УДК 004.04

РЕАЛИЗАЦИЯ ОДНОПОТОЧНОГО КОММУНИКАЦИОННОГО ПО

Дагаев Дмитрий Викторович, Директор по АСУТП, ОАО “ДЖЭТ”, Россия, Москва,
dvdagaev@mail.ru

Введение

При модернизации распределенной системы обработки данных АСУТП АЭС была поставлена задача замены коммуникационного ПО. Рассматривались действующее ПО на основе CORBA [1], реализующее клиент-серверное взаимодействие, технология DDS [2] на основе модели публикация-подписка и новая разработка ПО для этих задач. При этом нужно было повысить качество сетевого взаимодействия.

Действующая система была реализована в виде многопоточного пользовательского ПО с блокировками потоков до получения данных. При сбоях связи и зависаниях сети система штатно переходила на резервированные сети, с выдерживанием задержек, но время от времени приходилось сталкиваться с присущими CORBA проблемами [3]: "ошибками, относящимися к надежности потоков управления, надежности исключительных ситуаций и управлению памятью". Опыт эксплуатации показывал не всегда правильное функционирование при сложных условиях связи, ошибках в сети и переходах на резервные маршруты передачи данных. Кроме того, архитектурно сложно выглядели системы, требующие модели публикации-подписки.

Второй подход, в свою очередь, не слишком хорошо решает не присущие DDS клиент-серверные задачи. Предлагаемый [2] протокол RTPS использует UDP как транспортный уровень и переносит механизмы гарантированной доставки на пользовательский уровень. Это не первый и не последний способ реализации функциональности TCP в пространстве пользователя. Является ли это решение не худшим, чем транспортная реализация TCP, - это вопрос, требующий тщательного рассмотрения.

С точки зрения приложения важен еще момент буферизации данных, которые получены по UDP. DDS предоставляет многопоточные сетевые сервисы для этих целей, которые добавляют еще один уровень сложности.

Учитывая вышесказанное, было принято решение разработки коммуникационного ПО TA (англ. Transparent Architecture), призванного совместить клиент-серверную модель и модель публикации-подписки. Технология Transparent Architecture построена на основном временном цикле одного потока, который осуществляет прием и передачу данных без единой блокировки. Это, кстати, дает дополнительное преимущество, т.к. "все переключения контекста при получении каждого пакета сильно расходуют время центрального процессора и существенно снижают производительность сети" [4] .

1. Основные принципы реализации

Неблокируемость и немодальность – ключевые особенности Transparent Architecture. Клиентские программы в клиент-серверных приложениях часто построены по принципу выделения отдельного потока для приема данных от сервера. Типичными особенностями такой схемы является модальность, заключающаяся в переходе в режим ожидание чтения данных с таймаутом. Поток при этом блокируется, при обрыве связи происходят переходы от одного таймаута к другому.

Транспортный уровень построен на неблокирующих способах взаимодействия (в частности, сокетах). Прием данных осуществляется при наличии данных без ожидания, передача – без ожидания завершения, которое будет зафиксировано позднее. Главный цикл обеспечивает точность поддержания времени в соответствии с требованиями задачи и возможностями операционной системы.

Для Оберон-версии управление транспортным уровнем задается абстрактным типом Channel.

```
Channel* = POINTER TO ABSTRACT RECORD END;
```

Конкретные реализации TCPChannel, UDPChannel, MUDPChannel, MSHMChannel, ICMPChannel подключаются в зависимости от конфигурации с помощью паттерна "Абстрактная фабрика" [5]. Преимуществом объектно-ориентированного подхода является то, что любой из существующих протоколов или вновь разработанный может быть введен независимо от базовой части системы.

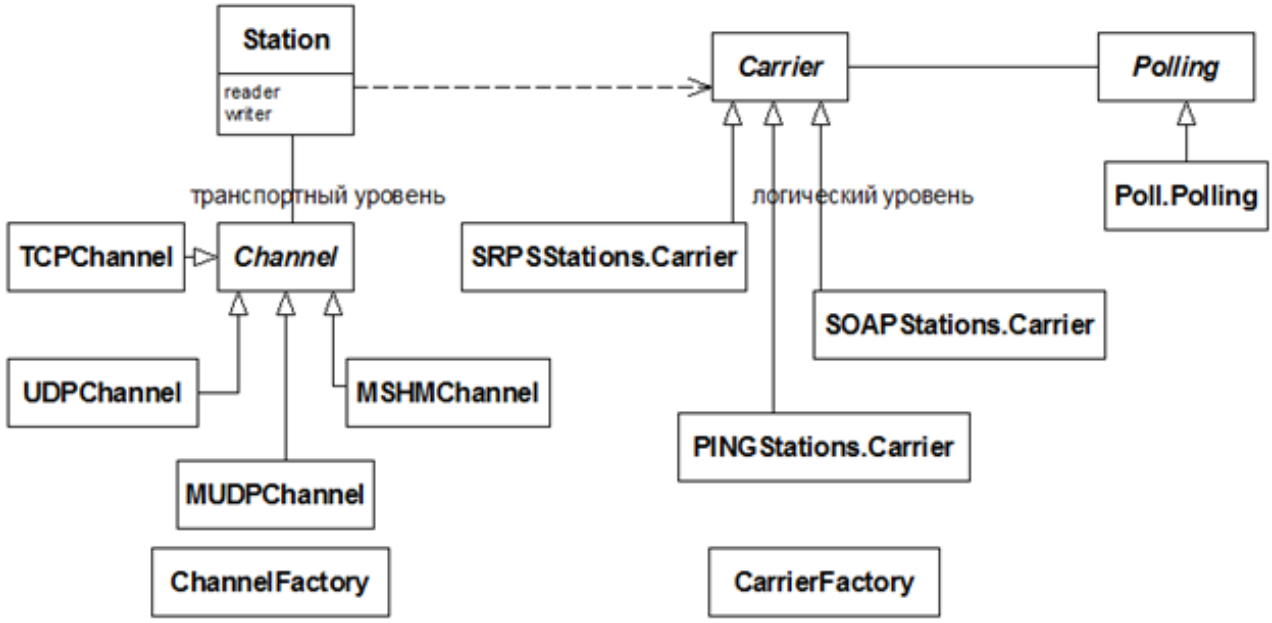


Рис. 1 – Иерархии типов транспортного и логического уровней

Transparent Architecture предлагает гибридный подход, меняя для разных задач транспортный уровень протокола:

- TCP – для клиент-серверных приложений;
- multicast UDP (MUDP) - для приложений публикации-подписки.

Аналогично задается логический уровень, т.е. формат данных протокола, при этом реализация SRPSStations формирует данные бинарного формата, SOAPStations формирует данные XML формата, а PINGStations формирует структуру ICMP пакета.

Объект типа Station играет роль паттерна "мост" в терминологии [5], позволяя разделить иерархии классов Channel и Carrier. Таким образом, транспортный и логический уровни протокола обмена могут развиваться и использоваться независимо друг от друга.

Паттерн стратегия из [5] используется для задания алгоритмов поллинга. Это сделано для оптимизации быстродействия системы в будущем: планируется ставить более эффективные стратегии, как epoll для Linux и Connection Points для Windows. Преимущество ОО-подхода здесь - это гибкость вплоть до управления базовыми алгоритмами.

2. Организация уровня данных

При проектировании было решено отказаться от блокирующих вызовов удаленных функций через прокси. Типичное решение для CORBA-клиентов вызов

```
Fun(x_in, y_out);
```

превращается в пару запрос-ответ. При этом единицей обмена становится не набор параметров функций, а сами параметры, представляющие собой структуры любой сложности.

Обмен данными происходит через специальные объекты t: Table, закрепленные за каждым типом данных. Это позволило использовать механизмы, при котором создаются цепочки программ, выполняющиеся в строгом порядке по мере возникновения некоторых сетевых событий. Для Клиент-Серверных приложений примером может служить цепочка:

```
v.val := “grep –i –r byte_order /usr/include/*”;
```

```
t.Request(v, TRUE, ProcessResults, NIL);
t.Call(NextRoutine, NIL);
```

которая создает запрос поиска файлов по шаблону, вызывает его на удаленном сервере, по событию – ответу на запрос – читает данные в переменную stringsList и вызывает программу обработки результата ProcessResults, после чего управление переходит к следующей программе NextRoutine. Все операции выполняются в строгой последовательности без блокировки основного потока. Ошибки разрывают цепочку выполнения, вызывая соответствующие обработчики.

Для обмена по подписке примерная цепочка такая:

```
v.val := "(value < 23.3) OR (value > 76.3)";
t.Write(v, TRUE);
t.Subscribe(TRUE, OnNextValue, NIL);
```

она уже выполняет многократно следующую за ней операцию OnNextValue при каждом ответе от удаленного узла, до тех пор, пока не сменится данная подписка, т.е. до вызова следующего t.Write.

Также было принято решение отказаться от IDL-компилятора, а использовать RTTI-информацию в тех реализациях, где она имеется. Для языков программирования, дающих доступ к информации времени выполнения (RTTI – RunTime Type Information), таких как BlackBox Оберон, используется RTTI. Для языков, не дающих подобного доступа, как C, используются специальные макросы, извлекающие эту информацию из текста определений переменных. В реализации Оберон-2 XDS информация, предоставляемая системой, является неполной, ее пришлось также расширять.

Протокол SRPS рассчитан на передачу данных сложных типов, включая составные структуры, массивы и динамические массивы, реализуемые через указатели в языках программирования. Динамические массивы могут содержать структуры, элементы которых являются динамическими массивами.

Информация о типах служит для сериализации пользовательских данных в сетевые посылки. При этом передаваемые по сети данные:

- Упакованы и распаковываются в структуры с учетом выравнивания на машине клиента;
- Все числа имеют сетевой порядок байт (big-endian);
- Динамические массивы передаются с длинами и размеров элементов;
- Каждый передаваемый тип данных сопровождается двумя контрольными суммами, и на стороне клиента данные эти могут отклоняться при несоответствии контрольных сумм.

3. Управление сервисом

В протоколе SRPS каждому уровню (Station, Channel, Table) соответствует свой набор параметров Качества Обслуживания (QoS – Quality of Service), с помощью которых узлы опеделяются по временным и надёжностным характеристикам информационного потока. Эти параметры устанавливают таймауты, размеры буферов обмена и параметры систем связи.

Временные характеристики задают период главного цикла, период обмена конкретной станции, интервал проверки работоспособности (пингования) станции, таймаут пингования, интервал установления соединения (для протоколов с соединением, как TCP), интервал ошибки таймаута для определенных запросов. Конфигурационные характеристики задают имена хостов и IP-адреса, номера портов.

Изменение конфигурации может проводиться в режиме онлайн, путем рассылки сообщений, содержащих структуру QoS и номер абонента. Получатель обрабатывает сообщение стандартным обработчиком HandleMsg и меняет параметры соединения.

4. Макетирование и Реализация приложения

При макетировании приложения распределенной обработки данных АСУТП, требующего и клиент-серверных операций, и публикации-подписки были оценены варианты реализации на DDS и на TA. В качестве DDS сравнивалась OpenSplice, community edition, версия 5.2, платформа Linux 2.6 для X86. В DDS-реализации потребовался промежуточный сетевой сервис networking, с которым работает приложение через общую память на локальной машине. На сервисе была включена конфигурация BestEffort, конфигурация reliable с обеспечением механизма последовательности и гарантии передачи данных была отключена. Следующая тестовая схема была реализована. Три программных узла обмениваются короткими посылками на 3-х разных компьютерах (10.0.201.216 10.0.201.206 10.0.28.5) с интервалом 20 мс. Из-за такого явления, как «джиттер» [4], посылки приходят на адресаты с различными задержками. Буферизация по времени отсутствует, если принятая посылка не попадает в интервал, то она отбрасывается. Процент не попавших в интервал фиксируется и служит оценкой качества работы коммуникационных схем в реальном времени.

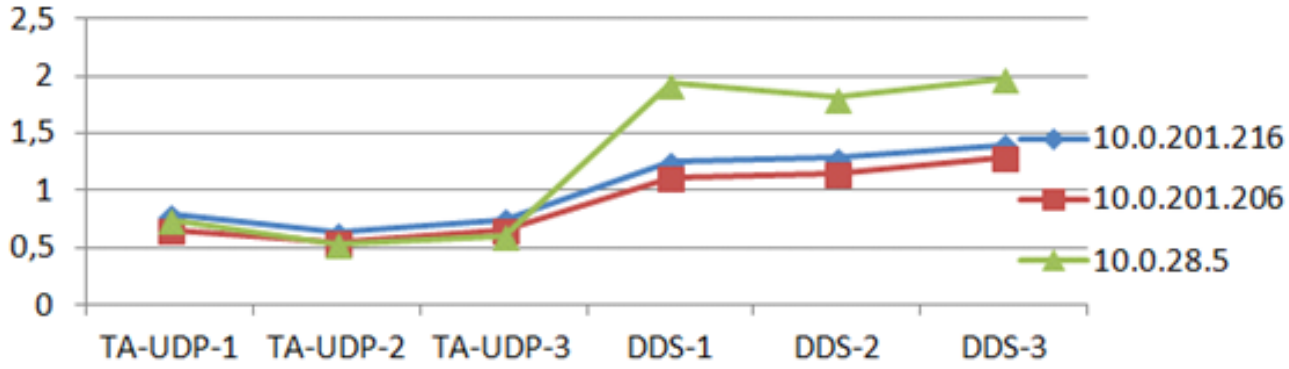


Рис. 2 – Процент числа пакетов, вышедших за границы интервала

Полученный результат свидетельствует о том, что дополнительные сервисы и потоки еще и вносят дополнительные задержки и девиации циклов, что может заметно ухудшить качество передачи данных. Естественно, буферизация на стороне клиента улучшает ситуацию, но не во всех случаях это возможно.

В результате решение было принято в пользу TA, как гораздо менее избыточного подхода, при использовании разных протоколы для подходящих им задач. В настоящее время на TA ведутся работы по модернизации ПО.

Первоначально коммуникационное ПО было реализовано в виде C-библиотеки с возможностью присоединения к BlackBox. Начиная с версии 2.0, появились Оберон-реализации для BlackBox и XDS. Все реализации тестировались для ОС Windows и Linux.

TA реализована, версия 2.0 доступна в исходных текстах как ta1 на sourceforge.net.

Литература

1. Michi Henning, Steve Vinoski. Advanced CORBA Programming with C++. Addison Wesley Longman, 1999.
2. Data Distribution Service for Real-Time Systems. Version 1.2. OMG available specification. Formal 07/01/01.
3. Michi Henning. The Rise and Fall of CORBA, ACM Queue, Volume 4, Number 5, June 2006. Пер. Сергей Кузнецов.
4. Таненбаум Э., Уэзеролл Д. Компьютерные сети, 5-е изд. СПб.: Питер, 2012.
5. Э. Гамма, Р.Хелм, Р.Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001.

УДК 681.3

РЕДАКТОР ОНТОЛОГИЙ ДЛЯ ОНТОЛОГОУПРАВЛЯЕМОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Грегер Сергей Эдуардович, доцент, Уральский федеральный университет имени первого Президента России Б.Н.Ельцина, Нижнетагильский технологический институт (фил.), Россия. Нижний Тагил, segreger@gmail.com

Ахметов Даниил Равилевич, студент, Уральский федеральный университет имени первого Президента России Б.Н.Ельцина, Нижнетагильский технологический институт (фил.), Россия. Нижний Тагил, dan1@liveinternet.ru

На VI конференции “Объектные системы – 2012” нами был представлен доклад на тему: “Редактор метамодели онтологической системы” [1] в котором рассматривалась возможность построения корпоративных порталов на основе CMS Plone с использованием онтологических моделей. Главной особенностью CMS Plone является использование в качестве хранилища контента объектной базы Zope Object Database (ZODB). Таким образом, целью разработки являлось создание редактора онтологий, хранимой в объектной базе ZODB. Перед нами стояла задача создать редактор, управляющий графом объектов и отображающий структуру графа в иерархическом виде.

Для хранения онтологии в объектной базе данных был разработан набор компонент, обеспечивающих представление концептов языка OWL, таких как *Class*, *Data Property*, *Object Property* и их атрибутов [2]. Для хранения индивидов онтологии был использован компонент *Individual*. Использование таких компонент позволило эффективно хранить онтологию и реализовать выполнение запросов к онтологии. При создании и модификации элементов онтологии сервер запросов манипулирует объектами указанных классов, используя средства управления объектной базы данных ZODB.

К разрабатываемому приложению были предъявлены следующие требования:

1. Редактор должен обладать возможностью загружать и визуализировать любую, представленную на языке OWL, онтологию.
2. Редактор должен предоставлять пользователю дружественный интерфейс, позволяющий быстро создавать и редактировать концепты онтологии.
3. Редактор должен предоставлять гибкий поисковый механизм, позволяющий легко искать и редактировать концепты онтологии.

Однако дальнейшие исследования привели к тому, что в разработанном редакторе был выявлен ряд недочетов касательно пользовательского интерфейса и возможности совместной разработки.

Перечислим основные недочеты предшествующей разработки:

1. Элементы онтологии представлены в виде каскада раскрывающихся списков (дерева), однако такой подход не позволяет увидеть связи между несколькими элементами онтологии.
2. При использовании онтологий больших размеров, редактор показывает очень низкую производительность, связанную с тем, что загружает все данные об онтологии при первом же обращении. Помимо нецелесообразного расходования оперативной памяти, происходит генерация очень большого количества html-разметки, что негативно сказывается на производительности браузера.
3. При изменении или добавлении новых элементов в онтологию происходит полная перезагрузка страницы, после чего данные об онтологии вновь загружаются в оперативную память, и генерируется html-разметка. В процессе моделирования, скорость заполнения онтологии пользователем довольно низка.
4. Не предусмотрены возможности совместной работы над онтологией.

Рассмотрим возможные пути устранения этих недочетов:

1. Для устранения первого недочета было решено добавить несколько дополнительных видов отображения онтологии. Так как онтология представляет собой граф, логично