Интеграция информационных и вычислительных ресурсов Интернета

Приходько С.А., Андрюхин А.И.
Донецкий национальный технический университет
alexandruckin@rambler.ru

Abstract

Prihodko C.A., Andruckin A.I. Integration of the Internet informational and computational resources. science". *The ways of integration of the WWW informational and computational resources are considered. The key condition of integration is an effective search based on metadata. A new object-oriented infrastructure is needed to replace the existing web-services.*

Введение

Несмотря на развитые коммуникационные возможности распространённых операционных систем (ОС), доступ к удалённым ресурсам в них не является прозрачным. Другими словами, ни одна из этих ОС не является истинно распределённой[1]. На протяжении времени были построены множество, как распределённых ОС, так и распределённых систем промежуточного слоя, пользующихся сервисами обычных сетевых ОС. Однако в силу различных причин (технических недостатков, ошибочной политики производителей, и др.) ни одна из таких систем не нашла массового применения. Нераспространение распределённых систем (как ОС, так и систем промежуточного слоя), на наш взгляд, препятствует интеграции информационных и вычислительных ресурсов Интернета, замедляя его развитие.

Постановка проблемы

Данная статья состоит из трёх частей. В первой авторы пытаются осветить современное положение дел в области распределённых систем. В ней приводится обзор некоторых из существующих распределённых архитектур с указанием их преимуществ и недостатков. Вторая часть данной работы посвящена ряду технических и организационных мер, направленных на увеличение возможностей поиска информации в Интернете, способствуя тем самым интеграции его ресурсов. В последней части этой статьи выполняется попытка обосновать необходимость единой распределённой инфраструктуры, которая способна интегрировать ресурсы Интернета в единое информационное и вычислительное пространство. В этой части авторы пытаются обрисовать основные свойства такой системы.

На сегодняшний день парк вычислительных машин, имеющих выход в Интернет, представлен сравнительно малым числом суперкомпьютеров и мейнфреймов, множеством серверов, а также огромным числом персональных, мобильных и встроенных компьютеров. Суперкомпьютеры заняты решением прикладных задач, требующих большого объёма вычислений. Мейнфреймы в основном используются для хранения и обработки бизнес информации больших компаний. Системы этих двух типов, как правило, имеют уникальные ОС, достаточно сильно привязанные к аппаратной архитектуре машины. Подавляющее количество серверов работает под управлением систем семейства Unix. Такие системы используются и в персональных компьютерах, однако, большая часть этого сектора работает под управлением семейства Windows NT. Широким спектром представлены ОС, управляющие мобильными и встроенными компьютерами. К ним относятся системы семейств Unix, Windows CE, QNX, Palm, Symbian и другие.

Каждая из вышеперечисленных ОС поддерживает свои системные вызовы, собственные форматы данных и исполняемых файлов, специальные библиотеки прикладных функций и многое другое. Эти различия сильно затрудняют взаимодействие компьютеров друг с другом. На протяжении лет предпринимались шаги в сторону нивелирования различий ОС путём утверждения и стандартизации различных

[1] Под распределённой системой будем понимать такую систему, которая обеспечивает единообразный доступ к информационным и вычислительным ресурсам управляемых ею компьютеров вне зависимости от физического расположения этих ресурсов.

сетевых протоколов. В результате появились и получили повсеместное распространение протоколы HTTP, FTP, SMTP, POP, Telnet, и многие другие. Эти протоколы решают проблему обмена данными между компьютерами, а также позволяют осуществлять удалённые вызовы, что во многом способствует интеграции Интернета. Однако, несмотря на развитые коммуникационные возможности этих распространённых ОС, у них нет гибкого и единообразного доступа к удалённым ресурсам и следовательно, ни одна из них не является истинно распределённой.

Распределённая система предполагает объединение управляемых ею хостов в единый виртуальный компьютер, распространяя на них как общую среду выполнения, так и общую форму представления данных. Очевидным преимуществом такого подхода является возможность наращивания суммарной вычислительной мощности за счёт одновременного выполнения различных частей программы на разных компьютерах (распределённых вычислений). Менее очевидным, однако, не менее важным преимуществом распределённых систем является объединение данных, хранимых на хостах системы, в единый однородный информационный ресурс. Следует отметить, что последнее достигается лишь при наличии развитых возможностей поиска данных внутри распределённой системы[1].

Обзор распределённых систем

Распределённое выполнение программ реализуется двумя типами систем. Системы первого типа представляют собой т. н. промежуточные слои, призванные устранить различия аппаратных архитектур и программных платформ. Системы второго типа являются полноценными распределёнными ОС, напрямую взаимодействующими с аппаратными средствами компьютера. Оба типа систем дают возможность прикладному программисту абстрагироваться от распределения его приложения между множеством компьютеров. Представим краткий обзор некоторых систем, реализующих распределённое выполнение.

Системы промежуточного слоя. Системы этого типа довольно многочисленны. Большая их часть является результатом исследовательских проектов крупных университетов США и Европы. Большинство таких систем не достигают уровня коммерческой реализации. Для нашего обзора выделим 4 основных проверенных временем архитектур промежуточного слоя: RPC, CORBA, COM, Java/RMI.

RPC(Remote Procedure Call) - технология вызова удалённых процедур, разработанная компанией Sun Microsystems в начале 80-х годов, как часть её UNIX-подобной операционной системы SunOS[2]. Для декларирования вызываемой функциональности RPC использует специальный язык определения интерфейсов (IDL), определяющий входные и выходные параметры вызываемых удалённых процедур (другими словами, их интерфейсы). На основе описания интерфейса IDL-компилятор (в UNIX он называется rpcgen) генерирует на клиентской и серверной машинах специальные программные заглушки. Клиентская заглушка является представителем удалённой процедуры в адресном пространстве клиента, обеспечивая прозрачность её вызова так, как если бы эта процедура была локальной. Серверная заглушка имитирует клиентский вызов процедуры в адресном пространстве сервера. Для осуществления удалённого вызова клиентская заглушка переводит входные данные в стандартизированную форму (при необходимости выравнивает структуры, меняет порядок байтового представления чисел, представление строк, и др.), производит перевод стандартизированного представления данных в последовательный поток байтов и отправляет заявку по сети. Серверная заглушка производит обратные операции (приём заявки, перевод и преобразование потока байтов), адаптируя аргументы клиента к вызываемой процедуре. По окончанию работы процедуры её выходные данные пересылаются клиенту, т.е. повторяется вышеописанный процесс пересылки, но в обратном направлении (рис. 1).

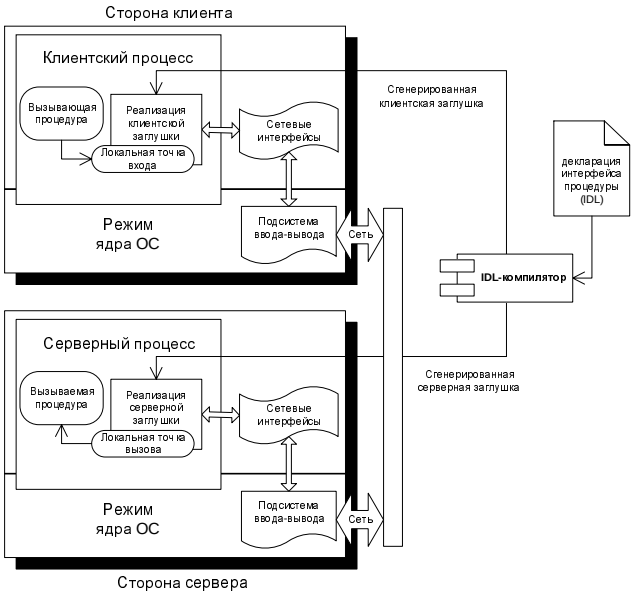

Рисунок 1 – Архитектура RPC.

К преимуществам RPC можно отнести поддержку единой метамодели вызова функциональности, определяемой семантикой IDL. Как следствие, отсутствует привязка к

конкретному языку программирования (клиент или сервер может быть написан на любом языке, для которого определено соответствие с IDL) или программно-аппаратной платформе. Идентичность вызова удалённой процедуры обыкновенному локальному вызову способствует гибкости и универсальности разрабатываемого программного кода.

Недостатком RPC можно назвать отсутствие поддержки объектно-ориентированной метамодели. Кроме того, отсутствует рефлексия, позволяющая во время выполнения программы определять интерфейсы существующих удалённых процедур и вызывать эти процедуры (заглушки компилируются статически).

CORBA (Common object request broker architecture) – спецификация общей архитектуры брокера объектных заявок, принятая некоммерческой организацией OMG (Object Management Group) в 1992 году[3]. Под CORBA понимается спецификация архитектуры, а не конкретная её реализация. Базируется на объектно-ориентированной метамодели, поддерживающей множественное наследование. Объекту реализации соответствует интерфейс IDL, посредством которого тот может обрабатывать заявки. Объект может одновременно выступать как клиентом, так и сервером для других объектов. Вызов функциональности сервера осуществляется посредством объектной ссылки. Кроме статических вызовов (используя заглушки) поддерживается динамическое формирование заявок, основанное на рефлексии. Объектные заявки пересылаются от клиента к серверу при помощи базового сервиса, называемым Брокером объектных заявок (ORB), который скрывает неоднородность и распределение. Каждому объекту ставится в соответствие специальный объектный адаптер, обеспечивающий связь между его бинарной реализацией и брокером объектных заявок. Адаптер отвечает за активацию (загрузку объекта в оперативную память, его подготовку к выполнению заявок) и деактивацию объекта (рис. 2). Метаданные, описывающие типы IDL, хранятся в репозитории интерфейсов, представляющий собой обычный объект CORBA, поддерживающий собственный интерфейс. CORBA определяет множество стандартных сервисов, являющихся надстройкой базовой архитектуры. К таким сервисам относятся сервис распределённых транзакций, сервис именования объектов и другие. Сервис именования поддерживает составные имена объектов, каждое из которых состоит из последовательности имён вложенных пакетов, а также имени самого объекта. По составному имени объекта может быть получена соответствующая объектная ссылка. К преимуществам CORBA отнесём наличие полной технической документации, представленной в виде законченных стандартов. Отметим также поддержку полноценной объектно-ориентированной метамодели с множественным наследованием. Поддержка рефлексивных динамических вызовов, а также единой модели обработки ошибочных ситуаций на основе исключений существенно повышает гибкость и мощность данной архитектуры.

Нельзя, однако, не отметить, что вышеперечисленные преимущества нивелируются за счёт чрезвычайной сложности архитектуры, абстрактности и непродуманности системных интерфейсов. Программирование в существующих средах CORBA является очень трудоёмким процессом. Очень серьёзным недостатком CORBA можно считать отсутствие эталонной реализации архитектуры, из-за чего многие её механизмы на практике оказались трудновыполнимыми или вообще ненужными. Кроме того, CORBA определяет недостаточно мощные механизмы обеспечения безопасности и поддержки версий. Детальный анализ этих и других недостатков данной архитектуры, других причин её неуспеха приведен в [4].

COM (Component Object Model) - компонентная модель объектов, разработанная компанией Microsoft в 1993 году, как развитие технологии OLE[5]. Являясь главным конкурентом CORBA, технология COM имеет очень схожую с ней архитектуру. Получила распространение исключительно на платформе Windows. Первоначально COM была разработана для построения бинарных компонентов на локальном компьютере, доступных сторонним программам независимо от используемого языка программирования. Такие компоненты были необходимы, в частности, для интеграции документов приложений Microsoft Office и поддержки отображения активных web-страниц браузером Internet Explorer. Позже Microsoft была добавлена возможность удалённого вызова функциональности COM-объектов. Поддерживается объектно-ориентированная метамодель на основе множественного наследования интерфейсов, а также рефлексивные динамические вызовы. Наследование реализации не поддерживается. Технология COM во многом ориентирована на достижение высокой эффективности. Как следствие, она предоставляет программисту меньший уровень абстракции по сравнению с CORBA. Например, COM-технология не поддерживает исключения. Вместо них в качестве результата операции возвращается числовое значение HRESULT, по которому прикладной программист должен судить о возможных ошибках выполнения заявки. Как и CORBA, COM поддерживает иерархически структурированное пространство имён. В качестве преимущества COM отметим тот факт, что COM-компоненты обладают исключительно

высокой производительностью (т.е. издержки при вызове функциональности минимальны).

К недостаткам архитектуры COM отнесём отсутствие поддержки наследования реализации, высокую сложность разработки и развёртывания COM-компонентов, а также низкую степень абстракции базовых интерфейсов. Кроме того, COM-технология реализована только для платформы Windows.

Java/RMI (Remote Method Invocation in Java) – удалённый вызов методов языка Java, реализуемый Sun Microsystems, начиная с версии 1.1 1996 года[6].Промежуточный слой Java/RMI представлен виртуальной машиной, интерпретирующей независящий от платформы байт-код. Интерфейс прямо или косвенно расширяющий системный интерфейс Java.rmi.Remote называется удалённым интерфейсом.

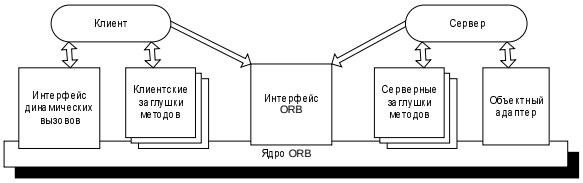

Рисунок 2 – Архитектура CORBA.

Если объект поддерживает удалённый интерфейс, то он называется удалённым объектом, и соответствующие ему методы могут быть вызваны из другой Java-машины на локальном или удалённом хосте. Рефлексия и динамические вызовы языка Java не распространяются на удалённые вызовы объектов. Реестр Java/RMI реализует именование объектов, но без непосредственной поддержки иерархичности пространства имён. Привязка имён к объектам возможна лишь для локальных процессов содержащего реестр хоста, что нарушает прозрачность доступа. К преимуществам Java/RMI можно отнести прозрачность вызова функциональности удалённых объектов, поддерживаемая средствами языка Java. Это, несомненно, положительно сказывается на простоте создания распределённых приложений. Нельзя не отметить также высокую переносимость, обеспечиваемую виртуальной машиной.

К недостаткам Java/RMI можно причислить низкую производительность, связанную с применением виртуальной машины. Среди языков программирования поддерживается только Java. Кроме того, Java/RMI не поддерживает рефлексию и прозрачный доступ к реестру именования.

Распределённые операционные системы. Среди распределённых операционных систем фактически не наблюдается ни одной коммерчески успешной. Для нашего обзора выделим три системы, представляющие собой законченные программные продукты, практически достигшие уровня коммерческих реализаций. Мы рассмотрим локальную распределённую систему Amoeba, а также две системы семейства Plan 9 (собственно, Plan 9 и её прямого потомка Inferno), доводящие парадигму UNIX до логического завершения.

Amoeba - распределённая система для локальной сети, спроектированная и реализованная проф. Э. Таненбаумом и его студентами в начале 90-х годов[7].Система базируется на группе машин единой архитектуры (поддерживается x86, SPARC, и другие), соединённых в локальную сеть. Архитектура Amoeba гетерогенна - каждая машина может быть пулом процессоров, сервером (именования, файлов и др.) или терминалом (рис. 3). Взаимодействие между процессами базируется на RPC. В целях обеспечения высокой производительности создатели Amoeba отказались от свопинга – все процессы располагаются в оперативной памяти. Система самостоятельно справляется с распределением ресурсов и балансировкой нагрузки.

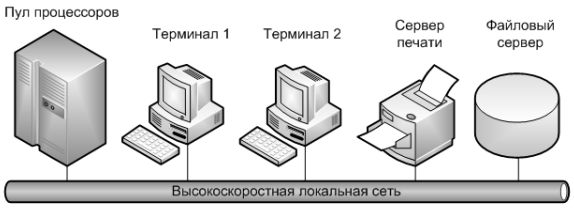

Рисунок 3 – Архитектура Amoeba.

Таким образом, Amoeba предоставляет пользователю простой интерфейс, имитируя единый виртуальный компьютер. Система практически в полном объёме поддерживают стандарт POSIX на уровне исходных кодов. Эту возможность обеспечивает специальная библиотека AJAX для эмуляции системных вызовов POSIX. К преимуществам Amoeba можно отнести поддержку интерфейса нераспределённой ОС, управляющей одним компьютером. Кроме того, Amoeba имеет библиотеку эмуляции POSIX-вызовов, которая обеспечивает переносимость исходного кода UNIX на Amoeba. Также нужно отметить высокую производительность системы, ориентированной, прежде всего, на решение вычислительных задач. Недостатком Amoeba можно считать плохую масштабируемость, связанную с гетерогенностью её иерархичной структуры, что не позволяет системе перешагнуть уровень локальной сети. Кроме того,

конфигурация системы предполагает наличие машин лишь одной архитектуры. К недостаткам Amoeba можно также отнести и то, что интерфейс системных вызовов базируется на процедурной метамодели и абстракции файла.

Plan 9 – распределённая операционная система, разрабатываемая Bell Labs с конца 80-х годов и призванная заменить UNIX в эру преобладания персональных рабочих станций [8].Система Plan 9 значительно развивает файловую парадигму UNIX – практически всё в системе, включая устройства, процессы, удалённые хосты и сетевые протоколы представлены в виде файлов. К файлу могут быть применены операции открытия, закрытия, чтения и записи. Системный интерфейс ориентирован на работу с текстовыми строками в кодировке Unicode (UTF-8). Так, например, для передачи команды устройству необходимо открыть соответствующий ему файл в папке /dev и записать в него текстовую строку в формате, понятном данному устройству. Ресурсы распределённой системы представлены в виде иерархии вложенных пространств имён файлов, причём общая структура дерева файлов является стандартной для любого хоста. Например, для каждого компьютера под управлением Plan 9 в папке /proc располагаются файлы текущих процессов. Пространства имён могут монтироваться к другим пространствам, копироваться, демонтироваться и удаляться. Такая функциональность позволяет получать идентичную среду на произвольном компьютере – пользователю нужно лишь зарегистрироваться под своей учётной записью. Манипулирование ресурсами производится при помощи простого, но мощного протокола 9P, команды которого обозначают атомарные файловые транзакции. 9P опирается на специальный транспортный протокол IL поверх IP и заменяющий собою другие протоколы транспортного уровня (TCP, UDP). Не смотря на последовательность разработчиков в реализации идеи «всё есть файл” Plan 9 имеет системные вызовы, не связанные с файлами (создание процессов, работа с разделяемой памятью и др.). Наряду с собственным интерфейсом прикладного программирования, система также обеспечивает среду эмуляции POSIX. К преимуществам Plan 9 можно отнести простоту и гибкость распределённой архитектуры, ориентированной на работу с файлами. Наличие среды эмуляции POSIX решает проблему перенесения известных UNIX-приложений под Plan 9, облегчая тем самым миграцию пользователей в эту систему.

Господство файловой парадигмы кроме своих преимуществ имеет и множество недостатков. Главный недостаток заключается в ограниченности абстракции файла, которая во многих случаях не является удобной для описания сложной функциональности.

Inferno – распределённая операционная система, разработанная в 1996 году английской компанией Vita Nuova во главе с Бобом Пайком, главным архитектором Plan 9, и продолжающая её архитектурную линию[9]. Inferno представляет собою компактную легко переносимую операционную систему, архитектурно схожую с Plan 9. Inferno может запускаться как в качестве полноценной операционной системы, надстраивающейся над оборудованием компьютера, так и в качестве приложения под распространёнными операционными системами, обеспечивая пользователям идентичную среду независимо от машинной архитектуры и способа своей загрузки. В Inferno роль протокола 9P играет усовершенствованный протокол Stix. Потоки программ выполняются в едином адресном пространстве. Защиту памяти обеспечивает виртуальная машина Dis, промежуточный код для которой может выполняться как в режиме интерпретации, так и в режиме JIT-компиляции. Интерфейс ядра представлен в виде вызовов системных компонентов через инструкции промежуточного кода. Dis поддерживает оптимизированную сборку мусора. Основным языком программирования высокого уровня в Inferno является компактный язык Limbo имеющий C-подобный синтаксис, но концептуально наследующий от компонентного языка Oberon. Ядро системы имеет развитую модель безопасности, включая встроенную поддержку шифрования. Множество архитектурных преимуществ Inferno были унаследованы от системы Plan 9. Выявляя собственные положительные стороны Inferno, нельзя не отметить следующие качества. Имеется возможность запуска, как в режиме операционной системы, так и в режиме приложения под существующую систему. Реализация виртуальной машины Dis проста, удобна и эффективна. Единый промежуточный код, не зависящий от архитектуры компьютера, обеспечивает полную переносимость программ на любые системы под управлением Inferno. Язык Limbo компактен и гибок, кроме того, содержит средства распределённого программирования. Система имеет встроенную поддержку безопасности на уровне ядра. К недостаткам Inferno можно отнести ограниченность файловой парадигмы, перешедшая к ней по наследству от системы Plan 9. Собственным недостатком данной системы можно считать отказ от использования аппаратной защиты памяти, приводящий к принципиальной невозможности безопасного выполнения высокоэффективных программ на машинном языке данного компьютера.

Увеличение возможностей поиска

Вышеприведённый обзор даёт некоторое представление о современном состоянии дел в сфере распределённых систем. По мнению авторов, несмотря на серьёзные теоретические и практические наработки в данной области, распределённые системы до сих пор оказывают малое влияние на мир Интернета, что негативно сказывается на интеграции его информационных и вычислительных ресурсов.

Интеграция ресурсов во многом опирается на возможность и удобство поиска необходимой информации. Рассмотрим вкратце существующие на сегодняшний день возможности глобального поиска информации в Интернете.

Главный поисковый сервис в Интернете предоставляют т.н. поисковые машины, индексирующие содержание web-страниц Всемирной Паутины. Главенствующим принципом их поиска является нахождение в содержании web-страницы текстовых соответствий символьным последовательностям поискового запроса (в более продвинутых системах имеются эвристические анализаторы, учитывающие словоформы лексем и неточные совпадения). Поисковые машины помогают также отыскивать файлы, однако такой поиск не является эффективным в силу своей текстовой природы (бинарным файлам ставятся в соответствие имена, часто несоответствующие их содержанию). Главенствующим файловым ресурсом сегодня становятся пиринговые (одноранговые) сети. Для поиска файлов внутри таких сетей, как правило, применяется хеширование. Имени и некоторым атрибутам реплик файла ставится в соответствие хэш-код его содержимого. Таким образом, пиринговые сети обеспечивают лишь поиск по строгому соответствию, не учитывая сходства по каким либо признакам.

Как известно, поиск информации может проводиться как по строгому соответствию (в теле запроса содержится подмножество искомой информации, или её хэш-код), так и по некоторой специализированной информации, описывающей искомые данные (метаданным). Примером метаданных могут служить атрибуты файлов, такие как дата и время создания. Легко видеть, что поиск, основанный на метаданных, обладает большей мощностью, позволяя формулировать более абстрактные обобщённые запросы. К сожалению, поисковые машины и пиринговые сети в основном поддерживают поиск по точному соответствию, что существенно ограничивает их возможности.

Рассмотрим подробнее основные черты инфраструктуры, обеспечивающей поиск по метаданным. Разным видам данных должны соответствовать различные виды метаданных. Так, например, у видеофильма могут быть атрибуты, описывающие название картины, год создания, имя режиссера, алгоритм декомпрессии и др. Иными словами, для эффективного поиска требуется, чтобы метаинформация, описывающая данные любого типа, была структурирована некоторым стандартным для данного типа образом.

Легко видеть, что широко распространённая файловая модель хранения данных плохо удовлетворяет поставленным требованиям структурирования. В самом деле, любая непрерывная последовательность байтов может рассматриваться в качестве файла, т.е. за структурирование данных внутри файла ответственен программист, а не сама система. Таким образом, файловая система структурирует информацию лишь на уровне иерархии файлов и директорий на диске. Для большинства из существующих форматов файлов соответствующая метаинформация представлена в неформальной форме, в виде текста документации на естественном языке, что не позволяет её использовать поисковыми алгоритмами. Таким образом, организация поиска, основанного на метаданных, по мнению авторов, требует замены существующей файловой инфраструктуры хранения данных.

На наш взгляд, объектно-ориентированный подход к представлению и хранению данных и алгоритмов является одним из предпочтительных. На самом деле, несложно представить инфраструктуру хранения, основной единицей которой будет являться объект, как экземпляр некоторого класса (алгоритмы и метаданные классов могут инкапсулироваться в экземпляры класса типов). В некотором роде такое представление можно рассматривать в качестве отображения классической реляционной модели. Так классы уподобляются таблицам, их экземпляры – записям, методы – хранимым процедурам, а триггеры - событиям. Следовательно, многие принципы, применимые к проектированию и реализации баз данных будут применимы и при данной системе хранения. Так, например, для ускорения поиска объектов можно применять индексирование, подобно тому, как это делается в реляционной СУБД.

Наличие развитой инфраструктуры хранения данных само по себе не обеспечивает мощные поисковые возможности. Данные одной и той же природы могут иметь различные формальные типы. Предположим, что система учёта почтовых отправлений оперирует адресами получателей в виде кортежей типа “(индекс, страна, город, улица, дом, квартира)”. Предположим также автоматизированную систему кредитования некоторого банка, работающую с адресами заёмщиков. Эти адреса

представлены в виде кортежей типа “(страна, город, улица, дом, квартира, индекс)”. Ясно, что кортежи обоих типов означают одно и то же, однако, поскольку они проектировались разными организациями, они не являются совместимыми друг с другом. Так, например, при автоматизированной проверке факта почтовой пересылки дорогостоящих покупок на адрес неплательщика придётся учитывать это различие представлений адреса в обеих системах, что усложнит поиск. Некоторые простые универсальные типы данных используются разработчиками в качестве стандартных, например, size_t в стандартной библиотеке Си или интерфейс System.Collections.ICollection в библиотеке классов Microsoft .NET Framework, однако большинство типов разрабатываются и используются несогласованно. Такая разобщённость типов хранимых данных существенно препятствует возможности поиска информации. Другими словами, построение мощной поисковой системы требует принятия некоторых организационных мер, направленных на стандартизацию типов.

По мнению авторов, решением данной проблемы будет являться создание некоммерческой организации, контролирующей специализированную распределённую базу типов, открытую для всеобщего доступа. В частности, при наличии стандартной объектно-ориентированной инфраструктуры такая база должна хранить интерфейсы и классы реализации (и те и другие должны представлять собой готовые компоненты, содержащие в себе документирующие атрибуты).

Облегчить ориентирование во множестве типов призван перманентный процесс их стандартизации. Стандартизировать классы реализации не представляется возможным из-за различия аппаратных платформ, постоянного совершенствования алгоритмов, а также различных ограничений, налагаемых на ту и иную реализацию (например, более быстрый алгоритм может потреблять большое количество памяти, что неприемлемо для некоторых случаев). Стандартизация же интерфейсов кажется целесообразной, несмотря на определённые трудности (к ним, в частности, относится частая зависимость интерфейса от алгоритмов и внутреннего устройства классов реализации).

Вопросы устройства и администрирования базы типов, а также начального мотивирования к её использованию нетривиальны и выходят за рамки данной статьи. Мы рассмотрим лишь некоторые аспекты работы такой базы. Кроме обеспечения открытого доступа к типам (любой желающий вправе получить хранящиеся интерфейсы и классы реализации, а также разместить собственные) управляющая организация время от времени проводит итерации стандартизации интерфейсов. В процессе стандартизации выявляются интерфейсы, уже являющиеся в сообществе разработчиков стандартами de-facto. Другими словами, в процессе эксплуатации нестандартные интерфейсы будут ”бороться” друг с другом в конкурентной борьбе. Можно предположить, что с течением времени разработчики начнут ориентироваться на наиболее удачные интерфейсы с точки зрения продуманности функционала и наличия качественных классов реализации. Таким образом будут “выкристаллизовываться” de-facto стандартные интерфейсы, которые и перейдут в официальный стандарт при очередной итерации стандартизации (рис.4). Каждая такая итерация предполагает пересмотр множества стандартных интерфейсов путём добавления новых и удаления устаревших.

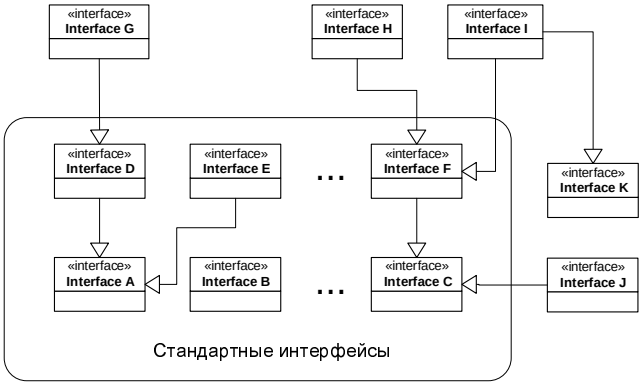

Рисунок 4 – Пример иерархии интерфейсов, хранимых в базе типов

Наличие множества стандартных интерфейсов позволит проводить эффективный поиск на основе метаданных. Так, например, для нахождения фильмов Педро Альмодавара, созданного режиссером в период с 1998-го по 2001 год, может быть составлен формальный запрос со следующей семантикой: ”найти экземпляр A класса B, поддерживающего стандартный System.Multimedia.IVideoDocument, интерфейс , причём для A значение свойства IVideoDocument.Director = ‘Pedro Almodavar’, а значение свойства IVideoDocument.ReleaseDate > 31.12.1997, но < 01.01.2002”.

По мнению авторов, стандартизация интерфейсов, кроме повышения эффективности поиска также поможет вывести на новый уровень, как унификацию компонентов, так и возможности масштабирования программных систем.

Распределённая инфраструктура для нового Интернета

Итак, по мнению авторов данной статьи, достичь нового уровня интеграции ресурсов Интернета поможет новая распределённая

объектно-ориентированная инфраструктура (РООИ). Такая инфраструктура могла бы стать для платформы WEB 3.0 тем базисом, которым на сегодняшний день являются web-сервисы для WEB 2.0.

Попытаемся выделить основные черты необходимой инфраструктуры. С одной стороны, она должна поддерживать системные вызовы, на которых смогут базироваться все типы существующих сегодня сервисов WEB 2.0. Т.е. на основе них должны легко реализовываться интерактивный графический web-интерфейс (то, что сейчас реализуется при помощи AJAX), децентрализация, фольксономия (совместная плоская категоризация информации), RSS (распространение информации по подписке) и другие подходы, характерные для WEB 2.0 [10]. С другой стороны, новая инфраструктура должна оставаться достаточно низкоуровневой, не навязывая собственные стратегии репликации, транзакций, балансировки нагрузки, выявления взаимоблокировок и др., т.к. те в большой степени зависят от конкретной решаемой задачи. Все эти возможности при необходимости могут быть реализованными на более высоких программных уровнях, опирающихся на системные вызовы РООИ.

Предлагаемая инфраструктура должна быть однородной и децентрализованной, надстраиваясь над множеством равноправных хостов. В качестве хоста может служить персональный компьютер, web-сервер, мобильный телефон, или даже встроенный контроллер. Различие платформ будет сглаживать виртуальная машина, работающая с единым форматом представления объектов.

Подсистемы ядра РООИ. Для развёртывания РООИ каждый хост должен иметь по экземпляру специального программного ядра, а также набор необходимых сервисов, надстраивающихся над ним. Выделим основные подсистемы ядра РООИ:

1.Подсистема объектов;
2.Подсистема среды выполнения;
3.Подсистема безопасности;
4.Подсистема памяти и хранения;
5.Подсистема сетевого взаимодействия.

Подсистема объектов. Важнейшей задачей подсистемы объектов является формирование объектно-ориентированной метамодели – философии РООИ. Практически все системные вызовы формально относятся к этой подсистеме. В частности, к ней относятся вызовы создания объектов, работы с ними, их перемещения и удаления (на самом деле, эта функциональность является результатом согласованной работы множества подсистем ядра).Рассмотрим подробнее некоторые особенности объектно-ориентированной метамодели. Её базовой единицей является объект – экземпляр некоторого класса (как было упомянуто выше, метаинформация, описывающая классы, инкапсулируется в экземпляры специального класса типов). Каждый объект ссылается на объект типа, описывающий его класс и находящийся в одном с ним адресном пространстве (рис. 5).

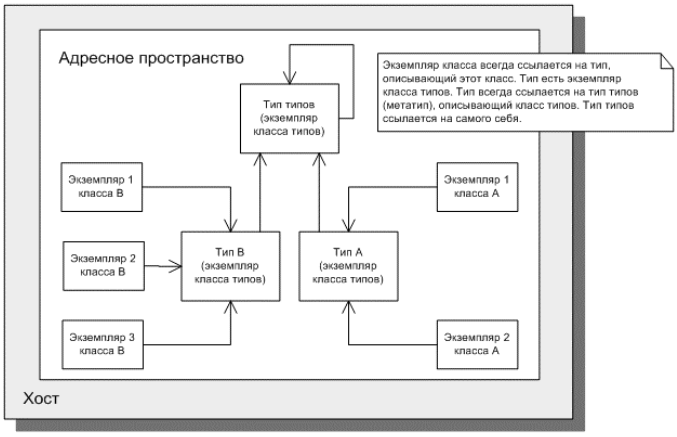

Рисунок 5 – Объекты и их типы

Подсистема среды выполнения. Код объектов работает в специальной среде выполнения, предоставляющей необходимые системные сервисы (адресное пространство, обработка исключительных ситуаций, многопоточность, и др.). Среда выполнения кода определяется соответствующей подсистемой ядра. Ядро РООИ должно содержать элементарные механизмы поддержания целостности объектов. В самом деле, объекты могут использоваться несколькими клиентами одновременно, и разработчику класса будет удобнее работать со встроенными примитивами синхронизации, а не реализовывать собственные. Эти примитивы будут предоставляться подсистемой среды выполнения. Более сложные механизмы поддержания целостности в виде плоских и вложенных транзакций должны реализовываться вне ядра в виде вышележащих программных слоёв. Опыт разработки сверхнадёжных операционных систем, таких как EROS [11], подтверждает полезность некоторых специфических механизмов поддержания целостности. К ним, в частности, относится возможность приостановки работающей программы, сохранения её текущего состояния, сохранения состояния всей системы в произвольный момент времени, а также возможность отката к предыдущему состоянию. Снятие снимка работающей программы, кроме увеличения надёжности, ещё и улучшает возможность масштабирования – в процессе своей работы программа может быть перенесена на более мощный или менее загруженный хост. Поэтому, проектируя подсистему среды выполнения, было бы целесообразным

предусмотреть возможность построения вышеперечисленных сервисов на её основе.

Подсистема безопасности. Распределённая инфраструктура должна опираться на единую модель безопасности. Однако, учитывая современные бурно развивающиеся технологии, нельзя привязываться к конкретным алгоритмам шифрования. Поэтому, модель безопасности должна быть в достаточной степени расширяемой.

Подсистема памяти и хранения. За выделение памяти под объекты отвечает подсистема памяти и хранения. Некоторые объекты в РООИ являются долгоживущими, т.е. могут сохраняться на внешних энергонезависимых носителях. Следовательно, при необходимости долгоживущий объект должен быть загружен с устройства хранения в оперативную память или обратно. Эта деятельность должна быть прозрачной для пользователя, подобно работе подсистемы свопинга в современных ОС.

Подсистема сетевого взаимодействия. За взаимодействие хостов отвечает подсистема сетевого взаимодействия. В её обязанности входит сетевой обмен между хостами по специальному протоколу управления объектами, обеспечивающему перемещение объектов, а также удалённый вызов их функциональности.

Вопреки популярности протокола HTTP, передающего данные в текстовых форматах HTML и XML, авторы отдают предпочтение бинарному протоколу по следующим причинам. Текстовые форматы удобны для обмена данными между различными несовместимыми друг с другом платформами. Так, на любой из них можно открыть присланный текстовый файл простым редактором. В РООИ эту функцию будет выполнять универсальный браузер объектов (аналог-дизассемблер сборок ILDasm.exe платформы Microsoft .NET), способный открывать любые данные. Таким образом, в рамках предлагаемой инфраструктуры единый бинарный формат обладает главным преимуществом стандартных текстовых форматов, будучи лишён их недостатков. К последним относятся избыточность и сложность обработки, что не всегда является приемлемым для обмена данными с маломощными или мобильными хостами (например, для обмена с мобильным телефоном через низкоскоростную радиосеть). Кроме того, текстовые форматы не подходят для пересылки больших объёмов данных (например, видео), а также налагают значительные ограничения на обмен информацией в реальном времени. Итак, подсистема сетевого взаимодействия базируется на едином бинарном протоколе управления объектами. Некоторые объекты могут быть доступными за пределами своих адресных пространств. Такие объекты имеют идентификатор, представляющий собой большое число, уникально идентифицирующее объект. Идентификатор генерируется во время конструирования объекта и остаётся неизменным весь период его жизни.

К важнейшим системным вызовам относится получение объектной ссылки по идентификатору объекта (вне зависимости от хоста, на котором располагается объект в данный момент). За поиск хоста, содержащего искомый объект, а также связь с ним отвечает подсистема сетевого взаимодействия.

Активные объектные ссылки должны оставаться в актуальном состоянии при миграции объекта в иное адресное пространство (включая миграцию на другой хост). Поддержка актуальности объектных ссылок также возлагается на эту подсистему.

Сервисы РООИ. Сервисы РООИ надстраиваются над ядром, расширяя его функциональность. К ним можно отнести сервис поиска, сервис транзакций, сервис репликации, сервис балансировки нагрузки, и другие (рис. 6).

Рисунок 6 – Ядро и сервисы РООИ

Поскольку сервис поиска имеет непосредственное отношение к данной статье, рассмотрим его подробнее.

Сервис поиска. Поисковая функциональность заключается в поддержке индексирования, а также во встроенном механизме нахождения объектов, оптимально использующем множество имеющихся индексов. Важно отметить, что индексирование по значениям свойств объектов[2] в общем случае мощнее индексирования по значениям их полей.

[2] Здесь под свойством будем понимать метод, возвращающий некоторое значение и при этом не изменяющий внутреннего состояния объекта.

В самом деле, код свойств может гибко преобразовывать значения полей из формы, удобной для хранения, в форму, удобную для сравнения во время индексации и поиска. Кроме того, индексы могут строиться не только на основе сравнения числовых и строковых значений, как практикуется в реляционных базах данных, но и на основе переопределённых операций сравнения объектов, что, опять же, повышает гибкость поиска.

Поддержка поиска не означает наличия транслятора SQL-подобных запросов. Поисковый запрос, на наш взгляд, лучше представлять в бинарном виде, как экземпляр класса запросов. На прикладном же уровне будут находиться трансляторы с универсальных и специализированных языков, позволяющие формулировать поисковые запросы в удобном для пользователя виде.

Заключение и реализация РООИ

Обрисовав основные очертания предлагаемой нами распределенной объектно-ориентированной инфраструктуры для будущего Интернета, попытаемся представить её перенос из абстрактного мира концептов в реальную жизнь. Программное обеспечение, реализующее такую инфраструктуру, может представлять собой как полноценную ОС, напрямую работающую с ресурсами компьютера, так и систему промежуточного слоя, надстраивающуюся над распространёнными ОС. Очевидно, что новая, никому не известная ОС, которая, к тому же, и не совместима с UNIX (что делает невозможным перенесение множества полезных программ), никогда не завоюет критическую массу пользователей. Потому единственно возможным путём представляется разработка системы промежуточного слоя, устанавливаемой поверх популярных ОС. Предполагаем, что такая система может стать популярной, используясь на первых порах лишь в качестве клиента пиринговой сети с повышенными поисковыми возможностями. Кроме того, на ранних этапах разработки было бы удобней отвлечься от низкоуровневого программирования аппаратного обеспечения. В случае распространения такой системы появится смысл её реализации в виде полноценной ОС на все распространённые платформы. При этом оба типа реализаций системы должны предоставлять идентичный интерфейс и полную совместимость друг с другом.

Литература

1. Anderson D. P. and Kubiatowicz J .The Worldwide Computer - An operating system spanning the Internet would bring the power of millions of the world's Internet-connected PCs to everyone's fingertips, Scientific American, March 2002.
2. Marshall A.D.,“Programming in C - UNIX System Calls and Subroutines using C“, A.D., 1994-2005;
3. Emmerich W. «Engineering distributed objects“, , University College London, John Wiley & Sons, Ltd, 2000.
4. Henning M.The Rise and Fall of CORBA, ACM Queue, Volume 4, Number 5, June 2006;
5. Don Box.Essential COM. Addison-Wesley Professional, 464 pp, 1997;
6.Grosso W. Java RMI, , O’Reilly Media, 572 pp, 2001;
7. Mullender S.J., van Rossum G., Tananbaum A.S., van Renesse R., van Staveren H.. Amoeba: a distributed operating system for the 1990s,. – Computer Volume 23, Issue 5, May 1990;
8. Pike R., Presotto D., Dorward S., Flandren B., Thompson K., Trickey H., Winterbottom P.Plan 9 from Bell Labs., Bell Laboratories, Murray Hill;
9.“Inferno: An Overview. Operating system for network application development”, Vita Nuova;
10. O’Reilly T.”What Is Web 2.0. Design Patterns and Buisness Models for the Next Generation of Software”, 09/30/2005;
11. Shapiro S.J.,“EROS: A Principle-Driven Operating System from the Ground Up”, Johns Hopkins, University Norm Hardy, Agorics, Inc.