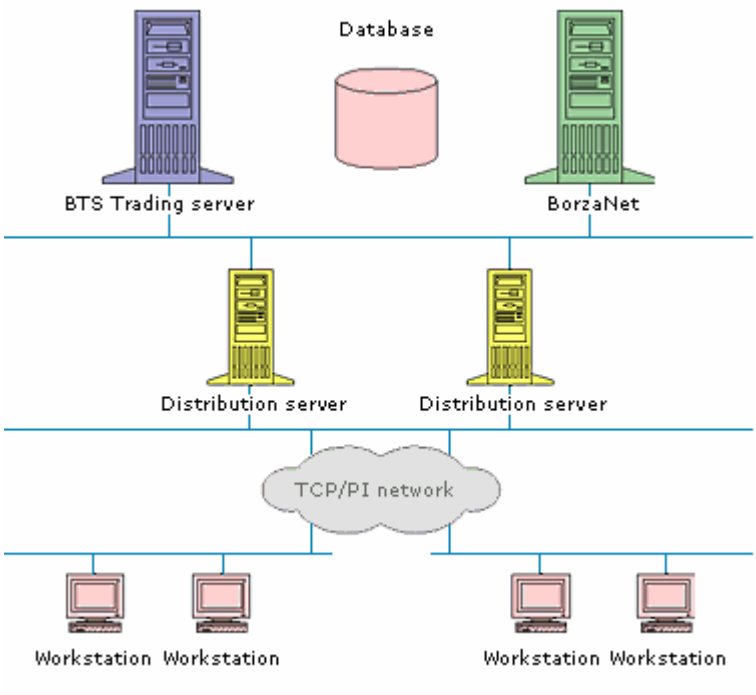

Рис. 1. Общая архитектура

Наличие постоянных соединений с терминалами и необходимость асинхронной (без дополнительных перезапросов) рассылки информации многим терминалам являются ключевыми факторами, отличающими системы передачи и обработки сообщений в реальном времени от классических клиент-серверных систем.

Очевидно, наиболее сложной и трудоемкой при реализации является серверная часть системы. Поэтому в дальнейшем мы ограничимся рассмотрением проблем, касающихся разработки серверной части.

Расширения архитектуры

Факторами, диктующими необходимость расширения архитектуры, являются:

- удаленность терминальных устройств;
- увеличение информационных потоков (возрастание количества передаваемых данных и интенсивности передачи);
- увеличение количества терминальных устройств.

Во всех случаях, связанных с загрузкой линий связи или временем передачи данных архитектура системы расширяется за счет увеличения количества компонент приема/передачи данных, а также организации промежуточных коммуникационных служб, оптимизирующих и/или концентрирующих информационные потоки.

При увеличении количества обрабатываемой информации расширение архитектуры достигается за счет распараллеливания обработки данных, т.е. включения в систему нескольких параллельно функционирующих компонент обработки данных. В этом случае, однако, приходится решать проблемы, связанные с синхронизацией доступа к данным и другим ресурсам [5, 6].

Проблемы разработки и модификации

Как уже отмечалось, имеются существенные отличия систем передачи и обработки сообщений в реальном времени от классических клиент-серверных систем. И, в отличие от последних, для реализации рассматриваемого класса систем не существует классических, проверенных практикой, стандартных решений. Достаточно отметить хотя бы тот факт, что необходимость поддержки постоянных соединений в сочетании с передачей определенных объемов информации по каждому из них требует определенной пропускной способности каналов связи. Таким образом, ошибка в оценке или настройке соответствующих технических параметров приведет, в конечном счете, к нарушению требований, предъявляемых к системе, независимо от качеств программного обеспечения.

Для компонент системы выбирается конкретная технология и разрабатывается прикладной протокол взаимодействия. Очевидно, что при успешной эксплуатации системы могут возникнуть вопросы ее дальнейшего расширения, дополнения функциональных возможностей и т.п. Поэтому важными вопросами, на которые требуется найти ответ в этом случае, являются:

- насколько масштабируемым является разработанный прикладной протокол, и как можно оценить стоимость его модификации в дальнейшем при необходимости увеличения числа компонент системы, или при размещении системы на множестве компьютеров;
- насколько выбранная технология связи компонент является масштабируемой, надежной, устойчивой и защищенной; как применение выбранной технологии может гарантировать соблюдение базовых требований к системе.

Далее, при проектировании системы должно учитываться базовое требование производительности. Необходимо рассмотреть, каким образом, и за счет чего это требование может быть удовлетворено.

Особое место занимают вопросы синхронизации потоков и/или процессов-компонент системы. При асинхронном поступлении запросов от терминальных устройств следует обратить внимание на очередность обработки этих запросов, в особенности, если такая обработка затрагивает одни и те же данные. Кроме того, внутренние события системы (например, связанные с таймером) также потенциально могут вызвать конфликт при обращении к данным.

Наличие постоянных соединений с терминалами и необходимость рассылки данных могут вызывать проблемы, связанные с неоднородностью функционирования каналов связи. При этом надо тщательно прорабатывать вопросы, связанные с утилизацией системных ресурсов – памяти и системных дескрипторов.

Наконец, любые модификации системы, находящейся в эксплуатации, должны в первую очередь рассматриваться с точки зрения неизменности базовой логики функционирования системы (синхронизации, технологий и протоколов взаимодействия) и удовлетворения базовых требований. Модификации, как правило, задумываются с целью изменения функциональности отдельных частей системы, и соответствующие изменения должны затрагивать только те ее части, которые отвечают за выполнение выбранных функций. Изменение технологии взаимодействия компонент, нарушение базовых требований ведут к необходимости перепроектирования системы в целом.

Несмотря на то, что для большинства перечисленных проблем существуют достаточно эффективные решения, мы, тем не менее, остановимся на некоторых из них более подробно.

Поддержка постоянных соединений с терминалами и рассылка данных. Укрупненная архитектура подсистемы коммуникаций показана на Рис. 2.

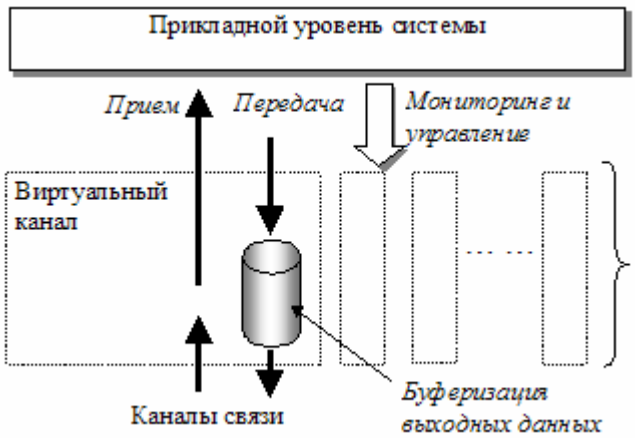

Рис. 2. Подсистема коммуникаций

Традиционно вопросы асинхронной связи решались путем организации отдельных процессов (потоков) для каждого канала связи. Более того, в операционных системах семейства UNIX данная задача решается только таким способом. Однако следует учитывать, что при большом количестве соединений (обычно порядка нескольких сотен) возникает эффект thread trashing, когда большая часть процессорного времени тратится на переключение контекста потоков, обслуживающих каналы связи. При этом быстродействие системы резко падает. Замедление передачи данных, в свою очередь, приводит к росту объема памяти, используемой для буферизации выходных данных. Это, в свою очередь, приводит к интенсивному использованию выгрузки на диск, что еще более замедляет работу системы. Такой процесс может развиваться лавинообразно до практически полной остановки всех выполняемых процессов. Далее следуют отказы в выделении памяти и аварийное завершение работы системы. [7]

Механизм защиты от подобных сбоев может предусматривать слежение за количеством соединений, ограничивая их некоторым разумным числом, а также за объемами памяти, используемой для буферизации выходных данных.

В операционных системах семейства Windows Server существует способ частичного решения проблемы thread trashing с помощью использования асинхронных коммуникаций и портов завершения ввода/вывода [8].

Вместе с тем, не существует универсальных способов разрешения ситуаций, связанных с неограниченным накоплением памяти при некорректном функционировании каналов связи или сетевого оборудования [9].

Синхронизация. При необходимости обработать множество одновременно поступивших в систему запросов следует решить две главные проблемы:

- очередность обработки запросов;
- возможность параллельной обработки.

Возможным решением обоих проблем сразу является организация отдельного потока обработки для каждого из поступивших запросов. При этом объекты данных, которые задействуются при обработке запросов, защищаются объектами синхронизации. Если для хранения указанных объектов используется одна из промышленных СУБД, то возможные конфликты доступа к объектам данных должных разрешаться на уровне этой СУБД, и необходимость в дополнительных объектах синхронизации отсутствует. При условии, что приоритеты запросов одинаковы, операционная система самостоятельно определяет очередность их обработки. Недостатками описанного подхода представляются, во-первых, возможность достижения эффекта thread trashing, а во-вторых – более низкая по сравнению с последовательной обработкой производительность, связанная с необходимостью переключать контекст потоков. Далее, если не использовать стандартные СУБД (во многих случаях это невыгодно с точки зрения затрат и производительности), то возникает проблема синхронизации потоков при обращении к данным. Поскольку зависимости объектов данных не всегда представляются очевидными, алгоритм определения возможности параллельной обработки (без синхронизации) может оказаться неоправданно затратным.

Другим решением является организация очереди запросов (Рис. 3).

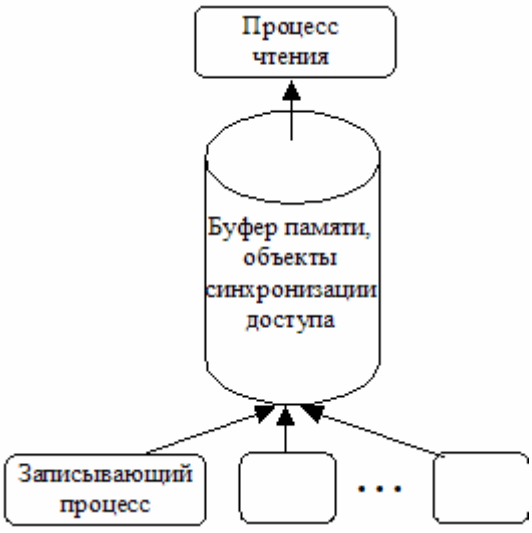

Рис. 3. Очередь

Очередь требует всего один объект синхронизации доступа, а также организацию отдельного потока чтения запросов из буфера. Однако при этом существует гарантия, что все запросы будут выполняться строго последовательно, а затраты на переключение контекста потоков, как в случае первого решения, сводятся к минимуму. Если, кроме того, предположить, что потоком чтения из очереди может служить главный поток выполнения процесса, и все без исключения события системы обрабатываются последовательно в этом потоке, то такое решение представляется весьма эффективным.

Шаблон архитектуры

Резюмируя все вышеизложенное, можно сформулировать укрупненную архитектуру системы в терминах программных компонент. Основных компонент всего четыре (Рис. 4): подсистемы коммуникаций, обработки (процессинга), хранения данных, а также очередь запросов.

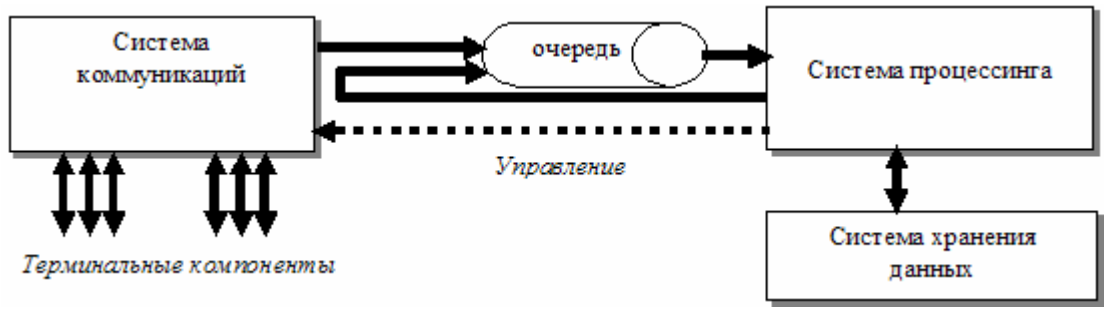

Рис. 4. Шаблон архитектуры

В данный момент несущественно, как реализуются перечисленные компоненты – в виде программных блоков в рамках одного процесса, либо в виде отдельных процессов, работающих на различных компьютерах. Вопросы конкретной реализации следует рассматривать при обсуждении конкретных нагрузок и параметров производительности системы.

Вопросы масштабирования

Проблемы, связанные с масштабированием системы, до сих пор не были упомянуты. Масштабирование выполняется с целью распределения нагрузок на отдельные компоненты системы. При этом система представляет собой множество процессов, распределенных в вычислительной сети и взаимодействующих друг с другом. В частности, при необходимости поддержки постоянных соединений с терминалами и рассылки данных при увеличении числа терминалов актуальной становится задача масштабирования подсистем приема/передачи данных. В результате некоторые компоненты системы должны дублироваться, а нагрузки должны распространяться между ними. При этом возникает проблема арбитража: при наличии n компонент, реализующих задачу T, необходимо выбрать, какой из этих компонент передать запрос. Отметим, что принятие такого решения не должна быть слишком ресурсоемким или продолжительным, поскольку следует учитывать требование производительности.

Заметим, что аналогичные задачи приходится решать и при реализации «горячего» резервирования компонент, т.е. при включении в систему компонент, функционирующих в режиме дублирования, и переключающихся к активному функционированию при выходе из строя активных компонент системы.

В данной работе предлагается подход к разработке распределенных систем, основанный на концепции агентной среды [10, 11]. Идея заключается в том, что любая компонента системы, выполняющая некоторое множество прикладных задач, может рассматриваться как агент.

Таким образом, система состоит из программных модулей, генерирующих и обрабатывающих сообщения (запросы), которые рассматриваются как документы, а циклы обработки сообщений – как транзакции.

Совместная работа модулей обеспечивается распределенной средой (набором системообразующих компонент).

Среда поддерживает спецификацию обмена сигналами UML([12], [13], [14]) – очереди сигналов, приоритеты, RPC.

Реализация агентной среды. Среда может быть определена как

$Env = (\{P\}, \{A\})$ где $\{P\}$ – множество атрибутов среды, $\{A\}$ – множество активных (погруженных в среду) агентов.

Среда реализуется как множество программных служб (Е-служб), функционирующих на одном или более компьютерах в локальной сети. Требование локальной сети выдвигается из соображений производительности. Кроме того, на одном компьютере не может функционировать более одной Е-службы.

При наличии двух или более Е-служб в распределенной системе, Е-службы образуют одноранговую сеть (кольцо). Все активные Е-службы разделяются по приоритетам. В системе не должно существовать более двух и более Е-служб с одинаковым приоритетом.

Агенты, исполняющие прикладные функции, будем также называть П-службами.

Среда обеспечивает единый интерфейс для подключения агентов и обмена сигналами (сообщениями, запросами).

Атрибуты среды. Атрибуты среды реализуются как типизированные именованные значения:

$$P = (N, t, v, \{AR\}),$$

где N – имя атрибута, t – тип, а v – значение. Множество $\{AR\}$ представляет собой права доступа к данному атрибуту для агентов заданных типов. Агенты могут получать информацию об имеющихся в среде атрибутах, создавать новые атрибуты и (при наличии соответствующих прав) читать и изменять значения атрибутов.

Агенты и типы агентов. Агенты реализуются как программные службы, реализующие прикладные функции (П-службы). Агенты являются экземплярами типов агентов, которые описываются как

$$AT = (N, \{AP\}, F_{init}, F_{clr}, \{F_{app}\})$$

где N – имя типа агента, $\{AP\}$ – множество атрибутов агента, F_{init} – функция инициализации агента при погружении в среду, F_{clr} – функция «очистки» при выгрузке агента, $\{F_{app}\}$ – множество прикладных функций, исполняемых агентами данного типа.

В процессе обработки сигналов (запросов) агент может генерировать новые сигналы и с помощью среды рассылать их тремя различными способами: одному определенному агенту, всем активным агентам заданного типа или всем активным агентам.

Принцип функционирования. Е-службы, образующие агентную среду, играют двойную роль: они выполняют диспетчеризацию сигналов, которыми обмениваются агенты, и одновременно служат арбитрами при выполнении функций резервирования и распараллеливания прикладных задач. Е-службы создают и постоянно поддерживают «карту» системы, которая включает информацию о расположении (пути доступа) и состоянии всех агентов.

Среда может использоваться также для получения оперативной информации о состоянии системы и управления системой.

Заключение

Изложенные в данной работе принципы реализации систем передачи и обработки сообщений в реальном времени являются общими для всех систем такого класса. Следует, однако, остановиться на выше описанном принципе масштабирования систем. Легко заметить, что приведенное решение может быть реализовано не только в распределенной системе, но и в рамках одного процесса без потерь производительности. Кроме преимуществ, предоставляемых возможностями масштабирования системы и резервирования ее компонент, другими очевидными преимуществами такого подхода являются:

- прозрачность и неизменность архитектуры системы: общая архитектура и базовая технология взаимодействия различных модулей не изменяются при любых модификациях системы;
- простота замены рабочих модулей: обновление (доработки) существующих модулей в процессе эксплуатации, а также и добавление новых модулей в систему осуществляется относительно простыми средствами, и, при определенных условиях, без необходимости новой сборки системы;
- гибкость: система легко изменяет состав прикладных модулей, настраивается на конфигурацию локальной сети и т.д.;
- контролируемость и управляемость: любые изменения конфигурации системы не влияют на возможности оперативного контроля и управления.

1. Толковый словарь по вычислительным системам // Под ред. В. Илленгуорта и др.: пер. с англ.: – М.: Машиностроение, 1989.
2. *Яковлев В.М.* О реализации функции хранения, гарантированной доставки и репликации данных торгово-информационных систем реального времени // Компьютерная математика. – 2002. – № 2. – С. 132 - 140.
3. *Яковлев В.М.* Принципы и задачи компьютеризации деятельности субъектов фондового рынка // Компьютерная математика. – К., 2001 – № 1. – С. 158 – 167.
4. *Парасюк И.Н., Яковлев В.М.* Информационные технологии на фондовом рынке: основы и архитектура // Проблемы программирования. – 2002. – № 1-2. – С. 333–340.
5. *Рихтер Дж., Кларк Джейсон Д.* Программирование серверных приложений для Microsoft Windows 2000 // Пер. с англ., Санкт-Петербург: Русс. ред., Питер, 2001. – 566 с.
6. *Рихтер Дж.* Windows для профессионалов // Пер. с англ. – М.: Microsoft Corp., 1997. – 679 с.
7. *Donald W. Gillies, Jane W.-S. Liu:* Greed in Resource Scheduling // IEEE Real-Time Systems Symposium. – 1989. – P. 285 – 294.
8. *Anthony Jones, Jim Ohlund.* Network Programming for Microsoft Windows, 2nd ed. – Microsoft Press, – 2002 – 580 p.
9. *Anthony Jones, Amol Deshpande.* Windows Sockets 2.0: Write Scalable Winsock Apps Using Completion Ports // MSDN Magazine, October, 2000, http://msdn.microsoft.com/msdnmag/issues/1000/winsock/
10. *A.Letichevsky, D.Gilbert,* A Model for Interaction of Agents and Environments, In D.Bert, C.Choppy, P.Moses (Eds), Resent Trends in Algebraic Development Techniques, LNCS 1827, 311-328, 1999.
11. *Letichevsky A.A., Kapitonova J.K., Letichevsky A.A. jr.* Insertion Programming and System Simulation // Proc. XXVI International Workshop on Modeling of Developing Systems, Kiev, Ukraine – 2003. – P. 19–20.
12. *Coulouris G., Dollimore J., Kindberg T.* Distributed Systems: Concepts and Design, Edition 3 // Addison-Wesley. – 2001. – 779 p.
13. *Гома Х.* UML. Проектирование систем реального времени, параллельных и распределенных приложений. – М.: ДМК Пресс, 2002. – 704 с.
14. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя: Пер. с англ. – М.: ДМК, 2002. – 432 с.