И.Б.Бурдонов, Иванников В.П., Косачев А.С.
Проект КЛАСТОС.
Труды SORUCOM-2011, Вторая международная конференция «Развитие вычислительной техники и ее ПО в России и странах бывшего СССР». Великий Новгород, 12-16 сентября 2011г., стр. 76-82.
7 стр.

Проект КЛАСТОС

Бурдонов И.Б., Иванников В.П., Косачев А.С.
Институт системного программирования РАН
ivan@ispras.ru

The KLASTOS Project

Burdonov I., Ivannikov V., Kosachiov A.
Institute for System Programming RAS

1. Введение

Объектная ориентация сегодня является общим свойством программистской культуры. Она стала глобальным подходом к построению программных систем, интегрировав базисные идеи, исторически возникшие в разных областях программирования – от языков до операционных систем и баз данных. Мы пришли к объектно-ориентированнному подходу, прежде всего, как разработчики операционных систем, что наложило свой отпечаток на наше понимание объектной ориентации: ее целей, проблем и способов их решения.

С начала 70-х годов в течение 10 лет мы занимались большими программными системами – как непосредственной разработкой и реализацией таких систем, так и проектированием средств для создания прикладных систем такого рода. Речь идет о проекте АС-6, в рамках которого создавались операционные системы для нескольких ЭВМ в локальной сети с единой распределенной системой управления внешними устройствами и линиями связи, распределенные информационные системы и распределенные прикладные системы управления космическими объектами.

Общими чертами всех этих систем было наличие большого количества асинхронно выполняемых процессов, управляющих сложными внешними объектами и обменивающихся информацией между собой. Управление этими процессами, их синхронизация, организация взаимодействия между ними, обеспечение защиты, надежности и живучести системы в целом, миграция программ с одной платформы на другую, повторное использование созданных ранее программ – вот основные проблемы, с которыми пришлось столкнуться в этой работе.

Анализируя накопленный опыт, мы сформулировали для себя две фундаментальные проблемы: проблему функционирования сложных многокомпонентных программных систем и проблему создания (проектирования и программирования) таких систем.

«Идеальная» *среда функционирования* – это такая среда, которая для компонентов сложной программной системы обеспечивает не только эффективные средства взаимодействия, но и максимальную асинхронность и защиту:

- независимое и асинхронное выполнение компонентов (с учетом их приоритетов и квантованием процессор ного времени для равноприоритетных компонентов);
- отсутствие общей памяти, защищенность памяти каждого компонента от доступа других компонентов;
- взаимный обмен между компонентами данными, передаваемыми по значению (сообщениями), с их буфери зацией вне компонентов для обеспечения полной асинхронности выполнения и с возможностью для каждого компонента самостоятельно управлять потоком получаемых им сообщений;
- возможность установления статических и динамических связей между компонентами любой требуемой кон фигурации с одновременным жестким контролем санкционированности доступа одного компонента к другому.

Впоследствии такие среды функционирования служб ОС, как компонентов сложных систем, получили название микроядер ОС. В дальнейших рассуждениях мы будем использовать этот термин.

«Идеальная» *среда разработки* программных систем должна обеспечивать:

- поддержку различных методологий проектирования («снизу вверх», «сверху вниз» и смешанную);
- спецификацию взаимодействия, отделенную от реализации;
- зависимость программирования компонента только от спецификации его взаимодействий;

- статическое связывание компонентов в систему с возможностью явного проведения связей и контроля типов;
- повторную используемость (без циклов «редактирование – компиляция») как атомарных компонентов, так и сконструированных систем в качестве компонентов систем более высокого уровня;
- быстрое прототипирование с возможностью автоматического встраивания средств отладки и трассирования;
- управление версиями;
- документируемость на всех этапах и уровнях разработки.

В терминах сегодняшнего дня можно сказать, что эти свойства сред функционирования и разработки программных комплексов соответствуют общим критериям объектно-ориентированного подхода. Ниже мы постараемся указать на особенности проекта КЛАСТОС и обосновать выбор решений. В отличие от других работ в области объектной ориентации (например, Smalltalk) проект КЛАСТОС исторически и логически развивался от разработки наиболее эффективных механизмов среды функционирования объектов к языкам программирования и проектирования таких объектов. В соответствии с этим и построено наше дальнейшее изложение.

2. Операционная система как среда функционирования сложных программных комплексов

Базовый объект, поддерживаемый микроядром, получил у нас название *кластер*. Его основные свойства:

- *полная инкапсуляция* – кластер обладает собственным защищенным адресным пространством;
- кластер – это *активный объект*, обладающий одним потоком управления;
- *обмен сообщениями* – кластер вызывает операцию в другом кластере, посылая сообщение на соответствующий операции вход (приемный порт) кластера.

Микроядро предоставляет кластерам минимальный набор системных вызовов, называемых *примитивами*.

Базовая модель взаимодействия

Существуют три модели взаимодействия, основанные на обмене сообщениями:

1) посылка сообщения без ожидания подтверждения приема;
2) посылка сообщения с ожиданием подтверждения приема;
3) посылка сообщения с получением ответного сообщения (синхронный вызов удаленной процедуры).

Анализируя эти модели, мы исходили из критерия максимальной асинхронности работы кластеров. Во второй и третьей моделях возможна блокировка текущего потока управления, и для полной асинхронности требуется наличие нескольких потоков управления в одном объекте. Мы отказались от поддержки нескольких потоков управления над общей памятью по той причине, что, во-первых, это может являться источником трудно обнаруживаемых ошибок, а, во-вторых, накладные расходы на синхронизацию, необходимую при работе с общей памятью, не существенно меньше накладных расходов на обмен сообщениями. Поэтому в качестве базовой нами была выбрана первая модель взаимодействия, что потребовало организации в микроядре промежуточной *буферизации сообщений*.

По причине отказа от блокировки потока управления мы предпочли *двухточечную схему взаимодействия*, поскольку схема широковещания эффективна только, если рассматривается как атомарное действие, включающее одновременную доставку (или не доставку) сообщения всем адресатам.

Синхронизация

При наличии только одного потока управления в кластере и отсутствии общих данных между кластерами синхронизация сводится к управлению выборкой сообщений, поступивших к кластеру и буферизованных микроядром. Сообщение соответствует вызову операции в объекте и адресуется *входу* (приемному порту) кластера, ассоциированному с операцией. Управление выборкой сообщений осуществляется самим кластером указанием набора входов, по которым ожидается прием сообщения, или, иными словами, набора операций, которые объект может выполнять в данном состоянии. Для обеспечения

хронологии сообщений микроядро буферизует их в *очередях* кластера, причем, если важен порядок прихода сообщений, адресованных разным входам, такие сообщения накапливаются в одной очереди.

Управление доступом

Исходя из критерия максимальной защиты, мы стремились обеспечить защищенность не только памяти, но и входов кластера от несанкционированного доступа. В качестве защищенной единицы доступа выбран не один вход (одна операция), а определенная статически *группа входов* (набор операций) кластера. Забегая вперед, можно сказать, что такой набор операций соответствуют интерфейсу объекта с клиентом. Тип интерфейса аналогичен той части определения класса, которая специфицирует взаимодействие с объектом и соответствует понятию абстрактного типа данных в языках без механизма наследования (CLU, ADA).

Кластер может реализовывать не одну, а несколько групп входов (предоставлять несколько интерфейсов), не одновременно доступных клиентам. Сам доступ реализуется как вспомогательный объект микроядра, называемый *подключением* (к группе входов).Кластер, реализующий группу входов подключения, называется кластером-*сервером*. У подключения может быть только один владелец, называемый кластером-*клиентом*, которому разрешено посылать сообщения на входы (вызывать операции) соответствующей группы.

Подключения могут динамически создаваться и уничтожаться кластером-сервером. Подключение может быть передано другому кластеру в сообщении – как параметр вызова операции в кластере. Передача подключений производится под контролем микроядра, которое проверяет право владения подключением и меняет его владельца после передачи. Этим аппарат подключений отличается от аппарата мандатов, в котором при передаче мандата другому объекту права по доступу, как правило, сохраняются у передающего объекта.

Кластер может лишиться права владения подключением также в результате посылки сообщения на некоторые выделенные входы группы, называемые *входами отстройки* и семантически соответствующие операциям завершения транзакции клиент-сервер (операциям отстройки). Например, операция закрытия файла прекращает для клиента доступ к операциям чтения/записи файла. При вызове операции отстройки подключение передается как неявный параметр и его владельцем становится кластер-сервер.

К одной своей группе входов сервер может создать несколько подключений и передать их разным клиентам. Микроядро предоставляет серверу средства различения клиентов при выборке сообщений от них, идентифицируя подключение, через которое сообщение посылается. Кроме того, сервер может задать *селективный прием* сообщения только от указанного клиента (идентифицируемого подключением).

Некоторыми подключениями кластер может владеть "с самого начала" – они появляются в момент загрузки кластера. Эти подключения описываются статически на этапе компоновки, рассматриваемой в разделе 3. Через статические подключения возможен доступ только к тем группам входов, которые при программировании сервера были определены как *статические группы входов*. Например, у кластера файла статической является лишь группа операция открытия файла.

Таким образом, описанные средства работы с подключениями обеспечивают возможность установления полностью защищенных межкластерных связей любой требуемой конфигурации и ее динамическое изменение.

Базовые примитивы (системные вызовы) микроядра

Описанные выше средства взаимодействия, синхронизации и управления доступом предоставляются пользователям (кластерам) как набор четырех базовых примитивов (системных вызовов) микроядра:

1. SEND – примитив посылки сообщения.
   Параметры:
   - подключение, через которое посылается сообщение;
   - номер входа в группе входов подключения;
   - параметры операции, ассоциированной с указанным входом.

2. ACCEPT – примитив приема сообщения.
   Параметры:
   - список входов, по которым ожидается прием сообщений;
   - подключение (указывается для селективного приема сообщения от одного клиента).

   Существуют две модификации примитива: с блокировкой кластера до поступления нужного сообщения и без блокировки.

3. NEWCON - примитив создания подключения.
   Параметры:
   - группа входов кластера.

4. DELCON - примитив уничтожения подключения.
   Параметры:
   - уничтожаемое подключение.

Удаленная процедура и модель клиент-сервер

Хотя микроядро реализует чисто асинхронную модель взаимодействия (send/receive), во многих приложениях часто бывает удобным использование удаленных процедур. Следует различать удаленную процедуру как базовый механизм взаимодействия и как *вид интерфейса*, предполагающий получение обратных параметров. Такой вид интерфейса реализуется с помощью базовых средств определенной дисциплиной работы клиента и сервера (которая может быть скрыта поддержкой языка времени исполнения):

1. Клиент определяет у себя специальную *обратную* группу входов для приема *обратного* сообщения от сервера, содержащего *обратные* параметры удаленной процедуры. Такая *обратная* группа входов состоит из одного *обратного* входа отстройки.
2. Когда клиент посылает серверу *прямое* сообщение на вход вида удаленная процедура, в число *прямых* параметров процедуры добавляется *обратное* подключение к *обратной* группе входов клиента.
3. Сервер, получив от клиента *прямое* сообщение на вход вида удаленная процедура и выполнив соответствующую этому входу операцию, посылает клиенту через *обратное* подключение *обратное* сообщение, содержащее *обратные* параметры процедуры. (Поскольку *обратный* вход является входом отстройки, *обратное* подключение возвращается от сервера к клиенту.)

В системе КЛАСТОС для организации программного обеспечения, оставшегося за чертой микроядра, используется модель клиент-сервер, общепринятая в системах с микроядерной архитектурой. Сервисы системы реализуются кластерами-серверами. Кластер-клиент посылает сообщение-запрос серверу, сервер выполняет работу и отправляет сообщение-ответ. Интерфейс клиента и сервера, таким образом, организован как вызов удаленной процедуры.

Сегментная организация памяти

КЛАСТОС – это система, основанная на обмене сообщениями между объектами, не имеющими общей памяти. Для того, чтобы избежать излишних накладных расходов, связанных с передачей больших массивов информации в сообщениях (перепись из памяти одного кластера в память другого кластера), вводится сегментная организация памяти. Сегмент – это еще один (наряду с подключением) вспомогательный объект, поддерживаемый микроядром. При передаче сегмента как параметра операции он изымается из виртуальной памяти кластера-клиента и при приеме сообщения помещается в виртуальную память кластера-сервера. Разумеется, это требует соответствующей аппаратной поддержки (аппаратура приписки).

Память кластера – это набор сегментов, два из которых (сегмент кода и сегмент статических данных и стека) кластер не может передавать как параметр операции. Управление сегментами может осуществляться специальным кластером-сервером, предоставляющим операции запроса, отказа и копирования сегментов. В последней версии микроядра для повышения эффективности эти операции реализованы как примитивы (системные вызовы).

В некоторых случаях для повышения эффективности микроядро допускает отступление от общего принципа отсутствия общих данных, предоставляя механизм разделяемых сегментов. При передаче разделяемого сегмента он остается в виртуальной памяти кластера-клиента. Множество кластеров, которым разрешается разделять сегмент, статически ограничивается.

При копировании и передаче разделяемых сегментов используется механизм «copy-on-write».

Базовые серверы

Базовые службы операционной системы, обеспечивающие создание и уничтожение объектов (кластеров) и управление устройствами ввода/вывода и внешней памяти, реализуются кластерами-серверами непосредственно над микроядром.

Создание объектов (динамическая загрузка) выполняется кластером-загрузчиком. Эта операция трактуется в КЛАСТОС как отложенная компоновка составного объекта и будет рассмотрена ниже при описании среды разработки программных систем.

При уничтожении кластера сервер-ликвидатор, помимо уничтожения сегментов памяти и подключений к группам входом кластера, выполняет оповещение всех кластеров, к группам входов которых уничтожаемый кластер имел доступ, об аварийной ликвидации кластера. Для этого через каждое подключение, владельцем которого является уничтожаемый кластер, посылается сообщение (без параметров) на специальный вход отстройки, имеющийся в каждой группе входов.

Если есть простой защищенный способ доступа к необходимым командам и портам внешнего устройства, причем в режиме пользователя, нет причин помещать драйвер устройства в микроядро. Такой драйвер реализуется кластером-сервером. Степень привилегированности такого кластера определяется степенью привилегированности доступа к командам и портам ввода/вывода. Прерывание от устройства интерпретируется микроядром как сообщение, перенаправляемое им к соответствующему входу кластера-драйвера устройства.

Два оптимизационных трюка

Частным, но распространенным случаем сообщения является сообщение без параметров – *сигнал* (например, внешнее прерывание).Для эффективной реализации передачи таких сообщений микроядром поддерживается специальный вид входов кластеров. Очередь сообщений к таким входам редуцируется до простого счетчика числа сообщений. Разумеется, информация о клиенте, передавшем сообщение, теряется.

Забегая вперед, скажем, что в проекте КЛАСТОС спецификации интерфейсов отделены от реализации. Это облегчает реализацию опосредованной коммуникации – использование объектов-посредников как «клея» между несовместимыми объектами. Частым случаем является ситуация, когда часть операций специфицированного интерфейса реализуется одним кластером (посредником), а часть – другим кластером. В целях оптимизации микроядро поддерживает *механизм несобственных входов* в группе входов кластера. Сообщение, направляемое к несобственному входу кластера-посредника, автоматически перенаправляется микроядром на вход другого кластера через подключение, владельцем которого является посредник. При создании подключения к группе входов кластера-посредника примитивом NEWCON для каждого несобственного входа указывается на какой вход и через какое подключение перенаправлять сообщения.

Наращиваемость системы

Выше мы набросали общие контуры нижнего уровня операционной системы как среды функционирования сложных программных комплексов – микроядро и базовые серверы. Конечно, за рамками нашего изложения остались многие детали реализации и функции этого уровня. Над этим уровнем располагаются серверы вышележащих уровней, реализующих файловую систему, управление задачами, СУБД, системы программирования и разработки и т.п.

Как прикладная система строится и система эмуляции операционной системы UNIX. Нами были опробованы два подхода к эмуляции. На первом этапе мы реализовали библиотеку базовых системных вызовов UNIX'a, достаточных для работы SHELL'a. Перенос программ осуществлялся путем перекомпиляции. Затем начал строиться эмулятор, обеспечивающий двоичную совместимость программ, исходные тексты которых недоступны. Этот эмулятор основан на перехвате системных вызовов (trap'ов) UNIX'a микроядром и перенаправлении их в эмулятор.

Распределенность

Основные свойства базовых объектов КЛАСТОС – асинхронность, отсутствие общей памяти, взаимодействие путем передачи сообщений – позволяют без перекомпиляции организовывать распределенные системы. В сетевом варианте доступ кластера-клиента к группе входов кластера-сервера, расположенного в другой машине, реализуется цепочкой:

подключение клиента к кластеру-делегату (посреднику) – кластер-делегат – подключение к сетевой службе – сетевая служба – межмашинная сетевая среда – сетевая служба – подключение к кластеру-делегату – кластер-делегат – подключение к кластеру-серверу.

Этот базовый уровень необходим, но не достаточен для создания полномасштабной распределенности. Остаются общие для такого рода систем проблема выбора процессора для выполнения компонентов пользовательской распределенной системы и проблема миграции как объектов пользователя, так и системных серверов.

3. ПРОКЛОС – среда разработки программных систем

Проект КЛАСТОС исторически и логически развивался от разработки наиболее эффективных механизмов среды функционирования объектов к языкам программирования и компоновки программных систем, состоящих из таких объектов. В общей системе разработки можно логически выделить три составные части: спецификация интерфейсов, программирование объектов (кластеров), реализующих эти интерфейсы, и компоновка составных объектов (подсистем). Для каждой из этих частей характерны свои специфические проблемы и способы их решения. Мы приняли решение о «физическом» разделении этих частей путем использования трех разных языков: язык спецификаций интерфейсов, язык программирования и язык компоновки, базирующихся на общей понятийной базе.

Это наше решение обосновывалось следующими соображениями:

1. Поскольку базовые объекты (кластера) не имеют общей памяти и, тем самым, программы этих объектов не имеют общих переменных, описание составного объекта как единого текста распадается на слабо связанные части - спецификации интерфейсов, программы кластеров и описание статических межкластерных связей.
2. Мы хотели обеспечить независимость языка компоновки от языка программирования кластеров с тем, чтобы можно было использовать различные языки программирования.
3. В качестве первого языка программирования нами был выбран язык Си, расширенный конструкциями, обеспечивающими межкластерное взаимодействие – СИКЛОС. Нам не хотелось дополнительно перегружать язык Си декларативными конструкциями языка компоновки подсистем, которые слабо совместимы с языком Си.

Язык спецификации интерфейсов

Язык спецификации интерфейсов описывает *интерфейсы как типы взаимодействия*, отделенные от реализации. Интерфейс – это набор операций; описание операции состоит из описания вида операции(операция отстройки, сигнал) и описания параметров операции. Интерфейс, определяющий тип взаимодействия клиента и сервера, является на стороне клиента типом подключения, а на стороне сервера – типом группы входов. Операция типизирует вход кластера.

Язык программирования СИКЛОС

На первом этапе проекта КЛАСТОС мы использовали "чистый"язык Си, применяя для организации взаимодействия технику «заглущек», генерируемых компилятором с языка спецификации интерфейсов, как самый простой и быстрый путь реализации. В дальнейшем нам пришлось пойти на расширение Си до СИКЛОСа для описания групп входов и введения типов подключений и сегментов, используемых как параметры операций (на первом этапе они интерпретировались как целое). Без этого было невозможно обеспечить типовой контроль. Язык СИКЛОС предоставляет средства межкластерного взаимодействия, работы с сегментами (типы сегментов и относительных указателей) и исключениями. В частности, введены средства, инкапсулирующие в поддержке языка времени исполнения механизм удаленного вызова процедуры.

Язык компоновки подсистем

Подсистема – это статически определенный набор компонентов со статически установленными связями между ними. Компонент подсистемы – это подсистема или кластер. Процесс построения подсистемы из компонентов происходит до ее использования и называется *компоновкой* подсистемы. Описание подсистемы на языке компоновки содержит два раздела: раздел внешних связей подсистемы и раздел состава и внутренней структуры подсистемы.

С точки зрения ее использования подсистема аналогична кластеру. Она тоже соответствует некоторому объекту (но в отличии от кластера – сложному, составному объекту), реализуя его операции и, в свою очередь, вызывая операции в объектах, определенных вне ее. Внешнюю для подсистемы операционную среду можно рассматривать как дополнительный фиктивный компонент подсистемы – *окружение*.

Раздел внешних связей подсистемы содержит подразделы описания групп входов подсистемы и групп входов ее окружения. *Группа входов подсистемы* отождествляется с некоторой группой входов компонента подсистемы и, в конечном счете, реализуется статической группой входов некоторого кластера. *Группа входов окружения* – это группа операций, которыми подсистема пользуется, но которые определены в объекте, реализованном вне ее. Группа входов окружения, в конечном счете, реализуется статической группой входов некоторого кластера, находящегося вне подсистемы.

Раздел состава и внутренней структуры подсистемы содержит список компонентов подсистемы и связей между ними. Поскольку компонент подсистемы сам является подсистемой (в частном случае – кластером), то его раздел внешних связей также содержит описание групп входов компонента и групп входов окружения этого компонента. Для компонента подсистемы его окружение - это «остальная часть» подсистемы, то есть совокупность других компонентов подсистемы, включая и фиктивный компонент «окружение подсистемы».

Группе входов окружения кластера или подсистемы соответствует одно или несколько подключений к этой группе входов, статически определенных в кластере или подсистеме (в конечном счете, кластерах). Для компоновки подсистемы необходимо каждую группу входов окружения каждого компонента отождествить с некоторой группой входов, вообще говоря, другого компонента, в частности окружения подсистемы. Эта операция отождествления называется *установлением связи* или *связыванием*. Связываемые группы входов должны быть одного типа (интерфейса).

Основной принцип процесса компоновки заключается в том, что связывание групп входов окружения подсистемы со статическими группами входов других подсистем является операцией использования подсистемы в том или ином контексте как компонента более сложной подсистемы. Требования к контексту сформулированы в описании групп входов окружения. На этом принципе основана общая методология создания разнообразных кластеров-посредников, подсистем конвейерной обработки и т.п.

Построение подсистемы из подготовленных заранее компонентов (подсистем и кластеров) соответствует проектированию «снизу-вверх». Для проектирования «сверху вниз» обеспечивается возможность не детализировать содержимое компонентов, типизируя лишь их внешние связи. Основное действие компоновки – связывание – использует только внешние связи компонентов и подсистемы в целом.

Корпусом называется компонент, в котором определены только его внешние связи и не определены состав и внутренняя структура. При проектировании «сверху вниз» вначале подсистема состоит только из

корпусов. На следующем этапе происходит детализация этих корпусов: они описываются как подсистемы, которые, в свою очередь, могут также содержать корпуса. Таких этапов может быть несколько, рано или поздно корпуса детализируются уже имеющимися кластерами и полностью определенными подсистемами.

Корпус неоднозначно определяет внешние связи подсистем, которыми он может детализироваться. Фактически, корпус задает *семейство* подсистем. Это семейство определяется набором статических групп входов корпуса и его окружения, то есть внешними связями корпуса. Каждая подсистема из данного семейства должна иметь все те статические группы входов, которые определены для семейства, но может иметь и другие статические группы входов. А набор групп входов окружения подсистемы должен быть подмножеством набора групп входов окружения семейства.

Полностью определенная (не содержащая корпусов) подсистема является в КЛАСТОС единицей динамической загрузки. Загруженная подсистема в процессе своей работы может изменяться: могут появляться новые компоненты (с помощью динамической загрузки), ликвидироваться компоненты, возникать (и исчезать) новые связи между компонентами. Однако, понятие подсистемы сохраняется и в процессе ее выполнения и используется в управлении процессором, памятью и задачами.

Динамическая загрузка

При загрузке полностью скомпонованной подсистемы необходимо произвести связывание групп входов ее окружения с группами входов уже загруженных кластеров. Для этого в КЛАСТОС используется понятие *параметризованного загрузчика* – это кластер (вообще говоря, подсистема), который имеет подключения ко всем тем статическим группам входов, с которыми должны быть связаны группы входов окружения загружаемой подсистемы.

Таким образом, в объемлющей подсистеме параметризованный загрузчик находится как бы вместо загружаемой подсистемы (или вместо корпуса, который может детализироваться этой подсистемой).Загрузка производится при обращении к его входу ЗАГРУЗИТЬ. Загруженная подсистема помещается «рядом» с загрузчиком (в отличии от детализации корпуса, когда подсистема помещается «вместо» корпуса), а подключения ее кластеров к группам входов окружения оказываются копиями подключений загрузчика.

Динамическая загрузка подсистемы может рассматриваться как отложенная компоновка объемлющей подсистемы – детализация – с одним существенным различием: при загрузке подсистема попадает в контекст уже работающей объемлющей подсистемы.

Выбор в качестве единицы загрузки скомпонованной подсистемы с уже установленными внутренними статическими связями, а также загрузка параметризованным загрузчиком с заранее установленными внешними статическими связями позволяют избегать лишней динамики, уменьшая тем самым накладные расходы времени выполнения. Вместе с тем возможность загрузки подсистемы различными параметризованными загрузчиками, находящимися в различном окружении и соответствующими различным загрузочным семействам, в которые «вкладываются»внешние связи данной подсистемы, позволяет помещать эту подсистему в различный контекст при загрузке. Этот своеобразный унифицированный способ параметризации подсистемы является также средством обеспечения защиты: подсистема может получить статический доступ только к тем кластерам, к которым ей разрешит иметь доступ соответствующий загрузчик.

Дальнейшее развитие ПРОКЛОСа

Описанные выше средства среды разработки программных комплексов образуют базовый уровень ПРОКЛОСа. Дальнейшее развитие может быть связано с созданием интегрированной системы, включающей пользовательски-ориентированный визуальный интерфейс, базы данных интерфейсов и объектов (кластеров и подсистем), быстрое прототипирование с возможностью автоматического встраивания инструментов отладки и трассирования, управление версиями и документирование на всех этапах и уровнях разработки.

4. Сравнение подходов

Проведем некоторое сравнение нашего подхода с наиболее распространенными.

Выбранная нами модель взаимодействия дает наибольшие возможности для распараллеливания работ, учитывая наш отказ от нескольких потоков управления (thread'ов) и наличия общей памяти. В то же время она позволяет на уровне поддержки языка организовывать другие модели, в частности, модель вызова удаленной процедуры.

Аппарат подключений и групп входов в отличие от аппарата мандатов (capabilities) дает возможность объекту динамически управлять доступом клиента к своим операциям. Разбиение всего множества операций сервера на группы, доступ к которым сервер предоставляет клиентам динамически как обратные параметры операций, дает возможность верифицировать средствами темпоральной логики развитие взаимодействий клиента и сервера во времени. Например, доступ к операциям «читать» и «писать» клиент получит только в ответ на обращение к операции «открыть» сервера файла. Этот доступ контролируется не средствами самого сервера, а микроядром.

Статическое связывание компонентов и доопределение внешних связей при загрузке подсистемы статически определенными подключениями решает проблему именования, которая очень актуальна для систем с только динамическим установлением связей. Кроме того существенно снижаются накладные расходы на установление связей.

Отделение спецификаций интерфейсов от их реализации дает, на наш взгляд, большие возможности для повторного сборочного использования программ, чем аппарат классов с наследованием. Особенно это проявляется в случае эволюционирующих систем и систем с масштабируемыми функциями. Наши средства дают возможность модификации компонента (или выбора из набора совместимых по интерфейсам компонентов) при построении систем с требуемыми свойствами. Нет проблем и с введением коммуникационных посредников. Важно, что все эти изменения не требуют переделок в остальных компонентах.

Основное отличие нашего подхода от ставшего уже классическим объектно-ориентированного подхода, которого придерживались в системах с классами объектов и наследованием классов, заключается в том, что у нас объектная ориентация проявляется на уровне создания и функционирования системы в целом, а не атомарного компонента (кластера).

5. Заключение

Проект КЛАСТОС так и остался исследовательской работой, так как ни в СССР, ни в России на нее не нашлось серьезных заказчиков. На начальной стадии этим проектом мы занимались в свободное от основной работы время. Особые трудности были связаны с отсутствием подходящей платформы для реализации, учитывая то, что реализация должна была начинаться с микроядра. Достаточно сказать, что первый вариант КЛАСТОС был реализован в 1989 г. на советском аналоге DEC Professional самой младшей модели.

Что же касается «идейной» части работы, можно сказать: то, что мы делали, мы делали параллельно с общим движением программирования к объектной ориентации и микроядерной архитектуре ОС. В итоге в проектировании не было допущено грубых ошибок, которых нам приходилось бы теперь стыдится. На наш взгляд, решения и выборы, сделанные нами в процессе разработки, представляют интерес для разработчиков аналогичных систем. В нынешних условиях дальнейшие перспективы проекта весьма туманны, поскольку зависят от того, сможем ли мы найти материальную и финансовую базу для сохранения коллектива разработчиков и продолжения работ.