Распределенные объектно-ориентированные системы

В.П. Иванников, К.В. Дышлевой, С.Г. Манжелей,
Л.Б. Соловская, А.Б. Шебуняев

Аннотация

Группа "Распределенные объектно-ориентированные системы" существует в ИСП РАН с осени 1995 года. Научные интересы группы связаны с распределенной обработкой в неоднородных средах и созданием унифицированной среды для разработки и выполнения распределенных объектно-ориентированных приложений. В частности, рассматривается применение объектно-ориентированного подхода для создания распределенных систем, активные и пассивные объектные модели, метаобъектные протоколы, построение распределенных информационных систем. Со времени создания группа участвовала в нескольких проектах по заказам компаний Nortel Networks, Biomax Informatics GmbH и работах, поддерживаемых Российским Фондом Фундаментальных Исследований (РФФИ). Были разработаны брокеры объектных заявок для языков C++ и Protel-2, предложена методология построения распределенных систем на основе модели оболочки, в стадии реализации находится информационная система для биологических банков данных.

1. Введение

Современные программные системы создавались и создаются в условиях существенной неоднородности и распределенности программной среды. Реализационная неоднородность определяется историческим, эволюционным разнообразием платформ, средств и моделей программирования. Информационная неоднородность связана с разнообразием прикладных контекстов, способов абстракций и степенью приближения к моделируемым объектам и процессам реального мира.

Развитие сетевых технологий предоставило удобную инфраструктуру для разработки больших распределенных программных комплексов. А потребность в распределенной обработке привела к появлению технологии создания *интероперабельных информационных систем*.

Под интероперабельностью понимается способность совместной, согласованной деятельности разнородных компонент системы для решения определенной задачи. Основу информационной архитектуры интероперабельных систем составляет концепция промежуточного слоя (middleware). Интероперабельность достигается введением общего механизма поддержки взаимодействия компонентов (например, брокера объектных заявок), введением общей базовой модели компонентов (как правило, объектной), унифицированного языка спецификаций интерфейсов, отделением реализации компонентов от спецификации их интерфейсов. Тем самым достигается однородность представления компонентов и их взаимодействия. Для полноты информационной архитектуры также вводится слой унифицированных ортогональных служб, конструируемых по тем же принципам, что и обычные приложения.

Распределенные технологии находят применение при построении больших систем с повышенными требованиями к надежности, быстродействию, открытости. К таким системам можно отнести телекоммуникационные программные комплексы, системы реального времени, информационные системы.

Для создания сложных систем с множеством межкомпонентных связей широкое распространение получили объектные технологии. К достоинствам объектного подхода можно отнести естественную декомпозицию предметной области на объекты в сочетании с инкапсуляцией данных, определение интерфейсов между компонентами, возможность рассматривать систему на разных уровнях абстракции, потенциальную возможность повторного использования как функциональности, так и архитектуры системы (полиморфизм, наследование). Такие крупные международные организации, как международный комитет по стандартизации International Organization for Standardization (ISO), консорциумы Object Management

Group (OMG) и Telecommunications Information Network Architecture – Consotium (Tina-C), компания Microsoft, приняли объектно-ориентированную парадигму за основу своих распределенных технологий.

Развитие распределенных технологий сопровождается разработкой открытых стандартов, активно поддерживаемых разработчиками программных продуктов. На настоящий момент наиболее популярны следующие промышленные стандарты на программное обеспечение промежуточного слоя: Common Object Request Broker Architecture (CORBA) консорциума OMG, Distributed Component Model (DCOM) компании Microsoft, Telecommunications Information Network Architecture (TINA) консорциума Tina-C.

Определенным ограничением перечисленных стандартов можно считать недостаточно последовательный подход к поддержке единой среды разработки и выполнения распределенных приложений. Как правило, в стандартах предлагается спецификация набора слабо связанных между собой сервисов (например, сервисы именования, конкурентного доступа), которые могут использоваться во время выполнения приложения, а этап разработки просто не рассматривается. Разработка конкретных проектов, связанных с построением распределенных систем, показала необходимость использования единой среды создания и выполнения распределенных приложений. Также в процессе работы был выявлен ряд недостатков, связанных с надежностью программирования и эффективностью выполнения распределенных приложений. Предложениям по решению этих и других проблем, связанных с программированием в распределенных средах, посвящена основная часть этой статьи.

Оставшаяся часть статьи имеет следующую структуру. Второй раздел посвящен задаче построения унифицированной среды разработки и функционирования распределенных приложений и ряду проблем, возникающих при этом. В третьем разделе более подробно рассматриваются конкретные проекты, реализованные группой. Эти проекты связаны с разработкой коммуникационного программного обеспечения для использования в распределенных телекоммуникационных системах и разработкой информационно-поисковой системы для биологических банков данных. Четвертый раздел статьи посвящен научным интересам группы, а именно, использованию метаобъектных протоколов для создания среды проектирования и выполнения распределенных приложений, рассмотрению активных объектных моделей, подходам к оптимизации процедур маршалинга, проблемам адаптации унаследованного кода в CORBA-среду. В заключении подводятся некоторые итоги и рассматриваются возможные направления исследований в области распределенных вычислений.

1. Среда разработки и выполнения распределенных объектно-ориентированных приложений

Для поддержки всего жизненного цикла распределенной системы предполагается наличие единой среды разработки и выполнения распределенных объектно-ориентированных приложений. Такая среда должна предоставлять расширяемый набор системных сервисов, которые могут прозрачным образом подключаться к объектам приложения и таким образом расширять их прикладную функциональность. Помимо этого, среда должна обеспечивать согласованное взаимодействие распределенных контекстов, в которых существуют объекты приложения. Для поддержки функционирования распределенной системы, а также для обеспечения возможности прозрачного подключения системных сервисов, необходим механизм взаимодействия объектов, отвечающий требованию позднего связывания. Техника позднего связывания подразумевает, что клиент получает объектную ссылку на серверный объект с помощью системного окружения уже во время выполнения приложения. Примером механизма, обеспечивающего необходимый стиль взаимодействия объектов, может служить брокер объектных заявок, используемый в архитектуре CORBA (OMG) [CORBA98].

По заказу компании Nortel Networks для использования в проекте по созданию открытых телекоммуникационных систем группой были разработаны брокеры для языков C++ и Protel-2.

Телекоммуникационные системы предполагают высокую надежность, быстрое время реакции, отказоустойчивость, параллельное обслуживание множества запросов. Постоянная эволюция и внедрение новых технологий, необходимость поддержки работающих унаследованных программ делает необходимым построение систем с

открытой архитектурой, способных функционировать в распределенных неоднородных средах.

Промышленный стандарт промежуточного слоя CORBA широко применяется для построения открытых распределенных систем в телефонии. Использование механизма брокера объектных заявок позволяет взаимодействовать компонентам системы, реализованным на разных платформах и, возможно, в разных объектных моделях.

Как и большинство стандартов на архитектуру распределенных систем, стандарт CORBA предусматривает спецификацию набора системных сервисов (например, сервисы именования, поддержки жизненного цикла объектов, конкурентного доступа). Но, как правило, эти сервисы разрабатываются изолировано друг от друга, а их функциональность частично пересекается. Более совершенная схема предполагает наличие единой среды разработки и выполнения распределенных объектно-ориентированных приложений, предоставляющей потенциально расширяемый набор ортогональных сервисов. Подключение сервисов, по возможности, должно происходить прозрачным для приложений образом. Принцип ортогональной декомпозиции особенно важен для возможности независимой разработки и подключения сервисов, а, следовательно, и для систематического расширения системы.

В рамках работы была предложена модель оболочки, обеспечивающая прозрачное подключение сервисов и согласование распределенных контекстов на основе метаобъектных протоколов (МОП) [IVDKMS97]. Техника метаобъектного контроля и рефлексии уже широко используется для проектирования гибких, динамически адаптивных программных систем в объектно-ориентированном программировании. Но, как правило, метаобъектный контроль применяется либо на фазе проектирования, либо на фазе выполнения. Однако наиболее естественным и удобным представляется использование метаобъектных протоколов на всех этапах жизни программной системы. Постановка такой задачи требует достаточно эффективной реализации механизма МОП. Одна из возможных реализаций на основе интерфейсных объектов была предложена и будет более детально рассмотрена в четвертом разделе этой статьи.

В процессе реализации брокера объектных заявок для языка C++ был обнаружен ряд недостатков в работе с отображением из языка спецификаций OMG Interface Definition Language (IDL) в язык реализации C++. Был предложен новый подход, основанный на оболочках-посредниках, обеспечивающих надежную схему работы с памятью [Dysh97]. Как показали дальнейшие исследования, использование таких посредников предоставляет технологическую базу для “прозрачного” расширения функциональности приложений и, в частности, для реализации МОП.

В распределенных системах неизбежно возникает необходимость преобразовывать данные из одного формата в другой. В связи с увеличением быстродействия компьютерных сетей становится актуальным построение эффективных протоколов для передачи данных в гетерогенных средах. При реализации брокера для языка C++ были рассмотрены различные подходы к оптимизации процедур преобразования данных (маршалинга), позволяющие реально увеличить скорость взаимодействия распределенных объектов [DKS97].

Как и любая другая технология, распределенные объектные технологии сталкиваются с проблемой использования унаследованного кода. Принцип отделения интерфейсов от реализаций предоставляет для этого определенные технологические возможности. При разработке брокера объектных запросов для языка Protel-2, предназначенного для работы в специализированной среде главного модуля телекоммуникационной системы, возникла необходимость учитывать уже существующий, унаследованный код. В частности, требовалось включать в порождаемый компилятором код уже существующие модули реализации, использовать нестандартные имена. Для поддержки унаследованного кода был предложен подход, расширяющий возможности стандартного отображения из IDL в язык реализации [KMS97]. Также были рассмотрены возможности автоматизации процесса адаптации унаследованного кода (в частности, порождение спецификаций интерфейсов по коду реализаций).

Наряду с традиционными пассивными моделями, определенный класс задач предполагает использование моделей активных объектов, более точно отражающих реальную действительность. Поэтому среда разработки и выполнения распределенных систем должна предоставлять возможность согласованного

взаимодействия активных и пассивных объектов, поддержку синхронных и асинхронных вызовов, их динамическую трансформацию. В процессе исследовательской работы был рассмотрен ряд известных активных моделей, предложена их классификация и на ее основе предложена *метамодель*, позволяющая, с одной стороны, предоставить программисту уже освоенные средства, заимствованные из предшествующих моделей, а с другой стороны, освободить его от решения типичных проблем и по возможности защитить от ошибок программирования.

Разработка брокеров объектных заявок, исследования, направленные на повышение надежности и эффективности программирования с использованием брокеров, тем или иным образом связаны с разработкой элементов единой среды разработки и функционирования распределенных приложений. Построение информационной системы для биологических банков данных является реальным примером создания именно такого распределенного объектно-ориентированного приложения. Эта информационная система создается на базе архитектуры CORBA с использованием брокера объектных заявок для языка C++ и стандартных системных сервисов. Необходимость совместного использования сервисов (в частности, сервиса именования и сервиса жизненного цикла объектов), потребность согласовывать действия клиента и сервера подтвердили необходимость унифицированной среды для поддержки распределенных вычислений.

3. Проекты распределенных систем

3.1. Телекоммуникационные системы

Использование распределенных технологий в телефонии объясняется спецификой предметной области: большое количество клиентов, обслуживаемых одновременно, требования к надежности, отказоустойчивости, времени реакции системы. Подключение новых сервисов, большой объем унаследованных программ, потенциальное развитие и внедрение новых технологий в уже работающую систему предполагает построение систем с открытой архитектурой.

Модуль, непосредственно выполняющий обработку телефонных звонков в режиме реального времени, в телекоммуникационных системах, как правило, реализован и выполняется на определенной платформе. Например, Digital Multiplex Switch (DMS), Nortel Networks, работает на специальном процессоре Compute Module (CM) под управлением операционной системы SOS. В простейшем случае все сервисы (секретности, авторизации, аутентификации и т.д.) могут разрабатываться в этой же среде. Но со временем разработка новых сервисов становится слишком сложной и дорогой операцией как из-за недостаточной инкапсуляции функциональности DMS, так и из-за невозможности выполнения всех сервисов на одной платформе.

Проекты создания открытых систем в телефонии (например, проект Open Node для Nortel Networks) ставят своей целью пошаговую разработку пользовательских сервисов и расширение функциональности главного модуля на стороне заказчика (или на "третьей" стороне). При этом существенно, что сервисы могут быть прототипированы и реализованы на разных платформах, наиболее полно отвечающих требованиям той или иной задачи. Это, позволяет, во-первых, разгрузить основной модуль и, во-вторых, более гибко и удобно адаптировать систему для конкретного заказчика. Независимая разработка сервисов упрощает управление системой в целом и внедрение новых технологических решений.

Применение объектно-ориентированного подхода в сочетании с той или иной распределенной технологией позволяет объединять сервисы, внешние по отношению к главному модулю и его среде, в единую интероперабельную систему, обеспечивая сервисам доступ к главному модулю в виде вызова удаленных объектов.

Создание открытых систем предполагает наличие открытых интерфейсов, специфицированных на уровне международных промышленных стандартов. Наличие таких стандартов обеспечивает гибкость и переносимость прикладных программ в рамках определенной технологии. В области телекоммуникационных систем существует ряд собственных стандартов и подходов. Например, технология Computer Telephony Integration (CIT) определяет интерфейсы прикладного программирования (API) для интеграции программного обеспечения с телекоммуникационным оборудованием. Среди предложенных в CIT стандартов и решений можно назвать Telephony Application Programming Interfaces (TAPI), реализованный компанией Microsoft, Analog Display System Interfaces (ADSI), NetWare Telephony Services

Application Programming Interfaces (TSAPI), реализованный компанией Novell, Signal Computing System Architecture Telephony Application Objects (SCSA TAO).

Одной из наиболее известных международных организаций, занимающихся разработкой телекоммуникационных систем, является консорциум TINA-C, образованный в 1993 году и объединяющий более 40 производителей телекоммуникационного обеспечения. Деятельность консорциума связана с разработкой стандартов и технологий, позволяющих повысить переносимость сервисов, а также обеспечить возможность их повторного использования. Так как телефония является одной из самых перспективных областей для применения технологий создания интероперабельных систем, деятельность TINA-C оказывает большое влияние на формирование стандартов в области распределенных технологий (OMG CORBA, ITU, ANSA).

3.1.1 C++ ISP ORB

В качестве базиса для построения интероперабельной системы, расширяющей функциональность модуля Digital Multiplex Switch (DMS) для Nortel Networks, был выбран подход, предложенный консорциумом Object Management Group (OMG).

Деятельность консорциума OMG направлена на создание новых компьютерных технологий и стандартов. Консорциум объединяет более 200 компаний, занимающихся разработкой программного обеспечения. Архитектура компонентов промежуточного слоя Common Object Request Broker Architecture (CORBA) – одна из наиболее известных его разработок.

Технология брокеров объектных заявок позволяет приложениям, реализованным на разных платформах, взаимодействовать в неоднородных распределенных средах прозрачным по расположению объектов образом. В архитектуре CORBA брокер играет роль "общей шины" в глобальном пространстве объектов, выполняя операции нахождения объекта, его активации, передачи параметров и результатов. Взаимодействие между объектами строится по принципу клиент-сервер, посредником между которыми является брокер объектных заявок.

Объектная модель DMS включает такие понятия из проблемной области, как Call, Half Call, Party, Facility, Connection. Интерфейсы этих объектов определяются на языке OMG Interface Definition Language (IDL). Независимая спецификация интерфейсов позволяет внешним сервисам обращаться к реализациям объектов, выполняющимся в среде DMS. В свою очередь, сервисы, удовлетворяющие определенным спецификациям и функционирующие на других платформах, могут быть вызваны объектами из среды DMS. Прозрачность по расположению объектов достигается путем использования специальных посредников (статических – стабов и скелетонов, или динамических).

Протокол взаимодействия между брокерами General Inter-ORB Protocol (GIOP) является частью спецификации CORBA. Наиболее распространенной версией этого протокола является отображение GIOP на TCP/IP, называемое Internet Inter-ORB Protocol (IIOP) и также включенное в стандарт. Как уже говорилось выше, главный “телефонный” модуль, как правило, реализован на специализированной платформе и, следовательно, в его среде используется свой внутренний протокол взаимодействия. Поэтому реализация проекта по расширению функциональности модуля DMS включает использование двух брокеров. Брокер, расположенный непосредственно в среде DMS и использующий внутренний протокол, взаимодействует с внешними сервисами посредством другого брокера, использующего стандартный CORBA протокол. А специальный посредник (Bridge) преобразует сообщения из внешнего формата в формат внутренних сообщений DMS.

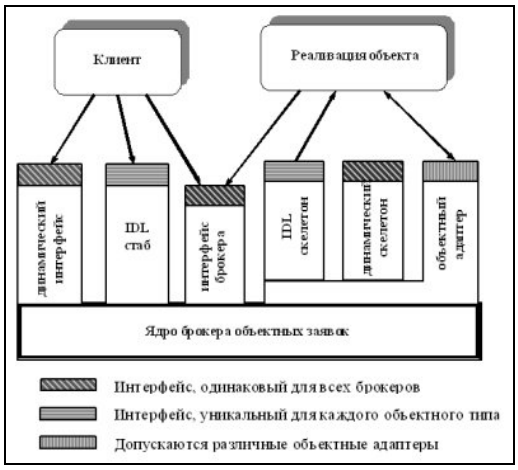

Рис. 1. Общая структура интерфейсов брокера объектных заявок

В качестве “внешнего” брокера был разработан C++ ISP ORB, удовлетворяющий

стандарту OMG CORBA 2.1 [CORBA97] (рис. 1). Реализация включает непосредственно библиотеку брокера и компилятор, реализующий полное отображение из IDL в C++. Брокер поддерживает работу как со статическими посредниками (стабами и скелетонами), генерируемыми компилятором, так и с динамическими интерфейсами (Dynamic Interface Invocation, Dynamic Skeleton Invocation).

Объекту в распределенном пространстве ставится в соответствие понятие объектной ссылки. Объектная ссылка инкапсулирует информацию о нахождении объекта с точки зрения конкретного брокера. Для передачи объектной ссылки между разными брокерами стандарт CORBA предусматривает преобразование в некоторое универсальное представление Interoperable Object Reference (IOR). Приложение имеет понятие об объектной ссылке в соответствии со стандартом отображения из языка IDL. Для языка C++ объектная ссылка это обычный указатель на объект. Таким образом работа с удаленными объектами ничем не отличается от работы с локальными, хотя в случае удаленного объекта указатель в действительности указывает не на сам объект, а на его посредника, передающего вызов брокеру для дальнейшей отработки. Но от приложения это скрыто, и можно говорить о прозрачности по расположению объектов.

Общий механизм выполнения запросов происходит по известной схеме удаленного вызова процедуры (Remote Procedure Call, RPC). В применении к объектно-ориентированному подходу можно говорить об объектном RPC (ORPC).

Код статических посредников, а также отображение типов и констант из IDL-спецификации порождается компилятором и известны уже на этапе компиляции. Благодаря поддержке динамических интерфейсов во время выполнения программы могут конструироваться запросы к объектам, интерфейсы которых не были известны на этапе компиляции.

Помимо базового механизма взаимодействия объектов на основе брокера объектных заявок, архитектура CORBA предусматривает использование ряда сервисов, интерфейсы которых также специфицированы в стандарте. В частности, для регистрации и поиска объектов в распределенном пространстве может быть использован сервис именования (Naming service). Любой объект, чтобы стать доступным брокеру и другим объектам, должен быть зарегистрирован в сервисе именования. С помощью этого сервиса приложение может по имени объекта получить его объектную ссылку. Далее эта объектная ссылка может быть передана как параметр другим объектам. Сервис именования является одним из основных сервисов, без которых использование брокера становится затруднительным.

В процессе реализации брокера ISP ORB был обнаружен ряд существенных недостатков отображения из языка спецификаций интерфейсов IDL в C++, затрудняющих работу программиста в рассматриваемой среде. Для решения проблемы повышения надежности программного обеспечения была предложена концепция оболочек-посредников, которая существенно упрощает программирование приложений в CORBA-среде. Важно отметить, что новый подход не является альтернативой стандарту. Система оболочек реализована в виде надстройки над произвольной системой управления объектами, удовлетворяющей стандарту CORBA 2.1. Подробнее этот подход будет рассмотрен в четвертом разделе статьи.

В процессе создания брокера ISP ORB были рассмотрены два подхода к реализации процедур преобразования данных (маршалинга), основанные на технике интерпретации и компиляции соответственно. С учетом известных достоинств и недостатков методов предпочтение было отдано механизму компиляции, обеспечивающему большую эффективность работы системы. Возможность использования техники интерпретации сохранилась для работы с данными, типы которых не известны на этапе компиляции, и для случаев, когда компактность системы имеет решающее значение.

Реализация брокера C++ ISP ORB удовлетворяет стандарту CORBA 2.1 и может быть использована независимо от рассматриваемого проекта. C++ ISP ORB поддерживает работу в нескольких режимах. Библиотека брокера может использоваться как в виде разделяемой библиотеки, так и непосредственно “собираться” вместе с приложением. Также брокер может работать как с использованием нитей (POSIX threads), так и без этого.

Бета-версия брокера свободно доступна и может быть получена по адресу http://www.ispras.ru/~dkv/isp_orb/download/ref_index.html.

3.1.2 Protel-2 ORB

Примером разработки брокера объектных заявок для специализированной среды модуля телекоммуникационной системы может служить брокер для языка Protel-2, предназначенный непосредственно для работы в среде модуля Digital Multiplex Switch (DMS). Язык Protel является внутренним языком программирования DMS.

Реализация брокера является распределенной относительно двух модулей – CM (Compute Module) и SDM, UNIX-системы, напрямую связанной с DMS и поддерживающей локальную систему сообщений (MTS). Эта распределенность прозрачна для сервисов, обращающихся к объектам DMS.

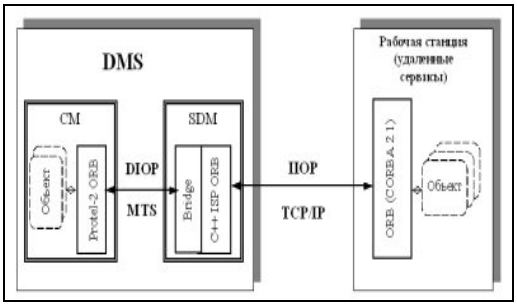

Рис. 2. Взаимодействие брокеров для языков C++ и Protel-2

Взаимодействие в среде DMS основано на собственном внутреннем протоколе передачи сообщений – DMS MTS. Специальный посредник (Bridge) преобразует сообщения из стандартного формата IIOP TCP/IP в DMS MTS и обратно (рис. 2). Для этого стандартные сообщения в формате General Inter-ORB Protocol (GIOP) “обертываются” дополнительными заголовками, в соответствии с типом сообщений. Новый протокол получил название DIOP (DMS Inter-ORB Protocol).

Важно отметить, что реализация брокера “многопроцессная”, то есть запросы выполняются не последовательно, а параллельно в разных процессах SOS, что позволяет увеличить производительность и существенно уменьшить время реакции системы на запросы. Главный процесс брокера создается вместе с запуском DMS и существует постоянно. При установлении связи с “внешним” брокером создаются процессы времени взаимодействия, которые завершаются при разрыве соединения. При получении брокером запросов создаются отдельные серверные процессы для выполнения каждого запроса.

Взаимодействие между процессами основано на работе с сообщениями в формате операционной системы SOS. Как правило, сообщение содержит идентификатор типа сообщения, указание, по какому адресу должен быть отправлен ответ, и ряд параметров. В большинстве случаев параметром сообщения является указатель на память, содержащую некоторые данные. Главный процесс брокера использует единственный “почтовый ящик”, куда поступают все запросы. Основное назначение этого процесса – диспетчеризация приходящих сообщений, создание серверных и коммуникационных процессов, и передача сообщений между ними. Все эти действия не требуют больших временных затрат и обращений к другим сервисам и, следовательно, могут быть выполнены последовательно.

Коммуникационные процессы существуют в паре: посылающему процессу соответствует получающий (receiving/sending processes). Создаются эти процессы либо при обращении клиентского процесса к удаленному объекту, либо при установлении связи с некоторым “внешним” брокером.

Серверный процесс создается главным процессом брокера на базе модуля, содержащего скелетон вызываемого объекта. После создания серверный процесс сразу же получает соответствующее сообщение от главного процесса, декодирует его содержание (IIOP-сообщение) и вызывает необходимый метод серверного объекта. Метод серверного объекта выполняется непосредственно в этом же процессе. После завершения выполнения метода серверный процесс записывает результат в предоставленную главным процессом память и посылает ему соответствующее сообщение и завершается.

В терминах языка Protel работа с брокером включает использование модулей нескольких типов. Во-первых, это модули, содержащие функциональность самого брокера (главный процесс, процесс-“слушатель”, процесс-“получатель” сообщений, процесс-“отправитель” и библиотека брокера). Во-вторых, это модули, сгенерированные компилятором, содержащие код посредников и отображения типов и констант, в соответствии с отображением из IDL и Protel-2. И, в-третьих, это пользовательские модули, непосредственно реализующие прикладную задачу. В свою очередь, пользовательские модули могут использовать библиотечный модуль брокера для доступа к функциональности самого

брокера и модули, сгенерированные компилятором и позволяющие манипулировать объектами, опубликованными в IDL-спецификации.

Для реализации брокера было разработано полное отображение из IDL в Protel-2 (объектный Protel), поддерживаемое соответствующим компилятором. Язык Protel является корпоративным языком Nortel Networks, и спецификация CORBA 2.1 не предусматривает стандартного отображения из IDL в Protel.

Как правило, одна IDL-спецификация отображается в один модуль на языке Protel-2, содержащий реализации стабов и скелетонов, а также все необходимые определения.

Модель языка Protel-2 достаточно сильно отличается от языка IDL, поэтому отображение включает ряд трюков. Например, Protel-2 не поддерживает вложенные области видимости и составные имена, используемые в IDL. Поэтому схема отображения имен из IDL в Protel реализована по примеру отображения из IDL в C. Составное имя в IDL отображается в “длинное” имя в Protel, которое конструируется из последовательности имен вложенных областей видмости. Также Protel поддерживает одиночную схему наследования объектов, тогда как IDL предполагает множественное наследование. И, следовательно, отображение включает преобразование из множественной модели наследования в одиночную. Такое преобразование неизбежно приводит к трансформации некоторых семантических зависимостей. Предложенный алгоритм позволяет сократить количество новых зависимостей в графе наследования.

При разработке отображения из IDL в Protel-2 внимание было уделено исследованию проблем адаптации унаследованного кода в CORBA-среду. Использование специальных директив препроцессору – прагм (#pragma) позволило, не расширяя базового набора конструкций IDL, уточнять отображение в соответствии с имеющимся кодом реализации. Более подробно предложенная система прагм и возникающие проблемы при адаптации унаследованного кода будут обсуждены в четвертом разделе.

3.2. Информационные системы в биологии (BiomaxRS)

Наряду с нарастанием объемов обрабатываемой информации, усложнением структуры информационных элементов (графика, мультимедиа, Java-приложения), одной из самых заметных тенденций, характерных для современных информационных систем, является использование распределенных технологий для объединения различных информационных систем в гетерогенные сети с унифицированным пользовательским интерфейсом на основе технологии "клиент-сервер". Другой характерной особенностью является использование каналов Internet и WWW-технологий для доступа к информационным системам и, как следствие, возможность использовать стандартные сетвые клиенты (browsers).

Исследования в биологии связаны с анализом большого количество фактической информации. Эта информация хранится, как правило, в текстовом виде в различных биологических банках данных (например, MEDLINE, GenBank). С развитием Internet и WWW-технологий появилась возможность удобного удаленного доступа к этой информации. А технологии интероперабельных систем позволяют говорить о совместном использовании различных информационных систем и их функционировании в неоднородных информационных контекстах.

Информационно-поисковая система для Biomax Informatics GmbH (BiomaxRS – Biomax Retrieval System) предназначена для работы с множеством биологических банков данных. Система предоставляет средства поиска и манипулирования банками данных (создание, удаление, наполнение, редактирование, индексирование), а так же средства администрирования системы в целом.

BiomaxRS является распределенной объектной системой, построенной на базе архитектуры CORBA, с использованием C++ ISP ORB. Основные функциональные элементы системы имеют API, специфицированные на OMG IDL. Языком реализации ядра системы является C++, но технология CORBA предоставляет возможность работать с клиентами, реализованными на других платформах. Для управления объектами используются стандартные CORBA-сервисы. В частности, для поиска объектов в распределенном пространстве используется сервис именования (Naming service).

Архитектура CORBA позволяет естественным образом работать с распределенными ресурсами. Различные банки данных могут храниться на разных машинах, что позволяет говорить о распараллеливании их обработки (поиск,

индексирование).

Система BiomaxRS поддерживает работу с различными форматами биологических банков данных. В качестве средства нормализации и семантической интероперабельности данных используется язык eXtensible Mark-up Language (XML), работа по стандартизации которого происходит под эгидой консорциума W3 [XML99].

Классические информационно-поисковые системы используют текстовое представление данных и, следовательно, классические методы поиска. Наряду с внедрением новых типов данных (графика, мультимедиа) рассматриваются новые модели документов. Широкие возможности для организации поиска, обработки и передачи между приложениями слабоструктурированных данных предоставляет язык разметки XML. Документы на языке XML представляют собой файлы ASCII, содержащие текст и теги, идентифицирующие структуры внутри текста. Для информационных систем существенно то, что XML удобен для кодирования собственных типов данных с помощью общепринятых файловых форматов. Средства XML позволяют создавать новые языки для описания структур данных в конкретных предметных областях (химия, математика, биология). На основе XML разрабатываются различные средства манипулирования данными, например, язык запросов XQL (аналог SQL для XML данных), язык XSL, позволяющий специфицировать отображения XML в HTML и другие представления.

Без использования единого формата данных (например, языка XML) система, работающая с разными форматами данных, вынуждена использовать разные алгоритмы для извлечения необходимой информации из текстов. Используя XML, система работает только с информацией, представленной в виде XML-документов, имеющих определенную структуру. Это существенно упрощает обработку данных. При этом семантически одинаковые части различных форматов выделяются на этапе перевода в XML и преобразуются к единому виду. Например, информация о датах, представляемая в разных форматах данных по-разному (например, 1-JAN-98, 01/01/1998 или 1998, 1 january), в XML-документе имеет единый вид:

```
<DATA day = "01" month = "01" year = "1998" />.
```

Использование XML в качестве внутреннего представления данных объясняется еще и тем, что в скором будущем XML обещает стать международным стандартом. Консорциум W3 осуществляет активную деятельность, направленную на стабилизацию спецификаций XML, что вдохновляет разработчиков на создание приложений на его основе. Исходя из этих прогнозов, XML может стать средством обмена данными с информационными системами, внешними по отношению к BiomaxRS, но также использующими XML. Кроме того, уже сейчас существует множество инструментальных средств и технологий для работы с XML-документами. Например, для простейшей визуализации XML-документа может использоваться стандартный браузер Internet Explorer 5.0.

В рамках объектной системы для манипулирования XML-документами удобно также использовать объектно-ориентированный подход. То есть компоненты системы взаимодействуют с XML-документом как с объектом. Соответствующие методы этого объекта предоставляют возможность создавать, удалять и редактировать части документа. Существует стандарт отображения XML структуры на объектную модель – Document Object Model (DOM), также поддерживаемый консорциумом W3. Имея IDL-интерфейсы (а спецификация DOM использует именно OMG IDL), такие объекты могут обрабатываться и в распределенной среде посредством CORBA.

Поскольку XML основан на текстах в формате ASCII, он может использоваться для более эффективного обмена данными по сравнению с бинарными технологиями обработки сообщений (CORBA, DCOM). В частности, в подходе CORBA естественным способом передачи объекта является передача его “по ссылке” (то есть, реально передается объектная ссылка, а не сам объект). Передача объекта “по значению” требует дополнительных усилий, например использования системных сервисов (Persistent Object Service или Externalization Service). Сочетание XML и CORBA позволяет продуктивно использовать возможности обеих технологий для создания более гибкой распределенной среды. Единственным спорным моментом при использовании XML для управления информационным наполнением системы является необходимость предварительного переформатирования данных в XML-формат. Но преимущества использования

XML-документов окупают этот недостаток.

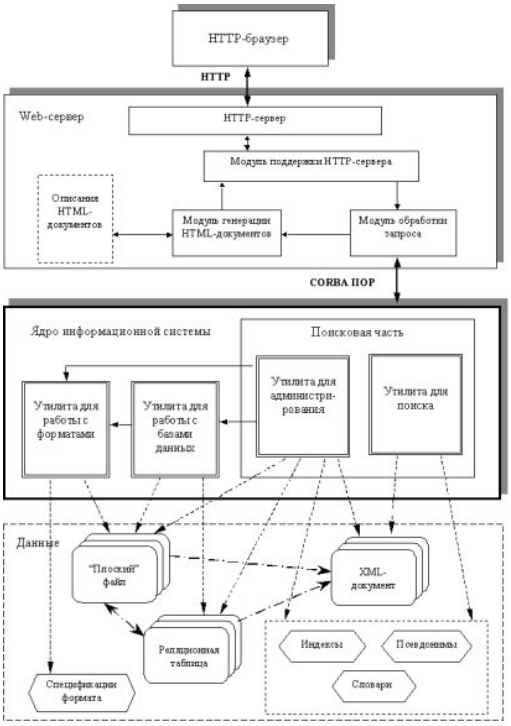

Рис. 3. Общая архитектура системы

Общение пользователя с информационной системой (рис. 3) происходит с помощью стандартного HTML-браузера, который получает HTML-документы от web-сервера, связанного с ядром системы. WWW-интерфейс предоставляется не только конечным пользователям, использующими поисковые возможности системы, но и для администрирования системы. Web-сервер взаимодействует с ядром информационной системы посредством CORBA и создает HTML-документы на основе результатов запроса. Ядро системы разработано в рамках технологии CORBA и имеет распределенную архитектуру. В соответствии с их функциональностью, в ядре системы можно выделить три части: утилита для работы с форматами (Format Tool), утилита для работы с базами данных (Data Base Tool) и поисковая часть системы (Retrieval System).

В рамках системы используется несколько внутренних представлений данных. Это, во-первых, XML-документы, получаемые из исходных “плоских” файлов с биологическими данными, а так же таблицы реляционных баз данных. Поисковая часть системы работает также с индексами, словарями и списками псевдонимов.

Основная функциональность системы связана с поиском и просмотром биологических данных и, следовательно, основная задача – это показ требуемого XML-документа пользователю. С точки зрения внутренних компонент системы, XML-документ- это объект, имеющий IDL-интерфейс (в соответствии со спецификацией DOM), и работа с ним осуществляется посредством CORBA. Для передачи такого объекта web-серверу (именно по значению, а не по ссылке) используется текстовое представление XML-документа. А web-сервер, в свою очередь, используя анализатор XML, преобразует XML-документ в HTML-страницу.

Рассмотрим подробнее составные части ядра системы.

3.2.1 Утилита для работы с форматами (Format Tool)

Эта часть системы обеспечивает работу с различными форматами данных. В качестве исходной точки для информационного наполнения системы служат так называемые “плоские” файлы, получаемые из других биологических банков данных. Как правило, каждый банк использует собственный формат для представления данных. В системе BiomaxRS для каждого такого формата используется специальный транслятор, преобразующий исходный файл в XML-документ определенной структуры. Добавление в систему нового формата связано с добавлением нового транслятора, полученного с помощью компилятора форматов. На вход компилятору подается описание формата, а на выходе получается исходный код на C++ соответствующего транслятора. Для спецификации форматов был разработан язык, позволяющий описывать требуемую структуру XML-документа и синтаксические правила разбора исходного файла.

3.2.2 Утилита для работы с базами данных (Data Base Tool)

Для хранения и редактирования данных в системе используются реляционные базы данных (Oracle, MySQL). Основное назначение утилиты – запись исходного “плоского” файла в таблицу базы данных и обратно, а также редактирование данных в таблицах. Для записи в таблицу исходный файл разбивается на отдельные поля с помощью соответствующего транслятора из Format Tool. После записи в таблицу доступ к данным может осуществляться посредством стандартных SQL-запросов.

Преобразование данных из “плоского” файла в таблицу и обратно взаимно однозначно. То есть, если не было произведено никакого редактирования таблицы, то полученный из нее “плоский” файл будет совпадать с исходным. Таким образом, утилита для работы с базами данных может служить “мостом” между двумя разными представлениями данных, используемыми в системе. Также существует процедура получения XML-документа из таблицы базы данных.

3.2.3 Поисковая часть системы (Retrieval System)

Это основная часть информационной системы, предназначенная для поиска и организации доступа к биологической информации, хранимой в виде XML-документов, а также для администрирования системы в целом. В соответствии с этой функциональностью и категориями пользователей Retrieval System разделяется на административную и пользовательскую (поисковую) части.

3.2.4 Утилита для администрирования системы (Admin Tool)

Основное назначение этого инструментального средства – администрирование внутренних данных системы для организации последующего поиска. Admin Tool работает с данными, представленными в виде банков, которые, в свою очередь, могут объединяться в семантически связанные группы (как правило, группа соответствует одному формату исходных данных). Каждый банк данных состоит из файлов, являющихся XML-документами и множества индексов, построенных на основе сбалансированного бинарного дерева. Admin Tool позволяет создавать и удалять банки данных, объединять их в группы, заполнять банки данных файлами, строить и уничтожать индексы.

Для более гибкой и универсальной схемы работы с индексами используется механизм дополнительных имен – псевдонимов (aliases). Набор псевдонимов редактируется и поддерживается администратором системы. Будучи одним для всей системы, для всех банков данных механизм псевдонимов дает возможность использовать одно имя при спецификации поиска по разным банкам данных с использованием разных индексов.

Особым видом индексов являются индексы по перекрестным ссылкам между банками данных. Множество косвенных ссылок представляет собой граф со множеством связей. При поиске бывает полезным учитывать не все имеющиеся связи, а только некоторое подмножество. Возможности редактирования этого подмножества предоставляется Admin Tool.

Для удобства организации поиска поддерживается множество словарей, хранящих именованные запросы. Admin Tool предоставляет средства редактирования этих словарей.

3.2.5 Утилита для поиска (Search Tool)

Эта часть системы предназначена непосредственно для поиска и доступа к биологической информации.

Основной способ поиска – это поиск по образцу в текстовых элементах документа. При задании образца можно строить регулярные выражения, используя символы ‘*’ (ноль или больше символов) и ‘?’ (ровно один символ). Например, ‘Verte*brat?’, ‘Primates’ или ‘*malia’. Для полей, содержащих имена авторов и даты, реализован специализированный поиск. Вместо одного регулярного выражения для авторов указываются отдельно образцы для поиска по фамилии и инициалам соответственно, а для дат – по дню, месяцу и году. Например, ‘name = Kitamura; initials = *;’. Также система поддерживает специализированный поиск по датам и числовым элементам. Для этих данных в спецификации запроса можно использовать операции сравнения.

В каждом запросе указывается один или несколько банков данных, конкретный индекс, по которому будет производиться поиск, и образец для поиска. Несколько подзапросов могут объединяться в один запрос с помощью логических связок (and, or и not). Особой поисковой операцией является поиск по перекрестным ссылкам между банками данных. Для спецификации запросов разработан специальный язык.

Результатом любого поиска является набор записей из некоторого множества банков данных, удовлетворяющих запросу. По умолчанию пользователю доступны только идентификаторы записей, по которым затем можно получить для просмотра их содержание. Помимо этого существует возможность указать при спецификации запроса имена тех полей, которые вместе с идентификаторами записей будут сразу доступны при просмотре результатов поиска.

Также система позволяет уточнять запросы, строя новые запросы над результатами предыдущего поиска.

3.2.6 Пользовательские интерфейсы

В BiomaxRS пользовательские интерфейсы строятся как клиентская часть CORBA-приложения, могут быть реализованы на произвольной платформе и быть удаленными по отношению к ядру системы. Интерфейсы предоставляют пользователям (локальным или удаленным) доступ к необходимым функциональным возможностям информационной системы.

Для информационных систем характерно повышенное требование к качеству пользовательских интерфейсов. Развитие сетевых технологий и концепции клиент-сервер предоставляет богатые возможности для построения пользовательских интерфейсов, используя стандартные средства сети Internet и WWW-технологий.

При разработке любого интерфейса, основанного на гипертекстовых технологиях, возникает вопрос о генерации web-сервером представляемых пользователю HTML-документов. В BiomaxRS используется архитектура web-сервера, предполагающая логическое деление на несколько уровней: некоторый конкретный HTTP-сервер и модуль, обеспечивающий взаимодействие с ним, модуль обработки запросов, представляющий собой уровень бизнес-логики, модуль генерации HTML-документов и некоторый внешний архив, содержащий описание пользовательского интерфейса. Элементами этого архива являются содержательное описание для каждого конкретного HTML-документа и единое описание графического оформления для всех HTML-документов системы. Описание осуществляется на языке EH (Extended Hypertext – расширенный гипертекст), специально разработанном для решения проблемы генерации содержимого HTML-документов. Документ на языке EH представляет собой некоторый шаблон, в который подставляются динамические данные, полученные от модуля обработки запроса. Динамические данные при этом могут иметь сколь угодно сложную структуру. Языковые средства EH позволяют производить сложную обработку динамических данных при помощи операторов циклов и условных операторов. Макроопределения и макроподстановки дают широкие возможности для переиспользования кода, а также для разделения оформления и содержания документов.

В рассматриваемой информационной системе web-сервер основан на универсальном HTTP-сервере Apache. Основная часть web-сервера представляет собой FastCGI-приложение, взаимодействующее с ядром информационной системой посредством технологии CORBA. В соответствии с предложенной архитектурой, FastCGI-приложение состоит из модуля поддержки FastCGI, модуля обработки запроса и модуля генерации HTML.

Web-сервер предоставляет доступ к системе посредством так называемых сессий. В начале работы каждый пользователь осуществляет авторизацию, после чего web-сервер заводит для пользователя отдельную сессию. Состояние всех сессий инкапсулировано в модуле обработки запроса.

Модуль обработки запроса фактически представляет собой некоторый автомат, управляемый HTTP-запросами. В ответ на каждый запрос модуль применяет указанную операцию к указанной сессии, меняя состояние последней. После выполнения операции, ее результаты используются для генерации и выдачи указанного документа. Поведение автомата определяется исключительно EH-описаниями и может быть изменено в динамике без перезапуска web-сервера.

После завершения выполнения запроса модулю генерации HTML передается указание на вывод документа из файла, содержащего определенный шаблон. Имя файла определяется модулем обработки запроса и передается в виде параметра. Модуль генерации HTML загружает указанный документ и осуществляет его вывод с использованием макроопределений языка EH, соответствующих текущей сессии.

Помимо гипертекстового интерфейса система предоставляет пользователям интерфейс, обладающий той же функциональностью, но работающий в режиме командной строки.

4.Научные исследования и разработки

4.1. Методология разработки распределенных приложений на основе модели оболочки

Создание унифицированной среды разработки и поддержки времени выполнения ориентировано на разработку распределенных приложений с высокими требованиями к эффективности, и, прежде всего, на приложения реального времени. В основе подхода к созданию такой среды лежат механизмы адаптации,

обеспечивающие гибкие возможности оптимизации взаимодействий удаленных объектов.

На концептуальном уровне множество всех объектов распределенного глобального пространства представляется покрытым множеством специального вида подмножеств – оболочек. Каждая оболочка содержит некоторую совокупность семантически взаимосвязанных объектов.

Любое взаимодействие двух объектов поддерживается прозрачным образом оболочками, содержащими эти объекты. Любой объект вызывает метод другого объекта независимо от их взаимного расположения (локального или удаленного), от свойств взаимодействующих оболочек (объектной модели, поддерживаемой конкретной оболочкой, организации мультидоступа к объектам оболочки, методов поддержки секретности внутри оболочки и т.п.) и от способов коммуникации между оболочками (может использоваться IIOP CORBA 2.0 [CORBA98] или любой другой специфический протокол).

В жизни объекта различают две фазы: проектирования и выполнения. Оболочки поддерживают и организуют взаимное согласование объектов на обеих фазах. То есть, оболочки являются одновременно и средой проектирования объектов, и средой выполнения.

Поскольку оболочки ответственны за поддержку согласованного прозрачного взаимодействия объектов, сами оболочки нуждаются в некоторой дисциплине согласования. В принципе, мы могли бы представить себе однородный мир оболочек, отвечающих некоторому одному стандарту организации взаимодействия между объектами, например, CORBA [CORBA98]. Мы, однако, хотели бы предложить более разнообразный и богатый мир оболочечных моделей, обладающих не только разными свойствами, но и разными свойствами описания этих свойств, т.е. метасвойствами. Согласование оболочек может происходить как на стадии проектирования объектов (статическое согласование оболочек), так и на стадии выполнения (адаптивное динамическое согласование).

Для гибкого согласования оболочек естественно использовать технику метаобъектного контроля [KRB91]. Однако традиционно метаобъектный контроль используется либо на фазе проектирования, либо на фазе выполнения.

В первом случае [Chi96] мы получаем высоко эффективный, но монолитный код, который уже не может быть изменен во время выполнения некоторым систематическим образом. В процессе распределенной обработки данных зачастую возникают ситуации, требующие проведения существенных изменений схем согласования оболочек. Например, это требуется в случаях миграции объекта (объект может переместиться в оболочку, имеющую другие свойства) или изменения свойств самой оболочки (например, изменения требований секретности в оболочке-предприятии). При второй альтернативе платой за исключительную гибкость (неограниченная возможность динамической трансформации) управления вызовами методов объектов во время выполнения является потеря производительности, по крайней мере, в десятичный порядок [ZC95, ZC96], в результате использования интерпретации.

Наш подход в использовании метаобъектного контроля [IZKN96, IDZ97] есть некоторый компромисс между двумя этими альтернативами. Его идея заключается в следующем. На стадии проектирования мы подготавливаем различные варианты возможного метаконтроля для организации взаимодействия объектов (возможно и его отсутствие). И тем самым получаем высоко эффективные варианты кода. На фазе же выполнения используется тот или иной вариант. Более того, текущий вариант во время выполнения может быть заменен другим (динамическое согласование оболочек).

Подключение прикладного объекта к метаконтексту, состоящему из метаобъектов, осуществляется с помощью специального метаобъекта, названного нами интерфейсным объектом (ИО). В терминах модели “клиент-сервер”, некоторый клиентский объект может вызывать метод серверного объекта либо непосредственно, не подключаясь к метаконтексту, либо через соответствующий ИО, выполняющий роль посредника. Этот посредник осуществляет пред- и пост- вызовы метаобъектов. Отметим, что для того, чтобы подключение ИО было прозрачным для объектов приложений, в этих приложениях должен использоваться некоторый механизм позднего связывания объектов.

Будучи средой обитания объектов, оболочка содержит набор ортогональных метасервисов (семантически связанные

коллекции метаобъектов). Предположение об ортогональности чрезвычайно важно и предоставляет следующие возможности:

- естественным образом подключать прикладные объекты к различным функциональным элементам метаконтекста прозрачно для прикладных объектов;
- развивать независимо различные функциональности, т.е. метасервисы;
- добавлять новые метасервисы, т.е. проводить метанаращивание оболочек без изменения уже существующих метасервисов.

Подчеркнем еще раз, что подключение к метаконтексту осуществляется прозрачно как для клиентских, так и для серверных объектов. Организация метаконтроля не затрагивает реализации приложений (клиента и сервера). В этом смысле можно говорить, что любой объект приложений рассматривается как “черный ящик”.

В случае удаленных объектов происходит естественное расщепление ИО на ИО клиента и ИО сервера (аналогия пары “стаб – скелетон” в архитектуре CORBA). Причем метаконтроль, производимый в этих двух ИО, может быть частично или полностью независимым. Когда нет полной независимости, необходимое согласование клиентского и серверного ИО обеспечивается соответствующими оболочками, в которых эти ИО находятся.

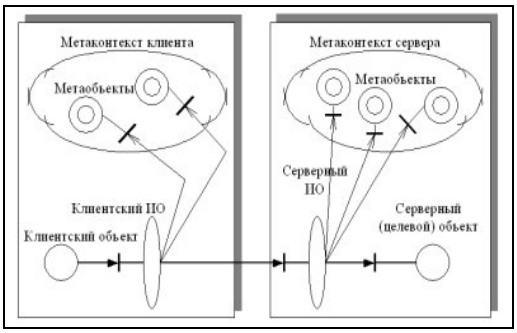

Рис. 4. Схема организации метаконтроля

На рисунке 4 приведена схема организации работы двух взаимодействующих объектов как через клиентский, так и через серверный ИО.

Для проектирования используется набор из трех языков: языка реализации, языка спецификаций и языка описания метаобъектных связей. Поскольку мы хотим иметь возможность получать надежный, высоко эффективный код, язык реализации должен быть объектно-ориентированным языком со строгой типизацией. В качестве языка спецификаций для описания типов объектов целесообразно использовать IDL. Такой выбор гарантирует совместимость с промышленным стандартом CORBA. Для описания различных вариантов подключения целевых объектов к метаконтексту, т.е. для описания наполнения интерфейсных объектов, предлагается некоторый язык шаблонов для пошаговой спецификации метаконтроля с рабочим названием – TL (Template Language).

Итак, предлагаемый подход к созданию единой среды конструирования распределенных систем базируется на нескольких основных идеях: ортогональной декомпозиции системы, метаобъектном контроле с использованием развитой техники объектных посредников, согласовании распределенных компонентов – оболочек.

4.2. Надстройка над стандартным отображением из CORBA IDL в C++

В процессе разработки брокера объектных заявок C++ ISP ORB был обнаружен ряд существенных недостатков отображения из языка спецификаций интерфейсов IDL в язык реализации C++, затрудняющих работу программиста в рассматриваемой среде.

Например, для работы с объектными ссылками для каждого интерфейса, скажем A, определенного в IDL-спецификации, стандартное отображение предусматривает два типа ссылок – A_ptr и A_var, совершенно различных по схеме работы с памятью. По сути, A_ptr является просто указателем, а A_var – специальный объект, имеющий конструктор и деструктор, владеющий памятью, на которую указывает. То есть, уничтожение объекта A_var обязательно влечет за собой и уничтожение памяти, на которую он ссылается. При этом значение переменной любого из этих типов может быть присвоено переменной другого типа, определено преобразование типа A_var в A_ptr. Во всех этих случаях дублирование памяти не производится. Не предусмотрено никаких средств контроля подобных манипуляций с данными этих типов. Это и служит весьма распространенным источником проблем при работе с объектными ссылками.

Правила передачи параметров и результатов, предполагаемые стандартом, существенно зависят от свойств передаваемого типа и направления передачи параметра (IN, INOUT, OUT,

RESULT). Простыми их можно назвать только для примитивных типов данных. Для составных же типов правила манипулирования памятью для передаваемых параметров зависят от свойств конкретного типа (например, переменной или постоянной длины).

Определенным недостатком можно считать требование стандарта, запрещающее передавать клиенту и серверу в качестве параметра или результата нулевой указатель. Даже если при выполнении метода сервера возникла исключительная ситуация, все OUT параметры и результаты должны быть чем-то заполнены.

Для каждого составного типа `T` стандарт предполагает использование дополнительного типа `T_var`. Объекты этого типа играют роль “умных” указателей, владеющих памятью, на которую ссылаются. Допустимые неявные приведения между объектами типов `T*` и `T_var`, как и в случае с объектными ссылками, провоцируют некорректную работу с памятью. Безобидное присваивание строки вида

```
CORBA::String_var var = “some string”;
```

влечет попытку удаления статической памяти при уничтожении переменной `var`.

Предлагаемый подход также основан на использовании специальных оболочек-посредников для данных составных типов. Но в отличие от посредников типа T_var, оболочки предполагают более последовательную и надежную работу с памятью. Пользователь может себе представлять оболочки как “умные” контейнеры, содержащие данные соответствующего составного типа. Основой концепции работы с памятью здесь является тезис о том, что память, содержащаяся в оболочках, *никогда* не доступна для управления пользователем. То есть оболочки сами создают память под данные и уничтожают ее. Пользователь же лишь может указать, откуда скопировать данные в эту память, может посмотреть ее содержимое и может изменять значения элементов составных данных, хранимых в оболочке.

Все неявные операции работы с памятью (присваивания и инициализации) заключаются в глубоком копировании данных. Неявное приведение из типа `T_var` в тип `T*` запрещено, так как семантика такого преобразования не вполне ясна. Для присваивания из оболочки в переменную соответствующего базового типа необходимо использовать тот или иной явный метод. Платя некоторой потерей эффективности, пользователь может использовать оболочки и не задумываться о правильности работы с памятью. При работе с ORB’ом пользователю предлагается проводить все необходимые манипуляции с данными только через подобных посредников. В частности, только с их помощью организуется передача параметров методов и получении результатов для составных типов.

Тип данных	In	Inout	Out	Return
Простой тип (S)	S	S &	S &	S
Составной тип (T)	const T_cvt &	T_cvt &	T_cvt &	T_cvt

Рис. 5. Предлагаемое отображение для типов параметров и результатов

Сама схема передачи параметров с использованием оболочек становится предельно простой (рис. 5). При этом способ передачи параметров не зависит от свойств типа.

Для повышения эффективности предусмотрен ряд явных операций, позволяющих оптимизировать работу с памятью там, где это действительно необходимо. Например, можно записать в оболочку указатель, не передавая при этом права на владение памятью. В этом случае уничтожение оболочки не повлечет за собой удаления памяти, которая может быть, например, статической. С помощью других явных операций пользователь может использовать для чтения память, управляемую оболочкой, без дополнительного копирования.

Важной особенностью оболочек является то, что их не обязательно явным образом инициализировать. Изначально они уже содержат данные, заполненные по умолчанию: все элементы данных, имеющие конструктор, заполнены с помощью конструктора по умолчанию, остальные – просто обнулены. Это гарантирует, в частности, возможность возврата результатов (в том числе через INOUT и OUT параметры) методов без их явного заполнения.

В отличие от объектов типа T_var, которые до явной инициализации содержат нулевой указатель, оболочки всегда находится в корректном состоянии. При первом обращении к оболочке, которой не было присвоено никакого значения, происходит инициализация данных по умолчанию в соответствии с внутренним типом.

В случае использования до явного заполнения значения оболочки, содержащей данные типа объединения (union), используется соглашение о том, что начальным значением дескриптора объединения является либо значение по умолчанию (default), указанное в IDL спецификации для этого типа, либо, если таковое умолчание не было определено, используется первое определенное в IDL спецификации значение дескриптора (схема, используемая при инициализации объединений в C++ – заполняется именно первое поле). В качестве данных объединение содержит данные того типа, который соответствует выбранному указанным образом дескриптору, и заполнены также по умолчанию, используемому для этого типа. В стандартном же отображении, до явного присваивания объединение находится в некорректном состоянии, и в принципе запрещено использовать дескриптор объединения без его явной предварительной инициализации.

Для контроля состояния оболочки предусмотрен специальный метод is_nil (). Он возвращает истину, если оболочка так и не была проинициализирована явно, и ложь в противном случае.

Предложенная концепция оболочек-посредников решает проблему повышения надежности программного обеспечения и упрощает программирование с использованием этого отображения. Важно отметить, что использование оболочек не является альтернативой стандартного отображения, а является его дополнением. То есть новый подход реализован в виде некоторой надстройки над произвольной системой управления объектами, удовлетворяющей стандарту CORBA 2.1.

4.3. Маршалинг данных в гетерогенных средах

В процессе тестирования брокера объектных заявок C++ ISP ORB было замечено, что при передаче данных между машинами, объединенными локальной сетью, подавляющая часть времени тратится на преобразование данных в формат, пригодный для передачи по сети, и обратно. В результате исследований было предложено несколько подходов к оптимизации процедур преобразования данных (маршалинга).

В распределенных неоднородных системах неизбежно возникает задача преобразования данных, связанного с использованием различных платформ и/или передачей данных по сети. Процедура преобразования данных из одного формата в другой получила название м*аршалинга*. Маршалинг включает линеаризацию сложных структур данных, учитывает порядок байтов, ту или иную стратегию выравнивания (значения базовых типов выравниваются на границу кратную их размеру в байтах; выравнивание всех границ на максимально возможный размер данных).

В распределенных технологиях формат промежуточного, сетевого представления данных также является частью стандарта. Например, стандарт CORBA включает формат уровня представления данных Common Data Representation (CDR).

В связи с увеличением быстродействия современных компьютерных сетей время маршалинга во многом стало определять скорость взаимодействия. Поэтому проблемы оптимизации форматов и построения эффективных протоколов передачи данных становятся ключевыми. Возможны различные подходы к повышению эффективности протоколов. Например, использование оптимальных по размеру данных и способу интерпретации схем кодирования. Ярким примером деятельности в этой направлении является ситуация с языком описания структурной информации Abstract Syntax Notation (ASN.1). Его использование для спецификаций протоколов передачи данных не получило широкого применения из-за недостаточно эффективной схемы кодирования Basic Encoding Rules (BER), изначально разработанной для ASN.1. Поэтому был предложен ряд альтернативных схем – Packed Encoding Rules (PER), Efficient Packed Encoding Rules (EPER), Lightweight Encoding Rules (LWER), более оптимальных как по размеру закодированной информации, так и по скорости кодирования/декодирования, что существенно увеличило популярность ASN.1. Другой подход связан с различными вариантами реализации самих процедур маршалинга: использование различных техник (компиляция, интерпретация), объединение атомарных операций на этапе компиляции протоколов, генерация процедур, оптимальных по размеру кода и времени выполнения в соответствии с результатами профилирования программы. Также рассматриваются варианты реализации эффективных базовых процедур маршалинга на аппаратном уровне.

На сегодняшний день возможность корректного взаимодействия программных систем в неоднородных средах

определяется использованием формальных языков спецификаций для построения надежных и верифицируемых протоколов. Процедуры маршалинга получаются по спецификациям с помощью средств кодогенерации. Но традиционно наиболее критичный по времени исполнения код программируется вручную. Сегодня возникает задача *автоматического* порождения эффективных и надежных протоколов передачи данных. Помимо надежности и верифицируемости получаемого автоматически кода, использование формальных языков спецификаций и применение техники компиляции позволяет еще на стадии анализа синтаксического дерева спецификации осуществлять ряд предварительных вычислений. Например, для данных, имеющих фиксированную длину, может быть вычислен размер буфера, необходимый для их размещения. При этом учитывается структура типа и соответствующие правила выравнивания внутренних полей. Применяемая таким образом техника смешанных вычислений предоставляет дополнительные возможности оптимизации процедур маршаллинга.

На примере реализации брокера C++ ISP ORB были рассмотрены различные подходы к реализации процедур маршалинга. Наиболее общим подходом к реализации процедур маршалинга для составных типов является выбор между возможностями интерпретации и компиляции (в сочетании с inline-подстановками). И интерпретация, и компиляция имеют свои хорошо известные достоинства и недостатки. Интерпретируемый код компактен, но сама процедура интерпретации достаточно медленна. Выполнение скомпилированного кода намного быстрее, но при этом размер кода может существенно увеличиться.

В контексте реализации брокера C++ ISP ORB интерпретация подразумевает процедуру рекурсивного обхода некоторой универсальной структуры (называемой TypeCode), которая описывает внутреннее строение составных данных. В случае компиляции для каждого составного типа компилятором порождаются типизированные процедуры маршалинга. Эксперименты с C++ ISP ORB показали, что время выполнения скомпилированных процедур маршалинга в 3-4 раза меньше времени, требуемого процедурой интерпретации (в зависимости от сложности типа). Использование скомпилированных процедур маршалинга дает наибольший выигрыш по времени при работе с данными глубокой вложенности. При маршалинге таких данных интерпретатор превращается в достаточно длинную последовательность рекурсивных вызовов. В то же время скомпилированные процедуры маршалинга, с учетом использования inline-подстановок, позволяют практически избежать накладных расходов на дополнительные вызовы процедур. При этом размер объектного кода за счет применения типизированных процедур маршалинга увеличивается в среднем на 30%.

Встречаются ситуации, когда использование определенной техники маршалинга неизбежно. Например, с помощью процедуры интерпретации можно “маршалировать” данные, типы которых не были известны в момент компиляции и, следовательно, для них просто не существует скомпилированных процедур маршалинга. В свою очередь, скомпилированный код более гибок по отношению к размещению данных в памяти. Если на некоторой платформе используются оптимизирующие компиляторы, то порядок полей структуры, зафиксированный в TypeCode, может не соответствовать реальному размещению данных и, следовательно, процедура интерпретации TypeCode не может быть использована.

Для реализации C++ ISP ORB время выполнения было наиболее критичным параметром, поэтому предпочтение было отдано скомпилированным процедурам маршалинга. Интерпретирующая процедура сохранилась для использования только в отдельных случаях. Но принимая во внимание перечисленные достоинства и недостатки подходов, более гибкой представляется комбинированная технология маршалинга с учетом частоты использования типов [Hosh96]. По принципу "локальности" рабочего множества, большую часть времени выполнения приложения по сети передаются данные некоторого ограниченного множества типов. Для этих типов порождаются типизированные процедуры маршалинга, более эффективные по времени выполнения, но существенно увеличивающие размер кода. В целях уменьшения размера кода для маршалинга данных остальных типов используется менее эффективная, но более компактная процедура интерпретации. Таким образом достигается определенный баланс между размером кода и скоростью

выполнения. Информация об использовании типов может быть получена путем профилирования программ или с помощью статического анализа IDL спецификаций (дополнительный проход по абстрактному синтаксическому дереву и вычисление атрибутов – частоты использования).

Другой способ оптимизации процедур маршалинга – объединение атомарных операций маршалинга для данных составных типов еще на этапе компиляции. Примитивы маршалинга включают вычисление границы выравнивания, вычисление размера памяти, необходимого для записи данных, выделение памяти и непосредственно запись самих данных. В общем случае последовательность таких действий выполняется для каждого примитивного типа, входящего в составной тип. Очевидно, что для данных имеющих, фиксированную длину, требуемый размер памяти может быть вычислен еще при построении типизированных процедур маршалига, на основе анализа абстрактного синтаксического дерева. А выделение памяти в буфере и на стороне сервера может быть осуществлено единой операцией. Если структура имеет определенную последовательность полей (например, поле типа short следует за полем типа long) можно исключить лишнюю проверку границы выравнивания. Компилятор протоколов, учитывающий перечисленные нюансы оптимизации, находится на стадии реализации.

4.4. Активные модели

Создание распределенных параллельных систем на сегодняшний день остается сложной задачей, хотя предложено много различных подходов и решений. Объектно-ориентированное программирование дало новый импульс в исследованиях, посвященных этой теме.

Понятие объекта и модель взаимодействия объектов, основанная на передаче сообщений (message passing), являются достаточно мощными и гибкими средствами для выражения различных видов вычислений. Многие объектные языки программирования и системы имеют параллельные и распределенные расширения, и почти каждое новое исследование в области распределенных систем основывается на объектном подходе. Как результат, сейчас создано и продолжает создаваться множество различных распределенных и параллельных объектных моделей. Например, языки программирования Ада-83 [ADA83] и Ада-95 [ADA95], среда функционирования и создания программ КЛАСТОС [BIK], акторные языки [AGH86], языки программирования C++// [CBR] и Java// [CVAY], модель постоянно хранимых потоков управления (Persistent threads) [MS94], система объектно-ориентированного моделирования приложений реального времени Real-time Object-Oriented Modeling (ROOM) [SGW94], модель взаимодействующих последовательных процессов, предложенная Ч. Хоаром, Communicating Sequential Processes (CSP) [CHH89].

В процессе исследовательской работы был рассмотрен ряд моделей, предложена их классификация и на ее основе предложена метамодель, позволяющая с одной стороны предоставить программисту уже освоенные средства, заимствованные из предшествующих моделей, а с другой стороны освободить его от решения типичных проблем и по возможности защитить от ошибок программирования.

Под активной моделью мы понимаем часть объектной модели, описывающую активности, соотношение их с традиционными объектами (в терминах конкретной объектной модели) и взаимодействие активностей между собой и, возможно, объектами. Понятие активности (процесс, задача, поток управления) – общая точка соприкосновения рассматриваемых моделей. Семантика активности различается от модели к модели. Также различаются и другие понятия активных моделей, каждое из которых можно рассматривать подробно и выделять на этом основании типы систем. Например, взаимодействие между объектами включает в себя соотношение между вызывающим и вызываемым объектом (один к одному, один к нескольким и так далее), способ их синхронизации (удаленный вызов процедуры, асинхронное взаимодействие и так далее). Эти и им подобные понятия положены в основу классификации.

Классификация моделей не является самоцелью и служит основой для создания метамодели.

Целью метамодели является описание некоторого класса моделей широко используемых систем (языков программирования, операционных систем, систем моделирования) и формирование базиса интероперабельности между приложениями, созданными в описанных моделях. Метамодель не ставит своей целью предоставить средства для описания моделей произвольных распределенных

параллельных систем. Исследуются в основном аспекты параллельного исполнения и взаимодействия объектов, а также связанные с ними вопросы синхронизации и конкурентного доступа. Рассматривается набор конкретных моделей (относительно которых можно заметить, что большинство из них применяются на практике) и иллюстрируется возможность выражения их понятий и примитивов через понятия и примитивы метамодели.

Модель предлагает общее понятие активного объекта.

Во время жизни объекта в распределенной параллельной системе можно выделить следующие важные для него этапы:

- создание объекта и создание соответствующей ему активности;
- уничтожение объекта и уничтожение соответствующей активности;
- взаимодействие объектов, которое в свою очередь состоит из более мелких шагов (получение объектной ссылки, формирование сообщения, его маршалинг и т.д.).

Передача сообщений в распределенной системе должна поддерживаться соответствующей инфраструктурой. Примером может служить технология CORBA и соответствующее программное обеспечение реализующее и поддерживающее ее.

Объект в метамодели состоит из следующих составных частей:

1. Генератор активностей – предназначен для создания активностей в разные моменты жизни объекта (является фабрикой объектов-активностей);
2. Объект-активность (в одном объекте их может быть один или несколько). Активности могут быть созданы статически или динамически;
3. Коммуникатор – обеспечивает взаимодействие объектов путем транспортировки сообщений от одного другому. Является посредником между взаимодействующими объектами. Семантика операций этого объекта влияет на способ общения между объектами. Может обеспечивать синхронные или асинхронные операции передачи сообщений;
4. Диспетчер – запуск активности на выполнение или активизация сообщений – принимаемых или посылаемых (в объекте может быть один или несколько диспетчеров). Удобно выделять два вида диспетчеров – для сообщений-запросов и сообщений-ответов. Диспетчеры могут реализовывать разные политики обслуживания соответствующих очередей. Политики (реализации разных схем) синхронизации могут быть оформлены в виде отдельных объектов;
5. Очереди запросов и ответов – поддерживают списки сообщений и служат для буферизации запросов и ответов;
6. Сообщения – динамически порождаемые объекты, инкапсулирующие запросы или ответы.

Предлагаемая схема предоставляет интерфейсы, обобщающие все рассмотренные модели. Она позволяет реализовывать указанные интерфейсы в соответствии с данной моделью. Предполагается библиотечная реализация системы на основе предлагаемой схемы, а также соответствующих средств поддержки.

4.5 Адаптация унаследованных систем в CORBA-среду

Технология построения распределенных систем подразумевает их принципиальную открытость. Принципы отделения интерфейсов от реализаций, спецификации интерфейсов с помощью средств, независимых от конкретных языков реализаций, предполагают потенциальную возможность расширения этих систем как с помощью новых технологий, так и за счет адаптации унаследованного кода (legacy system).

Прямой порядок разработки распределенной системы подразумевает следующую последовательность действий. Вначале интерфейс компонента специфицируется на независимом от реализаций языке определений. В архитектуре CORBA в качестве такого средства используется язык определения интерфейсов IDL. Далее по IDL-спецификации средствами стандартных трансляторов порождаются модули, отображающие типы и структуры данных, и реализующие необходимый библиотечный код (например, процедуры маршалинга). И уже после этого, с использованием полученного по IDL-спецификации кода, пишется реализация компоненты.

Для унаследованных систем реализация уже существует. Поэтому при модернизации унаследованной системы

самое важное – это суметь обеспечить взаимодействие ее объектной модели с моделью расширяющих объектов. В случае использования технологии CORBA модернизация системы включает следующие шаги. Во-первых, необходимо выделить ту часть системы, которую желательно сделать “открытой”. Во-вторых, тем или иным способом создать необходимые IDL-спецификации. И, наконец, в соответствии с прямым порядком разработки CORBA-приложений, реализовать полученные интерфейсы, опираясь на код существующей системы.

Первый этап является наиболее творческим и под силу только разработчику. Получение же IDL-спецификаций и кода посредников по имеющимся реализациям до определенной степени может быть автоматизировано.

При получении IDL-спецификации возникает необходимость в “обратном” отображении из языка реализации в IDL. Как правило, модели и уровни абстракции языков реализации и IDL сильно отличаются. И как следствие, “обратное” отображение становится неоднозначным, и требует дополнительные инструкции по уточнению. Обратим внимание на тот факт, что потом по IDL-спецификации будет получен код на языке реализации уже в соответствии с “прямым” отображением. И таким образом неоднозначности отображения необходимо уточнять дважды.

При разработке отображения из IDL в Protel-2 особое внимание было уделено исследованию проблем адаптации унаследованного кода в CORBA-среду. Было разработано “расширенное” отображение и реализован соответствующий компилятор, позволяющие по “расширенным” IDL-спецификациям генерировать код, учитывающий особенности существующей системы.

Язык IDL имеет встроенное средство для указания системно-зависимой информации компилятору – прагмы. В директивах pragma программист мог бы ссылаться на определения существующих в унаследованной системе типов, производить замену имен, использовать другие, уже скомпилированные IDL-спецификации. Прагмы имеют смысл только для того компилятора, который умеет их интерпретировать. Стандартный компилятор с IDL просто игнорирует неизвестные ему прагмы.

Для возможности использования существующего в унаследованной системе кода были введены следующие классы прагм.

- прагмы, предназначенные для включения текста на IDL или Protel-2 в IDL спецификацию или в порождаемые Protel-2 модули соответственно. Прагмы этого класса называются *includeProtel, includes, inline, useProtel, uses*;
- прагмы, позволяющие изменять имена, генерируемые IDL компилятором по умолчанию: *prefix, replaceid, filename*;
- прагмы, указывающие, что IDL спецификация для данного интерфейса будет использоваться только как клиентская или только как серверная: *noclient, noserver*;
- наконец, прагмы, дающие возможность подменить порождаемый IDL компилятором тип, класс или атрибут класса на существующие (например, в унаследованной системе) объявления: *maptype, mapinterface, mapattribute*.

Наиболее важным с точки зрения использования написанного кода является последний класс прагм. Вставленные в IDL-спецификацию, они заставляют IDL компилятор не генерировать тип, класс или атрибут по умолчанию, а использовать вместо него указанный Protel-2 тип, класс или атрибут соответственно.

Приведенные выше классы прагм предназначены для конкретного языка программирования – Protel-2, однако, аналогичным образом можно было бы расширить IDL компилятор с любого другого объектно-ориентированного языка.

Итак, хотя технология CORBA рассчитана на создание новых объектно-ориентированных приложений, она может быть успешно использована и для распределения существующих объектных систем.

5. Заключение

Возрастающая популярность распределенных систем подтверждает актуальность исследований в области распределенной обработки. Объектно-ориентированный подход, используемый как базовая модель во многих распределенных технологиях, остается одним их самых перспективных направлений в области разработки сложных распределенных приложений.

Исследования подтвердили необходимость создания единой среды разработки и функционирования для распределенных систем. Такая среда

должна обеспечивать возможность расширения функциональности приложений путем “прозрачного” подключения системных сервисов, а также возможность согласования распределенных контекстов, в которых существуют прикладные объекты. Используя наработки в области повышения надежности и эффективности распределенного программирования, такая унифицированная среда может существенно упростить работу по созданию больших распределенных программных комплексов. В процессе работы была предложена модель оболочки, обеспечивающая прозрачное подключение сервисов и согласование распределенных контекстов на основе эффективной реализации метаобъектных протоколов.

Часть проектов группы, реализованных в контексте исследований проблем распределенных вычислений, нашла конкретное практическое применение. В частности, брокер объектных заявок C++ ISP ORB свободно доступен в сети Internet, и судя по откликам, имеет определенную популярность. В рамках группы этот же брокер был использован для другого проекта, связанного с разработкой информационной системы для биологических банков данных, Biomax Retrieval System.

Литература

[CORBA97] Object Management Group. The Common Object Request Broker: Architecture and Specification, Revision 2.1, September 1997.

[CORBA98] Object Management Group. The Common Object Request Broker: Architecture and Specification, Revision 2.2, February 1998.

[COSS97] Object Management Group. CORBA Services: Common Object Services Specification, Revised Edition, July 1997.

[BZKSh95] Д.О. Брюхов, В.И. Задорожный, Л.А. Калиниченко, М.Ю.Курошев, С.С. Шумилов, Интероперабельные информационные системы: архитектуры и технологии, Системы управления базами данных, N 4, 1995.

[KK96] Л.А. Калиниченко, М.Р.Когаловский, Интероперабельность брокеров в стандарте CORBA 2.0, Системы управления базами данных, N 3,1996.

[FDM94] Ira R. Forman, Scott Danforth, Hari Madduri. Composition of Before/After Metaclasses in SOM. OOPSLA 94- 10/94 Portland, Oregon USA

[GC96] B.Gowing, V.Cahill. Metaobject protocols for C++: the Iguana approach. Proceedings of Reflection 96 Conference. 1996.

[Chi96] Shigeru Chiba. OpenC++ Programmer's Guide for Version 2. Technical Report SPL-96-024, Xerox PARC, 1996. (http://www-masuda.is.s.u-tokyo.ac.jp/openc++.html)

[IDZ97] В.П. Иванников, К.В. Дышлевой, В.И. Задорожный. Спецификация метанаращиваний для эффективного метаобъектного контроля. Программирование, N 4, 1997.

[IZKN96] Ivannikov V., Zadorozhny V., Kossmann R., Novikov B. Efficient Metaobject Control Using Mediators. 2nd Int.A.Ershov’s Conf., Novosibirsk, 1996

[KRB91] Kiczales G., des Rivieres J., Bobrow D. The Art of the Metaobject Protocol. MIT Press, 1991.

[MMAY95] Masuhara H., Matsuoka S., Asai K., Yonezava A. Compiling Away the Meta-Level in Object-Oriented Concurrent Reflective Languages Using Partial Evaluation. OOPSLA'95. 1995. p. 300-315.

[ZC95] Chris Zimmermann and Vinny Cahill. How to Structure Your Regional Meta: A New Approach to Organizing the Metalevel. In Proceedings of META '95, a workshop held at the European Conference of Object-Oriented Programming, 1995.

[ZC96] Chris Zimmermann and Vinny Cahill. It's Your Choice – On the Design and Implementation of a Flexible Metalevel Architecture. Proceedings of the International Conference on Configurable Distributed Systems, IEEE, Annapolis, Maryland, May, 1996. (ftp://ftp.dsg.cs.tcd.ie/pub/doc/dsg-100.ps.gz)

[IVDKM97] В.П. Иванников, А.Н. Винокуров, К.В. Дышлевой, В.Е. Каменский, А.В. Климов, С.Г. Манжелей, В.А. Омельченко, Л.Б. Соловская. Методология разработки распределенных приложений на основе модели оболочки. Вопросы кибернетики, N 3, 1997.

[KMS97] В.Е. Каменский, А.В. Климов, С.Г. Манжелей, Л.Б. Соловская. Применение стандарта CORBA для унаследованных систем. Вопросы кибернетики, N 3, 1997.

[DKS97] К.В. Дышлевой, В.Е. Каменский, Л.Б. Соловская. Маршалинг данных в распределенных системах: сравнение двух подходов. Вопросы кибернетики, N 3, 1997.

[Hosh96] Philipp Hoschka. Automating Performance Optimisation by Heuristic Analysis of A Formal Specification. IFIP TC 6/6.1 International Conference on Formal Description Techniques IX (Teory, application and tools), 1996.

[Dysh97] К.В. Дышлевой. IDL C++ меппинг. CORBA и возможная альтернатива. Вопросы кибернетики, N 3, 1997.

[XML99] Extensible Markup Language (XML) 1.0. W3C Recommendation, February1998. (http://www.w3.org/TR/REC-xml/.)

[AGH86] G. Agha, An Overview of Actor Languages, ACM SIGPLAN Notices, v. 21(10), October 1986.

[MS94] F. Matthes, and J.W. Schmidt, Persistent Threads, In Proceedings of the Twentieth International Conference on Very Large Data Bases, VLDB, Santiago, Chile, September 1994.

[BIK] Кузнецов С.Д., В.П.Иванников, И.Б.Бурдонов, А.С.Косачев, Г.В.Копытов. Принципы организации КЛОС - кластерной операционной системы. Программирование, N 6, 1990

[SGW94] B. Selic, G. Gullekson, P.T. Ward, Real-Time Object-Oriented Modeling, John Wiley and Sons, 1994 (ROOM).

[СНН89] Ч. Хоар. Взаимодействующие последовательные процессы. Москва, 1989 (CSP).
[CBR] D. Caromel, F. Belloncle, and Y. Roudier, The C++// Language.
[CVAY] D. Caromel, and J. Vayssiere, A Java Framework for Seamless Sequential, Multi-threaded, and Distributed Programming (Java//).
[ADA83] Н. Перминов. Ада-83, Москва, 1990.
[ADA95] Ada Reference Manual, Language and Standard Libraries, Version 6.0, ISO/IEC 8652:1995(E), December 1994.
[Jas99] Jason W. Pegg. An introduction to Active Server Pages. (http://www.auroradevelopment.com/OnlineTraining/Introduction.htm.)