

Технологии построения распределенных объектных систем

Сергей Семихатов

1. Введение

Рост популярности глобальной сети Internet и технологии World-Wide-Web в последнее время вызывает повышенный интерес к ним со стороны разработчиков информационных систем.

Изначально WWW создавался только как средство, предоставляющее графический интерфейс в Internet и упрощающее доступ к информации, распределенной по миллионам компьютеров по всему миру. При этом, основными компонентами являлись страницы, узлы, браузеры и сервера Web. Пользователям была предоставлена возможность навигации по Internet с использованием технологии гипертекста, поддерживаемой протоколом HTTP (Hypertext Transfer Protocol) и стандартом языка HTML (Hypertext Markup Language).

Появление CGI (Common Gateway Interface) решило проблему обмена информацией между сервером Web и такими программами как базы данных, которые не могут непосредственно обмениваться данными с браузерами Web. В результате появилась возможность реализации интерактивного взаимодействия конечного пользователя с программами стороны Web сервера, которые обрабатывали информацию, введенную пользователем в браузере, и в качестве результата возвращали сформированную HTML-страницу. Многие из существующих решений доступа к БД в среде Internet основаны на данном подходе.

Появление языка Java предоставило для разработчиков информационных систем абсолютно новые технологические решения построения приложений в среде Internet/Intranet (не стоит, однако, рассматривать Java только как часть технологии

WWW, поскольку она позволяет решать задачи гораздо более широкого класса, чем технология, базирующаяся на языке HTML, протоколе HTTP и CGI).

Но как быть в случае взаимодействующих информационных систем, основанных на технологии WWW? Как осуществить интеграцию этих систем друг с другом и с существующими крупными информационными системами?

Существуют различные решения этой проблемы, в основе которых лежит модель распределенных объектных технологий. Выбор такой модели во многом определяет характеристики строящейся информационной системы.

Известно, что распределенная информационная система состоит из совокупности взаимодействующих друг с другом программных компонент. Каждая из таких компонент представляет собой программный модуль, исполняемый в рамках отдельного процесса. Использование объектно-ориентированного подхода при создании крупных информационных систем позволяет рассматривать компоненты информационной системы на различных уровнях абстракции как объекты, каждый из которых обладал бы определенной линией поведения. Взаимодействие таких объектов, в большинстве случаев, осуществляется на базе некоторой среды взаимодействия, основной целью которой является реализация механизма обмена сообщениями в контексте гетерогенных распределенных сред, являющихся характерной чертой большинства крупных организаций.

Построение среды взаимодействия, есть один из труднейших этапов разработки информационной системы. Как показывает практика, создание разработчиками информационных систем собственной, уникальной среды взаимодействия объектов приводит с одной стороны к резкому увеличению затрат на реализацию проекта построения информационной системы, а с другой к неполноте (ущербности) полученного решения. Исследования проектов создания информационных систем позволяют сделать вывод, что для того чтобы избежать неоправданных затрат на разработку собственной, уникальной информационной среды, необходимо использовать уже существующие программные продукты, которые относятся к уровню промежуточного программного обеспечения (middleware) и реализуют так необходимые среды взаимодействия. Однако не все продукты уровня middleware могут использоваться в качестве среды взаимодействия объектов крупной информационной системы. Это обусловлено тем, что одним из основных требований к крупной

информационной системе является использование программных продуктов и технологий, удовлетворяющих международным и промышленным стандартам в области открытых информационных систем. В связи с этим, на сегодняшний день, в качестве кандидатов, реализующих высокоуровневую среду взаимодействия, все чаще рассматриваются продукты, поддерживающие стандарт CORBA.

Целью данной работы является теоретическое обоснование того, что на сегодняшний день технологии Java и CORBA лучше всего подходят для интегрирования систем, основанных на технологии WWW, с крупными информационными системами. То есть использование этих технологий уже сейчас позволяет поднять технологию WWW на новый уровень — уровень распределенных систем.

2. Распределенные объектные технологии

На сегодняшний день многие компании и производители поглощены прорывом в области Интернет-ориентированных распределенных клиент/сервер систем, основанных на модели распределенных объектов.

Система, построенная по технологии распределенных объектов, состоит из набора компонент (объектов), взаимодействующих друг с другом. При этом объекты, как правило, разбросаны по сети и выполняются отдельно друг от друга.

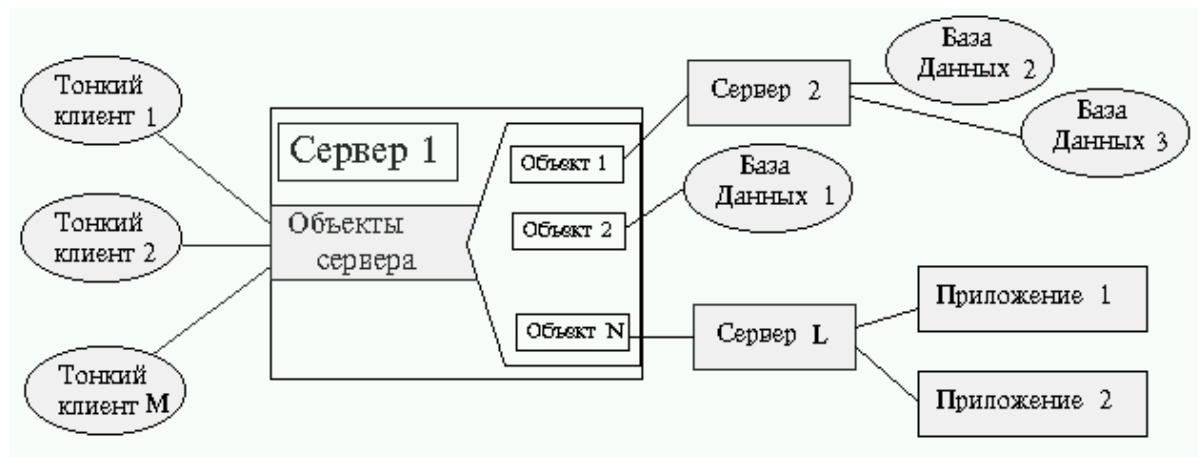


Рисунок 1. Модель распределенных объектов.

2.1. Преимущества использования

Использование технологии распределенных объектов позволяет пользоваться всеми преимуществами объектно-ориентированного подхода [2]:

- сокращение времени разработки (изолированная разработка)
- сокращение количества ошибок
- повторное использование программных компонент
- легче становится будущее изменение системы.

Еще одним важным достоинством таких систем является возможность построения так называемых легких клиентов.

2.2. Повторное использование кода

У программистов появляется возможность быстро и эффективно создавать многофункциональные приложения, используя уже существующие plug-and-play компоненты, что заметно снижает стоимость построения новой системы.

2.3. Изолированная разработка

Из-за своей модульной основы распределенные приложения позволяют осуществить изолированные друг от друга создание и изолированное изменение модулей (компонент). Вся система разбивается на различные замкнутые, автономные модули, работа над которыми может идти отдельно от других, но которые, в то же время, могут взаимодействовать с другими модулями системы. Для этого модули должны поддерживать протоколы и интерфейсы, определяющие принципы их взаимодействия. Но так как методы, существующие в модулях, изолированы от методов других модулей, то они могут разрабатываться независимо. Таким образом, степень реализации компонент не зависит от состояния кода в других частях системы. Становится возможной параллельная работа нескольких команд над различными частями приложения или системы. Взаимодействие же между разными модулями будет происходить путем установленных протоколов и интерфейсов.

2.4. Сопровождение приложений

В силу модульного подхода в системе, замена некой функциональной части приложения для решения возникающих проблем не требует глобальной перестройки

всей системы. Наоборот, замена кода происходит только в модулях, которые на самом деле этого требуют. Во-первых, это проще, а во-вторых, значительно быстрее. Снижается также и риск возникновения ошибок при самой замене кода, ведь изменения в большей степени затрагивают внутреннюю часть объектов.

2.5. Тонкие клиенты

В распределенной системе имеется возможность перенести всю функциональную логику информационной системы на ее серверную часть. В этом случае приложения-клиенты, с которыми общается пользователь, могут быть сделаны небольшими и легковесными. Системные ресурсы пользователя оказываются более свободными, а вся тяжесть функциональной логики реализуется высокомоощным сервером (или сетью из серверов). При этом клиент имеет доступ к практически неограниченному числу хранилищ информации и других объектов. Появляется возможность создания легковесных компонент, пригодных для быстрой загрузки через сеть (Internet) и запуска на компьютере клиента (к такого рода приложениям можно отнести апплеты или Active-X компоненты).

3. Технологии RMI, CORBA и DCOM

На сегодняшний день выделяются три различные технологии, поддерживающие концепцию распределенных объектных систем. Это технологии RMI, CORBA и DCOM.

3.1. RMI

Архитектура RMI (Remote Method Invocation, т.е. вызов удаленного метода), которая интегрирована с JDK1.1, является продуктом компании JavaSoft и реализует распределенную модель вычислений. RMI позволяет клиентским и серверным приложениям через сеть вызывать методы клиентов/серверов, выполняющихся в Java Virtual Machine. Хотя RMI считается легковесной и менее мощной, чем CORBA и DCOM тем не менее, она обладает рядом уникальных свойств, таких как распределенное, автоматическое управление объектами и возможность пересылать сами объекты от машине к машине.

На рисунке 2 показаны основные компоненты архитектуры RMI.

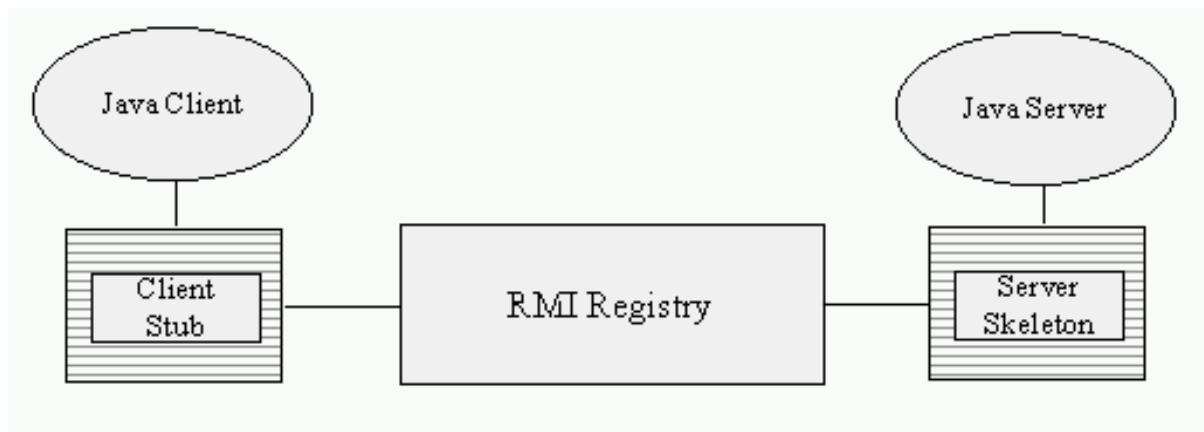


Рисунок 2. Модель RMI.

Client Stub (переходник для клиента) и *Server Stub* (переходник для сервера) порождены от общего интерфейса, но различие между ними в том, что *client stub* служит просто для подсоединения к RMI Registry, а *server stub* используется для связи непосредственно с функциями сервера.

3.2. CORBA

Технология CORBA (Common Object Request Broker Architecture), разрабатываемая OMG (Object Management Group) с 1990-го года, позволяет вызывать методы у объектов, находящихся в сети где угодно, так, как если бы все они были локальными объектами.

На рисунке 3 показана основная структура CORBA 2.0 ORB.

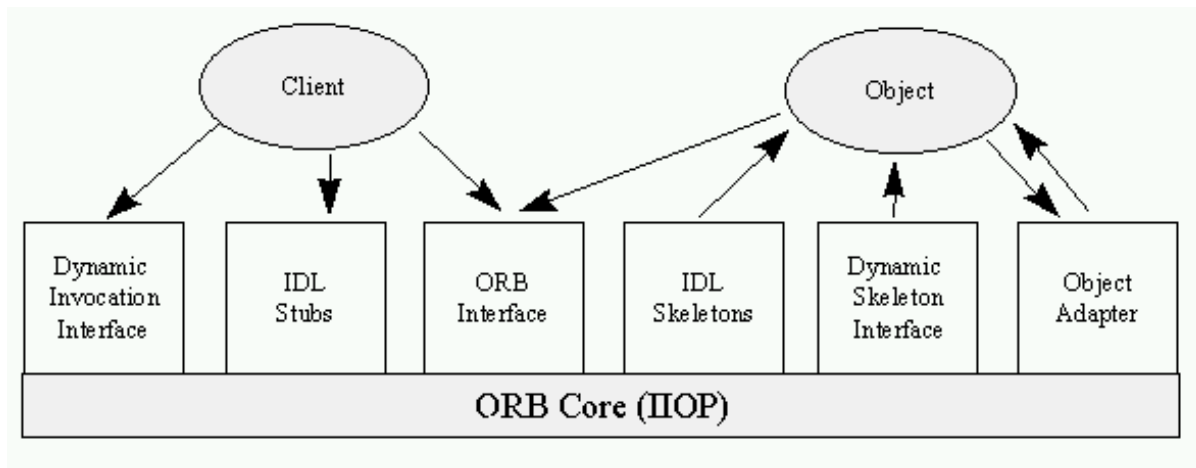


Рисунок 3. ORB (CORBA 2.0).

Dynamic Invocation Interface (DII): позволяет клиенту находить сервера и вызывать их методы во время работы системы. *IDL Stubs*: определяет, каким образом клиент производит вызов сервера. *ORB Interface*: общие как для клиента, так и для сервера сервисы. *IDL Skeleton*: обеспечивает статические интерфейсы для объектов определенного типа. *Dynamic Skeleton Interface*: общие интерфейсы для объектов, независимо от их типа, которые не были определены в IDL Skeleton. *Object Adapter*: осуществляет коммуникационное взаимодействие между объектом и ORB.

3.3. DCOM

Технология DCOM (Distributed Component Object Model) была разработана компанией Microsoft в качестве решения для распределенных систем в 1996-м году. Сейчас DCOM является главным конкурентом CORBA, хотя контролируется он теперь уже не Microsoft, а группой TOG (The Open Group), аналогичной OMG. Вкратце, DCOM представляет собой расширение архитектуры COM до уровня сетевых приложений.

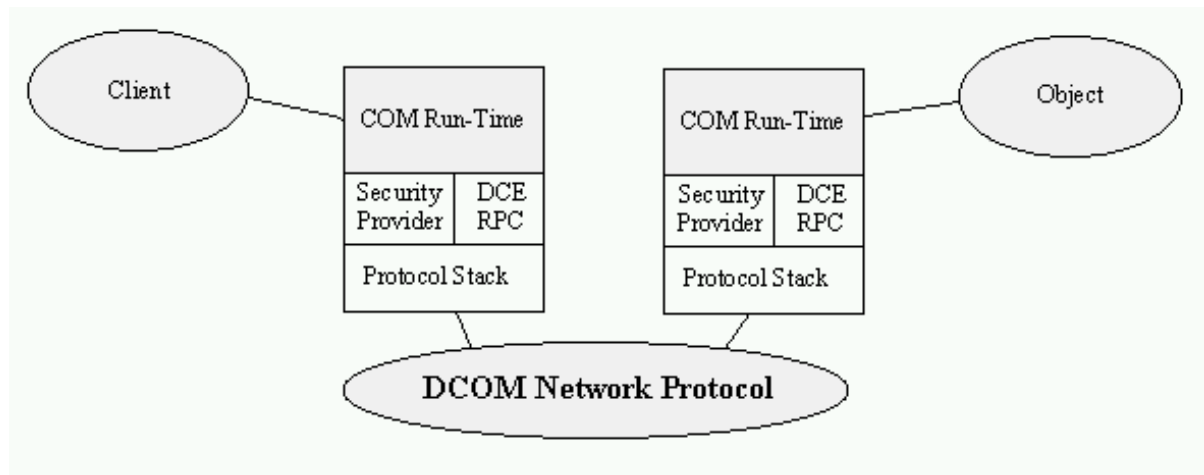


Рисунок 4. Архитектура DCOM.

4. Преимущества технологии CORBA над технологиями RMI и DCOM

У каждой из трех рассматриваемых технологий есть свои уникальные особенности, которые во многом характеризуют возможность или невозможность ее применения для решения поставленной задачи.

4.1. DCOM: за и против

Следуя [1] можно перечислить следующие достоинства и недостатки DCOM:

Достоинства	Недостатки
Языково-независимость	Сложность реализации
Динамический/статический вызов	Зависимость от платформы
Динамическое нахождение объектов	Нет именованя через URL
Масштабируемость	Нет проверки безопасности на уровне выполнения ActiveX компонент
Открытый стандарт (контроль со стороны TOG)	

DCOM является лишь частным решением проблемы распределенных объектных

систем. Он хорошо подходит для Microsoft-ориентированных сред. Как только в системе возникает необходимость работать с архитектурой, отличной от WindowsNT и Windows95, DCOM перестает быть оптимальным решением проблемы. Конечно, вскоре это положение может измениться, так как Microsoft стремится перенести DCOM и на другие платформы. Например, фирмой Software AG уже выпущена версия DCOM для Solaris UNIX и планируется выпуск версий и для других версий UNIX. Но все-таки, на сегодняшний день, DCOM хорош лишь в качестве решения для систем, ориентированных исключительно на продукты Microsoft. Большие нарекания вызывает также отсутствие безопасности при исполнении ActiveX компонент, что может привести к неприятным последствиям.

4.2. RMI: за и против

Как отмечалось в [1], RMI имеет следующие положительные и отрицательные стороны:

Достоинства	Недостатки
Быстрое и простое создание	Поддержка только одного языка — Java
Java-оптимизация	Свой собственный, не ПОР-совместимый протокол взаимодействия
Динамическая загрузка компонент-переходников	Трудность интегрирования с существующими приложениями
Возможность передачи объектов по значению	Плохая масштабируемость
Встроенная безопасность	

Благодаря своей легкоиспользуемой Java-модели, RMI является самым простым и самым быстрым способом создания распределенных систем. RMI – хороший выбор для создания RAD-компонент и небольших приложений на языке Java. Конечно, RMI не такая мощная технология, как DCOM или CORBA. В частности, RMI использует свой родной, не CORBA/ПОР-совместимый протокол передачи JRMP и может взаимодействовать лишь с другими Java объектами. Поддержка только одного языка делает невозможным взаимодействие с объектами, написанными не на языке Java. Тем самым, роль RMI в создании больших, масштабируемых промышленных систем,

снижается.

4.3. CORBA: за и против

Вот небольшой список достоинств и недостатков использования технологии CORBA.

Достоинства	Недостатки
Платформенная независимость	Нет передачи параметров `по значению'
Языковая независимость	Отсутствует динамическая загрузка компонент-переходников
Динамические вызовы	Нет именованя через URL
Динамическое обнаружение объектов	
Масштабируемость	
CORBA-сервисы	
Широкая индустриальная поддержка	

К основным достоинствам CORBA можно отнести межязыковую и межплатформенную поддержку. Хотя CORBA-сервисы и отнесены к достоинствам технологии CORBA, их в равной степени можно одновременно отнести и к недостаткам CORBA, ввиду практически полного отсутствия их реализации. Более подробное описание этих свойств можно найти в [1,3,5].

4.4. Почему CORBA?

Почему CORBA является наиболее эффективной, современной, пригодной для крупных проектов технологией распределенных объектов? Потому что, хотя обе технологии — и CORBA, и DCOM чрезвычайно схожи по своей функциональности и своим возможностям (многоязыковая поддержка, динамический вызов, масштабируемость и др.), у DCOM отсутствует важный критический элемент — мультиплатформенная поддержка. Одного факта, что в настоящий момент DCOM не поддерживает целиком межплатформенную переносимость, вполне достаточно, чтобы не рассматривать его как полноценное, законченное решение. Кроме того, в то время как в состав OMG уже сейчас входят более 700 членов (компаний-производителей

программных продуктов, компьютеров, телекоммуникационных систем, разработчиков прикладных систем и конечных пользователей), и практически любая спецификация, разработанная этим консорциумом, фактически становится стандартом, DCOM лишь недавно стал переходить из рук Microsoft в руки аналогичной OMG организации — группе TOG (The Open Group).

И еще один плюс технологии CORBA: круг производителей продуктов, поддерживающих данную технологию, значительно шире, чем аналогичный круг для DCOM. Таким образом оказывается, что именно CORBA — технология, полностью предназначенная для промышленных, открытых, распределенных объектных систем.

5. Технология CORBA

5.1. Введение в CORBA

В конце 1980-х и начале 1990-х годов многие ведущие фирмы-разработчики были заняты поиском технологий, которые принесли бы ощутимую пользу на все более изменчивом рынке компьютерных разработок. В качестве такой технологии была определена область распределенных компьютерных систем. Необходимо было разработать единообразную архитектуру, которая позволяла бы осуществлять повторное использование и интеграцию кода, что было особенно важно для разработчиков. Цена за повторное использование кода и интеграцию кода была высока, но ни кто из разработчиков в одиночку не мог воплотить в реальность мечту о широко используемом, языково-независимом стандарте, включающем в себя поддержку сложных многосвязных приложений. Поэтому в мае 1989 была сформирована OMG (Object Management Group). Как уже отмечалось, сегодня OMG насчитывает более 700 членов (в OMG входят практически все крупнейшие производители ПО, за исключением Microsoft).

Задачей консорциума OMG является определение набора спецификаций, позволяющих строить интероперабельные информационные системы. Спецификация OMG — The Common Object Request Broker Architecture (CORBA) является индустриальным стандартом, описывающим высокоуровневые средства поддержания взаимодействия объектов в распределенных гетерогенных средах.

CORBA специфицирует инфраструктуру взаимодействия компонент (объектов) на представительском уровне и уровне приложений модели OSI. Она позволяет рассматривать все приложения в распределенной системе как объекты. Причем объекты могут одновременно играть роль и клиента, и сервера: роль клиента, если объект является инициатором вызова метода у другого объекта; роль сервера, если другой объект вызывает на нем какой-нибудь метод. Объекты-серверы обычно называют ``реализацией объектов''. Практика показывает, что большинство объектов одновременно исполняют роль и клиентов, и серверов, попеременно вызывая методы на других объектах и отвечая на вызове извне. Используя CORBA, тем самым, имеется возможность строить гораздо более гибкие системы, чем системы клиент-сервер, основанные на двухуровневой и трехуровневой архитектуре [6].

5.2. IDL

Язык OMG IDL (Interface Definition Language — Язык Описания Интерфейсов) представляет собой технологически независимый синтаксис для описания интерфейсов объектов. При описании программных архитектур, OMG IDL прекрасно используется в качестве универсальной нотации для очерчивания границ объекта, определяющих его поведение по отношению к другим компонентам информационной системы. OMG IDL позволяет описывать интерфейсы, имеющие различные методы и атрибуты. Язык также поддерживает наследование интерфейсов, что необходимо для повторного использования объектов с возможностью их расширения или конкретизации.

IDL является чисто декларативным языком, то есть он не содержит никакой реализации. IDL-спецификации могут быть откомпилированы (отображены) в заголовочные файлы и специальные прототипы серверов, которые могут использоваться непосредственно программистом. То есть IDL-определенные методы могут быть написаны, а затем выполнены, на любом языке, для которого существует отображение из IDL. К таким языкам относятся C, C++, SmallTalk, Java и Ada.

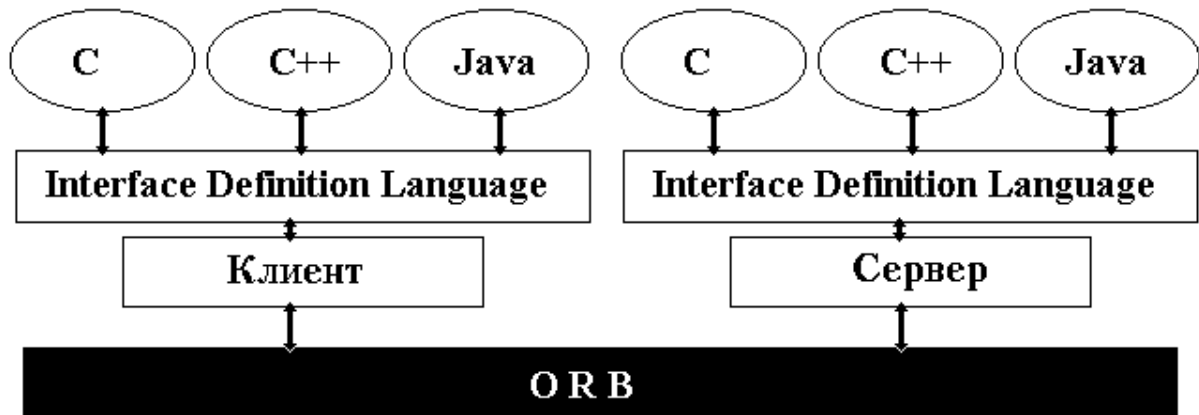


Рисунок 5. CORBA IDL отображения в модели Клиент/Сервер.

С помощью IDL можно описать и атрибуты компоненты, и родительские классы которые, она наследует, и вызываемые исключения, и, наконец, методы, определяющие интерфейс, причем с описанием входных и выходных параметров.

Структура CORBA IDL файла выглядит следующим образом:

```
module <identifier> {
    <type declarations>;
    <constant declarations>;
    <exception declarations>;

    interface <identifier> [:<inheritance>] {

        <type declarations>;

        <constant declarations>;

        <attribute declarations>;

        <exception declarations>;
    }
}
```

```
[<op_type>]<identifier>(<parameters>)  
  
[raises exception] [context]  
  
.  
  
.  
  
[<op_type>]<identifier>(<parameters>)  
  
[raises exception] [context]  
  
.  
  
.  
}  
  
interface <identifier> [:<inheritance>]  
  
.  
  
.  
}
```

Репозиторий Интерфейсов (Interface Repository), содержащий определения интерфейсов на IDL, позволяет видеть интерфейсы доступных серверов в сети и программировать их использование в программах-клиентах.

5.3. Object Management Architecture

Осенью 1990 года OMG впервые опубликовала документ *Object Management Architecture Guide (OMA Guide)*. Он был подкорректирован в сентябре 1992. Детали Common Facilities (Общие средства) были добавлены в январе 1995. Следующий рисунок показывает четыре основных элемента этой архитектуры:

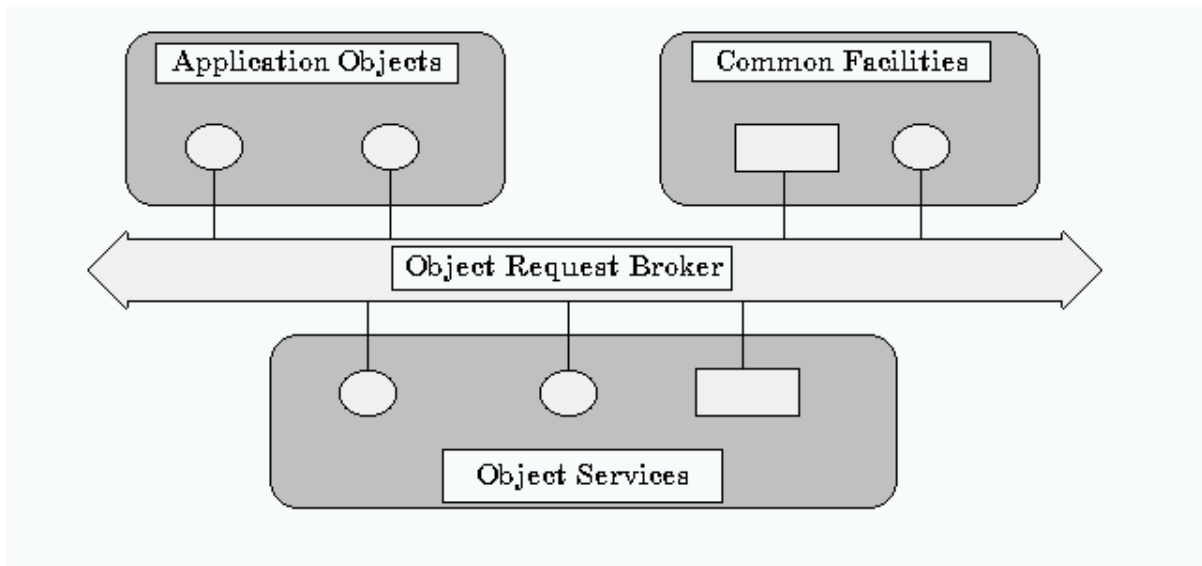


Рисунок 6. OMG's Object Management Architecture

1. *Object Request Broker* определяет объектную шину CORBA.
2. *Common Object Services* представляют собой коллекцию служб, снабженных объектными интерфейсами и обеспечивающих поддержку базовых функций объектов [7].
3. *Common Facilities* образуют набор классов и объектов, поддерживающих полезные во многих прикладных системах функции. Прикладные объекты представляют прикладные системы конечных пользователей и обеспечивают функции, уникальные для данной прикладной системы [7].
4. *Application Objects* — это прикладные бизнес-объекты и приложения, которые являются основными потребителями всей CORBA инфраструктуры.

5.3.1. Object Request Broker

ORB (Object Request Broker, то есть брокер объектных запросов) — это объектная шина. Она позволяет объектам напрямую производить и отвечать на запросы других объектов, расположенных как локально (на одном компьютере, но в разных

процессах), так и удаленно. Клиента не интересуют коммуникационные и другие механизмы, с помощью которых происходит взаимодействие между объектами, вызов и хранение серверных компонент. CORBA-спецификации затрагивают лишь IDL, отображение в другие языки, APIs для взаимодействия с ORB и сервисы, предоставляемые ORB.

CORBA ORB предоставляет широкий набор распределенных middleware услуг. Шина ORB позволяет объектам находить друг друга прямо в процессе работы и вызывать друг у друга различные службы. Она является гораздо более тонкой системой, чем другие клиент/сервер middleware-системы, такие как RPC (Remote Procedure Calls) или MOM (Message-Oriented Middleware).

5.3.2. От RPC к ORB

Чем механизм вызовов CORBA отличается от механизма RPC? Да, эти механизмы похожи, но тем не менее между ними есть серьезные различия [5]. С помощью RPC можно вызвать определенную функцию. А с помощью ORB можно вызвать метод у определенного объекта. Разные объекты классов могут по-разному отвечать на вызов одного и того же метода. Так как каждый объект управляет своей собственной (в добавок личной) информацией, то метод будет вызван на сугубо конкретных данных.

В случае RPC, будет выполнен лишь какой-то конкретный кусок кода сервера, который и взаимодействует с данными сервера. Все функции с одинаковыми именами будут выполнены абсолютно одинаково. В RPC отсутствует конкретизация вызовов, в том смысле, в каком это происходит в ORB. В ORB все вызовы функций происходят к конкретным объектам, тем самым, результаты этих функций могут быть совершенно различны. Вызовы функций обрабатываются в специфичной для каждого отдельного объекта форме.

5.3.3. Достоинства ORB

В теории, CORBA представляется как лучшая клиент/сервер middleware-система, но на практике она хороша лишь настолько, насколько хороши продукты, ее реализующие. К основным коммерческим ORB относятся: Orbix от фирмы IONA Technologies; DSOM от IBM; ObjectBroker от Digital; JOE от Sun; Visibroker от Visigenic и Netscape; ORB+ от HP.

Небольшой список тех выгод, которыми обладает каждая CORBA ORB [5]:

- **Статические и динамические вызовы методов.** CORBA ORB предоставляет возможность либо статически определить вызовы методов прямо во время компиляции, либо находить их динамически, но уже во время работы программы.
- **Отображение в языки высокого уровня.** CORBA ORB позволяет вызывать методы у серверных компонент используя любой из некоторого списка языков высокого уровня — C, C++, SmallTalk, Java и Ada. Совершенно неважно, на каком языке написаны объекты. CORBA отделяет интерфейсы от реализации и предоставляет языково-независимые типы данных, что позволяет осуществлять вызов методов, минуя границы какого-то конкретного языка программирования и конкретной операционной системы.
- **Само-описывающаяся система.** С помощью своих метаданных, CORBA позволяет описывать интерфейс любого сервера, известного системе. Каждая CORBA ORB должна поддерживать Репозиторий Интерфейсов, который хранит необходимую информацию, описывающую синтаксис интерфейсов, поддерживаемых серверами. В своей работе клиенты используют эти метаданные для осуществления вызовов к серверам.
- **Прозрачность.** ORB может выполняться как сам по себе (например на портативном компьютере), так и в окружении целого мира других ORB, с которыми она взаимодействует путем CORBA 2.0 ИОП (Internet Inter-ORB Protocol) протокола. ORB может осуществлять меж-объектное взаимодействие и для одного процесса, и для нескольких процессов, выполняющихся на одной машине, и для процессов, чье выполнение происходит в сети, под разными операционными системами. Реализация этих взаимодействий, однако, нисколько не затрагивает сами объекты. В общих чертах, при использовании технологии CORBA, разработчик не должен беспокоиться ни о таких вещах как расположение серверов, запуск (активирование) объектов, выравнивание размера переменных в зависимости от платформы и операционной системы, ни и о том, как осуществляется передача сообщений. Решение всех этих задач берет на себя продукт, поддерживающий стандарт CORBA.
- **Встроенная безопасность.** Все свои запросы ORB дополняет некоторой контекстной информацией которая обеспечивает сохранность данных.
- **Полиморфизм при вызове методов.** Как уже говорилось, ORB не просто вызывает удаленный метод — ORB вызывает метод на удаленном объекте. То есть

выполнение одних и тех же функций на разных объектах будет приводить к различным действиям, в зависимости от типа объекта.

5.3.4. Object Services

CORBA *Object Services* представляет собой набор сервисов системного уровня, представимых в виде компонент с некоторыми определенными IDL-интерфейсами. Эти компоненты, в некотором смысле, дополняют функциональность ORB. Их можно использовать для создания, именованного компонент и многого другого. На сегодняшний день OMG определил четырнадцать стандартных сервисов.

К сожалению, практически все коммерческие ORB не поддерживают ни один из сервисов, и лишь немногие (*Visibroker*) — один, два.

5.4. Common Facilities

Common Facilities (общие средства) заполняют пространство между ORB и объектными службами с одной стороны, и прикладными службами, с другой. Таким образом, ORB обеспечивает базовую инфраструктуру, *Object Services* — фундаментальные объектные интерфейсы, а задача *Common Facilities* — поддержка интерфейсов сервисов высокого уровня, которые, впрочем, могут включать специализацию *Object Services*. Таким образом, операции, представляемые *Common Facilities*, предназначены, в частности, для использования *Object Services* и прикладными объектами. Реализуется это посредством наследования стандартных Интерфейсов [7].

Общие средства делятся на *горизонтальные* и *вертикальные*. К горизонтальным сервисам относятся такие общие сервисы, как, например, управление информацией, задачами, всей системой, то есть средства, не зависящие от конкретных прикладных систем. К вертикальным же относятся сервисы, специфичные для какой-либо деятельности — например, медицина, финансы.

5.5. Application Objects

Объекты, если они участвуют в обмене с ORB, должны определяться с помощью IDL. Обычно приложения состоят из нескольких взаимодействующих бизнес-объектов. И,

как правило, приложения-объекты строятся поверх предоставляемых ORB, Common Facilities и Object Facilities сервисов. Суть для заказчиков (или системных интеграторов) заключается в том, чтобы собрать разные бизнес-объекты в одну систему, при том, вне зависимости от производителя.

6. CORBA и WWW

Ответ на поставленный во Введении вопрос — как объединить информационные системы, основанные на технологии WWW, с другими (в том числе и распределенными) информационными системами? — заключается в следующем: необходимо связать технологию распределенных объектов (то есть технологию CORBA) с технологией WWW. Целью настоящей работы является более детальное рассмотрение связки CORBA и WWW. О внедрении же CORBA в информационные системы выходит за рамки настоящей работы. Более подробную информацию об этой проблеме можно найти, например, в [3,5].

Было выделено два различных решения поставленной задачи. Первое решение строится на применении технологии CGI, а второе — на применении технологии Java. В рамках проделанной работы было исследовано практическое применение этих технологий, с использованием продуктов Orbix и OrbixWeb от IONA Technologies.

6.1. CORBA-CGI

В чистом виде, технология CGI состоит в следующем: для формирования HTML-страницы Web-сервер запускает некий CGI-скрипт. При этом CGI-скрипт может реализовывать довольно сложную функциональную логику и обращаться, например, к базе данных. CGI-скрипт представляет собой отдельное исполняемое приложение. Язык, на котором написан скрипт, не играет большой роли.

Решение CORBA-CGI основывается на том, что исполняемый CGI-скрипт является одновременно и одной из компонент распределенной системы. Главное отличие от технологии CGI заключается в том, что CGI-скрипт является не просто исполняемым модулем, а он является одновременно и CORBA-клиентом. В некотором смысле, скрипт служит точкой входа и выхода в распределенной системе, внутри которой могут протекать различные процессы.

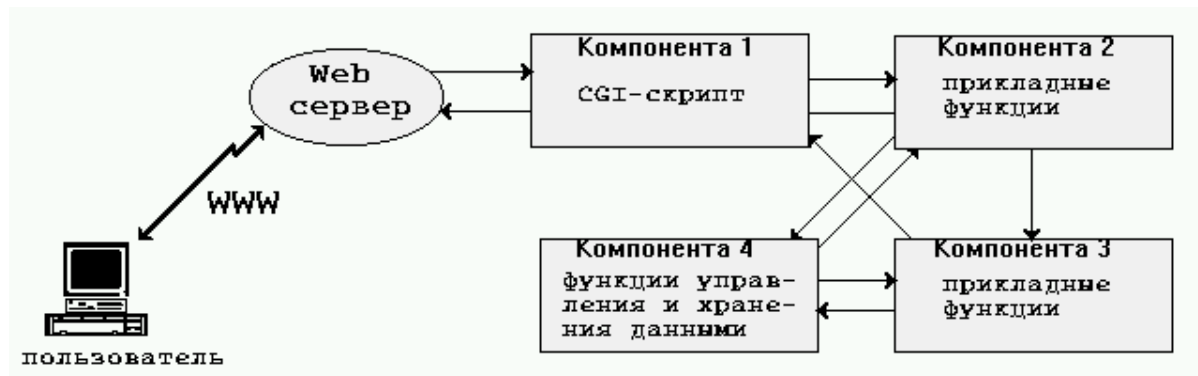


Рисунок 7. CGI и CORBA.

К преимуществам этой технологии можно отнести практически все преимущества использования CORBA. Кроме того, пользователь работает с привычными для него HTML-страницами, что особенно важно при работе с объемной текстовой информацией.

Теперь о недостатках этой технологии. Практика показывает, что во-первых — это невысока эффективность системы. Каждый раз при перезагрузке страницы необходимо заново выполнять CGI-скрипт, заново устанавливать соединения с другими компонентами системы. Это не эффективно, что особенно сильно проявляется, когда система одновременно используется несколькими пользователями. Во-вторых, не жертвуя эффективностью, невозможно своевременно уведомлять пользователя об изменениях, произошедших в просматриваемой им информации. Они будут видны лишь при перезагрузке страницы. То есть пользователь участвует в системе исключительно в роли клиента.

Частным решением проблемы эффективности выполнения CGI-запросов (особенно в многопользовательской системе), может быть расширение функциональности используемого Web-сервера, путем добавления в него соответствующих функций. В этой области была проделана работа по встраиванию в Web-сервер Apache (платформы SunOS, WindowsNT) поддержки обращения к Orbix ORB.

Сложности также возникают при создании сложных, ветвистых пользовательских интерфейсов. Дело в том, что системе, при выполнении запросов помимо веденной в окне браузера информации, необходима дополнительная, скрытая от пользователя и характеризующая текущее состояние системы информация. Вся интерфейсная часть

системы привязана к окну браузера, а выводимая на экран информация создается динамически в зависимости от параметров запроса и внутреннего состояния информационной системы на некоторый момент времени. Вследствие этого, если пользователь производит неконтролируемую навигацию вперед-назад по просматриваемым страницам, повышается риск десинхронизации просматриваемой пользователем и хранимой на сервере информации, что недопустимо. Более того, система должна следить и за тем, чтобы вольные переходы от страницы к странице имели бы логический смысл, что тоже немаловажно. Тем самым, в случаях сложных пользовательских интерфейсов необходимо наличие жесткого контроля за текущим состоянием информационной системы.

6.2. Java-CORBA

Второе решение проблемы связывания технологий CORBA и WWW — язык Java. Дело в том, что OMG стандартизировала отображение из IDL в Java. Имеются программные продукты, реализующие связь CORBA и Java — например, OrbixWeb, Visibroker. Java-технология основана на том, что при обнаружении тега `<APPLET>`, браузер через сеть загружает к себе необходимые для этого апплета Java-классы и запускает его. При этом на машине пользователя запускается Java Virtual Machine (JVM), внутри которой и выполняется загруженный апплет.

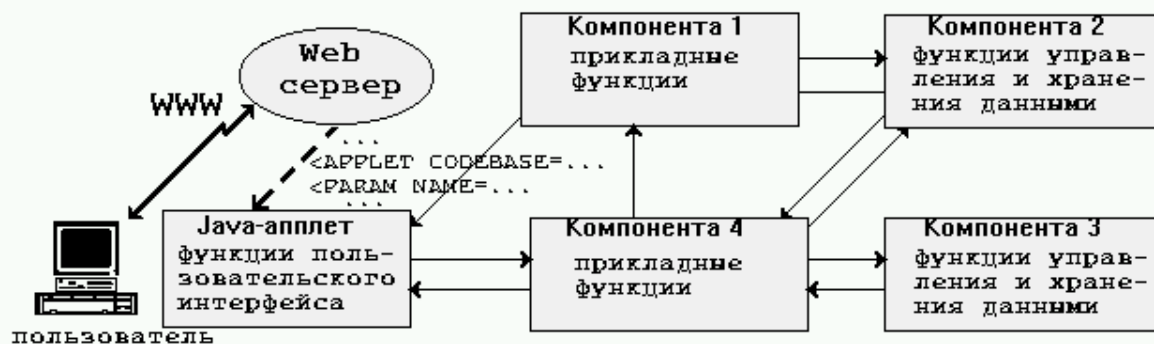


Рисунок 8. Java и CORBA.

Java-апплет, являющийся CORBA-клиентом, устанавливает все необходимые соединения с другими (серверными) приложениями системы и именно через него к пользователю идет вся информация. Апплет играет роль пользовательского интерфейса для данной распределенной системы. Количество выполняемых апплетов

ничем не ограничено — вопрос лишь в достаточных вычислительных ресурсах системы.

Коротко о достоинствах и недостатках технологии Java-CORBA:

Достоинства	Недостатки
Java — платформенно-независимый язык, все апплеты будут выполняться одинаково на любой системе	для эффективного выполнения Java-приложений, система пользователя должна обладать достаточно мощными вычислительными ресурсами, что особенно важно для сложных (насыщенных) пользовательских интерфейсов
имеется возможность создания сложных пользовательских интерфейсов	Не все браузеры поддерживают Java, или ее последние версии
апплеты могут выполнять роль и клиента, и сервера	
все преимущества как технологии CORBA, так и технологии Java	
не возникает проблем при одновременной работе нескольких пользователей. Нет необходимости каждый раз заново загружать апплет, как это происходит в случае с CGI.	

Такие встроенные в Java средства как многопоточность, позволяют легко реализовывать и синхронное, и асинхронное взаимодействие апплетов с другими приложениями. У технологии Java-CORBA практически нет слабых мест. Единственная проблема, которая может возникнуть — необходимость наличия мощных вычислительных ресурсов на стороне пользователя. Это, конечно, серьезный недостаток, но он, скорее, является недостатком языка Java.

6.2.1. Java — путь к синхронизации информации

Как уже отмечалось, при использовании технологии Java-CORBA, Java-апплеты могут играть роль и клиентов, и серверов. Это позволяет создание ``живых" страниц, то есть страниц, информация на которых меняется практически постоянно. Например, если апплет представляет собой отображение состояния некоторого устройства, то при

переходе устройства из одного состояния в другое, информация в апплете практически мгновенно изменится соответствующим образом. Причем это может быть информация любого рода — как графическая, так и текстовая. Возникает вопрос: а каким способом происходит обмен информацией между апплетами и остальной частью системы?

В CORBA существует два различных механизма передачи сообщений — механизм Push и механизм Pull [4].

6.2.2. PUSH и PULL

Механизм PULL представляет собой следующее: когда клиент готов обрабатывать сообщения, он опрашивает сервер на наличие у того новых сообщений. Если таковых не имеется, клиент через некоторый промежуток времени повторяет операцию. При этом, в зависимости от контекста решаемой задачи и пропускных способностей сетевых каналов, тип взаимодействия клиента и сервера может быть как асинхронным, так и синхронным.

Механизм PUSH, в некотором смысле, противоположен механизму PULL. В этом случае сервер сообщений сам, по мере поступления новых сообщений, будет информировать об этом клиентов. То есть клиенты сами являются серверами, а сервер сообщений лишь вызывает у них соответствующие методы, «вталкивая» им сообщение. Как и в модели PULL, взаимодействие клиента и сервера сообщений может быть и асинхронным, и синхронным.

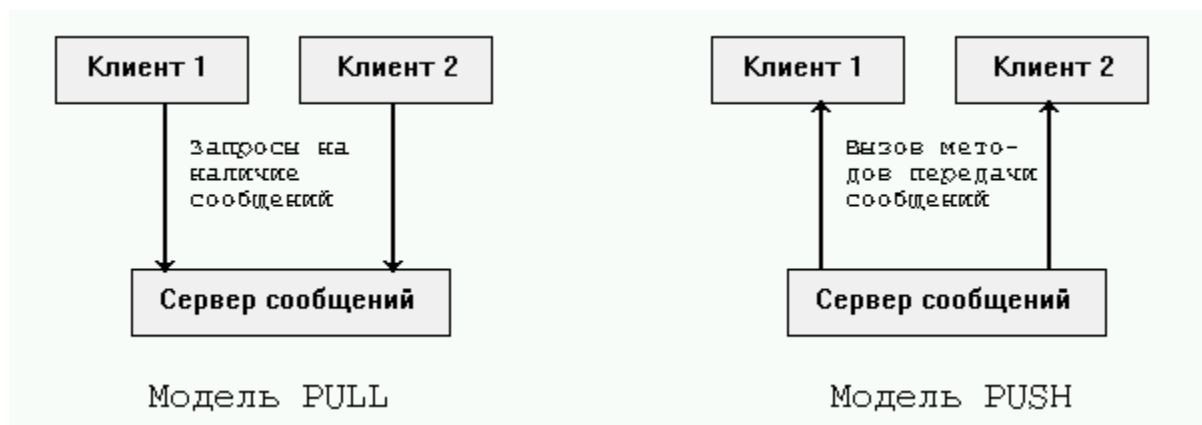


Рисунок 9. Механизмы PULL и PUSH.

При использовании механизма PULL, на обработку каждого запроса клиента сервер

расходует свои системные ресурсы, что при наличии большого числа клиентов, регулярно опрашивающих его, заметно снижает его производительность. Многое также зависит и от того, сколь часто клиенты опрашивают сервер. При большом количестве клиентов-ожидаателей и коротком времени между двумя запросами к серверу, производительность сервера снижается еще больше.

Представим себе, что связь между клиентами и сервером неважная. В этом случае, эффективность работы механизма PULL будет еще снижаться по мере ухудшения качества связи. Врем выполнения запросов будет увеличиваться, а каналы связи заняты.

В модели PUSH сервер сообщений гораздо менее загружен. Сервер освобожден от необходимости регулярно реагировать на вызовы клиентов-ожидаателей. Наоборот, теперь он имеет дело с клиентами-слушателями. ``Вталкивание" сообщений клиенту применяется особенно в тех случаях, когда сообщения, появившиеся на сервере сообщений, должны быть немедленно обработаны клиентом (клиентами). Да и при наличии некачественной связи между узлами, механизм PUSH гораздо более рентабелен, чем PULL — он использует информационный канал лишь один раз для каждого клиента.

При реальных разработках информационной системы, имеют место оба представленных способа взаимодействия компонент. Разумная комбинация компонент информационной системы, поддерживающих PUSH / PULL модели обмена сообщениями, позволяет достичь высокого уровня гибкости и производительности создаваемой информационной системы.

6.2.3. Использование средств Java

Использование PUSH-технологии позволяет практически мгновенно и эффективно пересылать обновленную информацию всем заинтересованным приложениям. При использовании технологии Java-CORBA реализация PUSH-технологии очень проста. У заинтересованного в сообщении пользовательского клиента (апплета) создается специальный слушатель. Этот слушатель запускается основным апплетом в отдельном процессе с использованием механизма *Threads (нитей)* в Java. При запуске слушатель, реализующий специфичный для данного типа сообщений IDL-интерфейс, информирует сервер сообщений о своей заинтересованности в приеме сообщений

данного типа, после чего начинает ждать. При получении сообщения класс-слушатель может уже напрямую взаимодействовать с классом-клиентом.

6.2.4. Применение технологий Java-CORBA и CORBA-CGI

Итак, практика показывает, что технология Java-CORBA лучше всего подходит для создания WWW CORBA-клиентов, которые:

- имеют нестандартный, или не HTML-подобный пользовательский интерфейс
- активно взаимодействуют с другими компонентами информационной системы в течение времени
- должны участвовать в системе и в роли клиента, и в роли сервера.

Технологию же CORBA-CGI выгоднее применять в случае, если:

- идет работа с большими объемами текстовой информации
- системные ресурсы на стороне клиента маломощны.

Несмотря на то, что преимущества технологии Java-CORBA над технологией CORBA-CGI значительны и область применения шире, обе рассматриваемых технологии хорошо подходят для объединения WWW-систем и клиент/сервер-систем. Технология CORBA-CGI расширяет возможности CGI, а технология Java-CORBA возможности всего WWW - — до уровня распределенных объектных систем.

7. Заключение

В результате проделанной работы было показано, что на сегодняшний день технологии Java и CORBA прекрасно дополняют друг друга в качестве универсального, мощного средства для решения проблемы объединения систем, основанных на технологии WWW, с подобными и другими, в особенности распределенными, информационными системами.

Однако уже совсем скоро у технологии CORBA может появиться очень опасный соперник --- продвигаемая Microsoft технология DCOM, которая уже сильно потеснила CORBA с рынка Windows-ориентированных систем. Технология же RMI, наоборот, делает шаги навстречу CORBA. Начиная, видимо, с версии JDK1.2, протокол RMI будет выполняться поверх протокола IIOP, что, конечно же, выгодно всем Java и CORBA разработчикам.

Массовое использование технологии Java-CORBA выведет Internet на совершенно новый уровень взаимодействия. Internet будет все более похож на гигантскую объектную распределенную систему. Тем самым, наконец, произойдет переход от Web, к новой, объектной сети — ObjectWeb.

8. Литература

1. Albertson, T., *Best practices in distributed object application development: RMI, CORBA and DCOM*, http://www.developer.com/news/techfocus/022398_dist1.html; *Distributed object application development: The Java-RMI solution*, http://www.developer.com/news/techfocus/030298_dist2.html"; *Distributed object application development: The Java-CORBA solution*, http://www.developer.com/news/techfocus/030998_dist3.html; *Distributed object application development: The DCOM solution*, http://www.developer.com/news/techfocus/031698_dist4.html.
2. Johnson, J., Skoglund, R., Wisniewski, J., *Program Smarter, Not Harder. Get Mission-Critical Projects Right the First Time*, McGraw-Hill, Inc., 1995
3. Mowbray, T.J., and Zahavi, R., *The Essential CORBA: Systems Integration Using Distributed Objects*. John Wiley & Sons, Inc., 1995
4. Object Management Group, *Event Service Specification*.
5. Orfali, R., Harkey, D., and Edwards, J., *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, Inc., 1996
6. Ахтырченко К.В., Леонтьев В.В., *Распределенные объектные технологии в информационных системах*.
7. Брюхов Д.О., Задорожный В.И., Калиниченко Л.А., Курошев М.Ю., Шумилов С.С., *Интероперабельные информационные системы: архитектуры и технологии*. СУБД, 1995, номер 4.