



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Licenciatura en Ciencias de la Computación

Tesis de Licenciatura

Desarrollo de Sistemas Distribuidos en Internet

Director: Dr. Claudio Righetti
Co-Directora: Lic. María Eva Lijding
Tesistas: Biscotti, Fabián
Ronconi, Leandro

2000



mencionar lentitud en la ejecución, falta de facilidades gráficas propias, entorno nativo de programación DOS y falta de documentación.

El haber diseñado e implementado de un sistema de distribución de documentos para Internet, desarrollado enteramente en Java, que hoy es uno de los lenguajes mas



Tesis de Licenciatura

Indice

RESUMEN	3
BREVE RESUMEN POR CAPÍTULOS	6
CAPÍTULO 1	
Comparación de Arquitectura y Lenguaje	10
CAPÍTULO 2	
Sistema a implementar	44
CAPÍTULO 3	
Necesidades del sistema	47
CAPÍTULO 4	
Decisiones de implementación del prototipo	50
CAPÍTULO 5	
Desarrollo del Agente de Servicio	61
CAPÍTULO 6	
Desarrollo del Agente de Usuario	75
CONCLUSIONES	81
GLOSARIO	85
BIBLIOGRAFÍA	88
ANEXO I	
Experiencias de la Implementación	90
ANEXO II	
Comparación Java con C++	95
ANEXO III	
Manual de instalación	119
ANEXO IV	
Manual del usuario	129
ANEXO V	
Base de Datos	140



Resumen

El presente trabajo tiene el objetivo de identificar por medio de un análisis comparativo, las arquitecturas y lenguajes mas apropiados para el desarrollo de sistemas distribuidos; también diseñar e implementar un prototipo de un modelo de distribución de documentos, que trabaje en el ambiente de Internet, permitiendo plasmar parte de las investigaciones realizadas y aportar una solución particular a la distribución de objetos.

Se realiza una comparación de arquitecturas y lenguajes para determinar las principales características que cada uno posee para desarrollar sistemas distribuidos, considerando las necesidades actuales en cuanto a seguridad, facilidades de implementación, manejo de objetos remotos, multiplataforma y tipo de comunicación en Redes de Area Amplia y especialmente en Internet.

El análisis entre diferentes arquitecturas o entornos de programación incluye a JDK, CORBA y DCOM que han sido identificados como algunos de los mas adecuados para la implementación de sistemas distribuidos, debido a que poseen la capacidad de integrar usuarios dispersos geográficamente, con diferentes equipamientos, a través de plataformas robustas y flexibles de desarrollo.

Como conclusión sobre las arquitecturas analizadas podemos decir que en general presentan funcionalidades similares para la implementación de sistemas distribuidos. Las diferencias conceptuales se basan en que tanto CORBA como DCOM integran elementos heterogéneos de software, mientras que RMI-JDK tiene funciones de conectividad y recursos compartidos por medio de un lenguaje nuevo y propietario llamado Java.

También se realiza una comparación entre los lenguajes de programación Java y C++ en cuanto a sus características más importantes, considerando que estos son dos de los lenguajes mas utilizados en la actualidad para implementaciones en Internet.

De la comparación de los lenguajes podemos concluir que C++ ejecuta los programas en un tiempo considerablemente menor. Java en tanto, tiene un mejor manejo de la memoria y presenta como uno de los puntos más destacados, la filosofía “write one, run anywhere”, lo que implica que un programa compilado puede ejecutar en cualquier ordenador que posea la máquina virtual de Java.

El diseño del sistema distribuido a implementar corresponde al modelo del Sistema de Distribución de Objetos (ODS) [RNL 97]. El Sistema de Distribución de Objetos utiliza los recursos ya existentes en Internet e intenta optimizar su uso a partir de la implementación de redes de ruteo y redes de distribución, para hacer que los objetos estén disponibles para los usuarios cuando estos los necesiten. ODS define el acceso a copias locales de los objetos del sistema en vez acceder al lugar de origen cada vez que se necesita un objeto y además, hace que este esté disponible localmente en forma transparente para el usuario.

Las características mas relevantes que se identificaron para el desarrollo de este sistema son: base de datos independiente del lenguaje de programación, lenguaje



portable para poder ejecutarse en diferentes sistemas operativos, posibilidad de implementar esquemas de seguridad y manejo de diferentes tipos de comunicaciones.

Como resultado de las diferentes comparaciones y análisis de arquitecturas y lenguajes, se selecciona el lenguaje Java bajo el entorno de desarrollo JDK para implementar el prototipo, con el objetivo de verificar en forma práctica las características de Java y JDK y comprobar si el sistema distribuido planteado es una buena solución para proveer a usuarios finales de objetos que se encuentran distribuidos en la red.

Se implementó el Agente de Servicio que es el componente básico del Sistema de Distribución de Objetos y el Agente de Usuario como front end de acceso para interactuar con el sistema. Se limitó la distribución de objetos en general a la distribución de documentos, utilizando un sistema de publicación en el cual cada autor puede editar publicaciones en los temas de interés en los cuales está autorizado. La administración del sistema se realiza en forma centralizada en los Agentes de Servicio. Se establecieron dos tipos de conexión, una punto a punto para los casos en que se necesita garantizar la comunicación y otra utilizando datagramas donde no es imprescindible una conexión fiable.

El Agente de Servicio tiene como objetivo el ingreso y transmisión de documentos a través de la red en forma transparente para los usuarios. Está desarrollado en diferentes módulos independientes pero relacionados y complementarios funcionalmente dándole al sistema mayor flexibilidad y robustez.

El Agente de Usuario es la interfase que le permite a quién utiliza el sistema realizar operaciones sobre los documentos. Se comunica con el Agente de Servicio sin que el usuario necesite conocer donde reside, logrando así transparencia de localización. Las operaciones sobre los documentos, tales como altas, bajas, modificaciones y consultas, dependen del perfil que tenga asignado el usuario.

Los objetos que maneja el sistema constan de dos partes fundamentales, la primera corresponde a la meta información, que contiene atributos propios del objeto necesarios para su administración e información resumida del documento. La segunda parte es el documento propiamente dicho, es decir, el contenido de las publicaciones.

A modo de conclusión sobre el diseño e implementación del prototipo podemos decir que cubrió satisfactoriamente nuestras expectativas. Podemos comprobar que el sistema es adecuado para transmitir objetos sin un alto grado de actualización, tales como documentos en general, aportando además a la optimización del tráfico en la red dado que se accede a copias locales de los mismos. Con respecto al desarrollo técnico podemos mencionar la posibilidad de ejecutar el código generado en diferentes plataformas y el acierto de haber utilizado un modelo cliente-servidor, que nos permite intercambiar componentes de software en forma fácil y flexible. Los beneficios de haber elegido Java como lenguaje de programación fueron capitalizados a través de programas con menos líneas de código, manejo automático de memoria, facilidad en la administración de errores y la posibilidad de convertir funciones en applets para ser ejecutados en browser estándares. De las deficiencias encontradas en Java podemos



mencionar lentitud en la ejecución, falta de facilidades gráficas propias, entorno nativo de programación DOS y falta de documentación.

El haber diseñado e implementado de un sistema de distribución de documentos para Internet, desarrollado enteramente en Java, que hoy es uno de los lenguajes mas utilizados, representa uno de los logros más importantes que podemos citar. También el conocimiento adquirido en el estudio de las principales alternativas para la implementación de sistemas distribuidos, tanto a nivel de arquitecturas como de lenguajes, nos brinda la posibilidad de poder elegir la solución mas adecuada para cada necesidad.



Breve resumen por Capítulos

Capítulo 1

Comparación de Arquitectura y Lenguaje

Existen actualmente diferentes modelos o arquitecturas para objetos distribuidos. Dos de las más conocidas y usadas actualmente son DCOM (Distributed Component Object Model) y CORBA (Common Object Request Broker Architecture) [AKP98]. También vamos a introducir en la comparación a Java RMI (Java Remote Method Invocation) que es el estándar para la invocación de Objetos Java desde JDK 1.1 [SEG98].

Cada una de estas arquitecturas se utiliza para diseñar, desarrollar y mantener sistemas distribuidos a gran escala. Además todas ellas utilizan el concepto de cliente – servidor tanto para las comunicaciones como para los otros tipos de servicios que se brindan en este esquema.

Vamos a analizar las características de cada una como entorno de programación y marco para el desarrollo de sistemas en Internet. Definiremos las principales fortalezas y capacidades de las arquitecturas como así también las debilidades y funcionalidades faltantes.

Entre los lenguajes que actualmente se utilizan en Internet y en aplicaciones para redes de área amplia vamos a realizar una comparación entre C++ y Java. Se expondrán las principales características de cada uno y las conclusiones sobre sus capacidades para desarrollos de este tipo.

Capítulo 2

Sistema a Implementar (ODS)

Se realiza una breve descripción del Sistema de Distribución de Objetos (ODS) [RNL 97], con sus principales características, y se presenta el modelo de computación distribuido que en él se propone como una alternativa para implementar la distribución de información en una red de área amplia.

Se incluye una somera descripción de sus componentes principales, las funciones necesarias para su desarrollo y el modelo de ruteo. Esto es presentado como marco para el análisis de diferentes arquitecturas y lenguajes actuales de programación y desarrollo en Internet.

El modelo presentado está pensado específicamente para cubrir las necesidades de un sistema que no precisa mantener en forma on-line, pero sí con un alto grado de accesibilidad y en forma transparente a los usuarios finales, objetos sin un alto índice de cambios en sus contenidos. Los objetos se acceden en forma local y son actualizados en forma off-line.



La posibilidad de clasificar los objetos hace que el sistema de distribución sea a la vez de flexible y óptimo en cuanto al uso de los recursos para ordenar el acceso a los datos en Internet, cosa que contrasta con la mayoría de los casos actuales en la red.

Capítulo 3

Necesidades del Sistema

Se presenta un análisis de las necesidades en cuanto a funcionalidad, seguridad, flexibilidad, comunicaciones y otros aspectos, tanto del lenguaje como de la base de datos que fueron identificados para la implementación de sistemas en gran escala, y en particular para ODS, sobre redes de área amplia y más específicamente sobre Internet.

El análisis indica que la base de datos debe manejar sentencias SQL (Structured Query Language) por ser este el standard utilizado en la mayoría de las aplicaciones actuales. Debe ser independiente del lenguaje a utilizar y poseer la suficiente capacidad para soportar grandes volúmenes de datos como así también esquemas de seguridad y auditoría, que permitan el control y seguimiento durante su utilización.

Es deseable que el lenguaje que se vaya a utilizar para la implementación, pueda manejar diferentes tipos de conexiones, tales como punto a punto, datagramas, etc. Es importante además, una eficiente detección de los errores producidos en la comunicación, para poder tomar las medidas adecuadas cuando se detectan las anomalías.

Entre los puntos más importantes del lenguaje que se analiza, están los referidos a Portabilidad y Mutiplataforma como parte de la flexibilidad necesaria de los sistemas para los distintos ambientes existentes en las grandes redes. Además, por el entorno donde se desarrolla el modelo, es necesario poner mucho énfasis en la seguridad. El lenguaje deberá tener cubierto este punto con una amplia funcionalidad para poder lograr un sistema sólido y seguro.

Otros aspectos que se analizan y que deben ser tenidos en cuenta para calificar al lenguaje de programación elegido para la implementación son: capacidad para transmitir gran volumen de información, manejo de objetos, fault tolerance y arquitectura cliente / servidor, entre otros.

Capítulo 4

Decisiones de Implementación del Prototipo

Como alternativa de implementación hemos decidido volcarnos al desarrollo de un sistema de Distribución de Documentos. Esto es un subconjunto de la especificación completa del sistema ODS [RNL 97] que limita el alcance original propuesto para objetos en general, al específico de publicaciones escritas.



Dentro de las decisiones de implementación se incluye el desarrollo del Agente de Servicio que es el componente básico para poder utilizar el sistema de distribución de documentos sobre Internet.

También vamos a implementar un Agente Usuario, que permita el acceso de los usuarios a la Red de Distribución de Documentos, que será el medio para poder colocar y consultar objetos en la misma. No era imprescindible realizar este desarrollo ya que se podría acceder al sistema a través de un navegador estándar de Internet, pero es un buen test para el lenguaje y la arquitectura elegidos, y un entorno amigable para el usuario final.

Capítulo 5

Desarrollo del Agente de Servicio

Se definen las características principales del Desarrollo del Agente de Servicio, las decisiones con respecto a la funcionalidad que se implementaron y las que debieron ser dejadas de lado.

Se destaca la modularidad desarrollada dentro del Agente de Servicio, permitiendo amplia flexibilidad y brindando la oportunidad de probar en forma independiente las distintas partes que se fueron poniendo en producción y el modelo de comunicaciones asignado en cada caso.

Además de las principales características y funcionalidad de cada módulo, se describe en detalle el camino o circuito que los documentos del sistema deben recorrer, tanto para ser ingresados como para ser extraídos, y las diferentes etapas que atraviesan dentro del Agente de Servicio.

Se analiza y especifica el protocolo de comunicaciones que se utilizará entre el Agente de Servicio y el Agente Usuario. Dicho protocolo está implementado en forma totalmente transparente y abierta, de modo que cualquier cliente que cumpla dicho protocolo puede comunicarse e interactuar con el sistema sin necesidad de efectuar cambios en el mismo.

Capítulo 6

Desarrollo del Agente de Usuario

El Agente de Usuario es la interfase que le permite al usuario realizar operaciones sobre documentos: introducción de nuevos documentos, modificaciones, consultas, control de versiones, etc. Se encarga de comunicarse con el Agente de Servicio en forma transparente para el usuario, es decir, el mismo no necesita conocer donde reside el Agente de Servicio, logrando así transparencia de localización.



Se describe la funcionalidad del Agente de Usuario y el tipo de interfase que se desarrolló para hacer de la misma un módulo que ayude al usuario en el uso del sistema. Entre sus características principales podemos destacar las interfases gráficas, el control automático de conexión con el Agente de Servicio, capacidad de ser ejecutado tanto en Windows 95 como Windows NT, configuración flexible de parámetros, y otros aspectos que hacen a la utilización en forma eficaz del Agente de Usuario.

Se especifican también las principales características técnicas del Agente de Usuario, tales como el tipo de comunicación que utiliza y los mecanismos de control. Funciona como cliente en un sistema Cliente-Servidor, envía requerimientos al Servidor (Agente de Servicio), éste recibe el mensaje y lo procesa (en general son sentencias SQL – Structured Query Language-) y devuelve el resultado de la operación al Cliente (Agente de Usuario). Entre el cliente y el servidor se establece una comunicación punto a punto.

Conclusiones

Las conclusiones principales derivadas de la investigación y del trabajo realizado, muestran los aspectos más importante respecto a la investigación de las diferentes arquitecturas, como así también en lo relativo a las cualidades de Java para este tipo de desarrollo.

Se vuelcan en esta parte las conclusiones finales de los autores del trabajo sobre la experiencia de la investigación y la aplicación de las herramientas elegidas. También se incluye el grado de cumplimiento de las expectativas previas con respecto al lenguaje y al entorno de programación utilizado para el desarrollo.

También se muestran algunas alternativas para continuar con la implementación del prototipo, ya sea agregando nuevas funcionalidades o ampliando el alcance de las ya existentes.



Capítulo 1

Comparación de Arquitectura y Lenguaje

De las arquitecturas que existen hoy en día para desarrollos de aplicaciones distribuidas vamos a analizar tres de las más conocidas y usadas actualmente. DCOM (Distributed Component Object Model), CORBA (Common Object Request Broker Architecture) y Java RMI (Java Remote Method Invocation) que es el estándar para la invocación de Objetos Java desde JDK 1.1.

Vamos a analizar las características de cada una como entorno de programación y marco para el desarrollo de un sistema distribuido. Definiremos las principales fortalezas y capacidades de las arquitecturas como así también las debilidades y funcionalidad faltantes.

En general las arquitecturas tienen mas elementos en común que grandes diferencias. Todas ellas tienen orientación a objetos con la mayoría de las funciones básicas que todos los modelos de objetos poseen, estructura de cliente - servidor para la organización y distribución de los procesos y RPC para las invocaciones remotas, tanto de funciones y procedimientos como de objetos remotos.

Quizás lo que más diferencia a las arquitecturas es el área donde se pueden adoptar cada una de las aplicaciones para optimizar sus funciones. CORBA se destaca en la comunicación entre empresas, DCOM es lo mas apropiado para computadoras personales y Java RMI aparece como lo mas indicado en los desarrollos de Internet.

Arquitectura

Introducción:

Se va a introducir una breve descripción de los principales componentes que utilizan todas las arquitecturas: modelo de computación cliente – servidor, RCP, orientación a objetos, diferentes tipos de comunicaciones y framework de arquitecturas.

Esos modelos muestran la arquitectura que se necesita para construir, desarrollar, administrar y mejorar sistemas distribuidos en gran escala. Todos han adoptado un modelo de objetos para modelar, a través de hipótesis y herramientas, la implementación de los sistemas con técnicas basadas en objetos.

Vamos a analizar brevemente las características de cada uno como entorno de programación y marco para el desarrollo de un sistema en Internet como el que estamos planteando en el capítulo 2.



Cliente - Servidor

El concepto cliente – servidor es utilizado para definir la forma en que interactúan, entidades lógicamente separadas, dentro de una red para brindar una determinada funcionalidad a los usuarios.

Esta concepción de computación está incluida en lo que se llama plataformas abiertas [ORF97] y tuvo su advenimiento principal con la inclusión masiva de las PCs, dado que permitió balancear la carga de trabajo que antes estaba centralizada en los mainframes entre todos los componentes de las redes.

Prácticamente todas las organizaciones a nivel mundial están reviendo la forma de desarrollar nuevas técnicas para implementar proyectos Cliente Servidor. Parece que esta arquitectura seguirá siendo la dominante en la próxima década, teniendo en cuenta las diversas alternativas e inversiones que se manejan para su implementación y mejoras.

El éxito de los sistemas Cliente Servidor a aumentado mucho desde sus inicios a principio de la década del 90. Actualmente solo el 5% [EUR95] de los sistemas Cliente Servidor fallan en los requerimientos en cuanto a números de transacciones, accesos concurrentes de usuarios, performance de acceso, crecimiento, etc.

Hay diferentes alternativas en cuanto a la implementación y a la funcionalidad de cada parte, usualmente el servidor administra recursos en forma centralizada que brinda a los clientes a pedido de estos. Normalmente la comunicación toma la forma solicitud – respuesta, siempre iniciada por el cliente.

Los elementos básicos que existen en este tipo de sistemas son:

- **Servicio:** es la funcionalidad, tarea, proceso, datos, etc. que brinda un servidor a los clientes que acceden al mismo.
- **Servidor:** es la entidad central que abastece del servicio a los clientes que lo requieren, usualmente maneja el concepto de permisos de acceso y tipos de requerimiento. Podemos dividir a los servidores en dos categorías principales:
 - **Servidor de Aplicaciones:** es la parte que maneja la lógica del negocio que se está modelando con el sistema, usualmente los programas o aplicaciones se encuentran en estos servidores.
 - **Servidor de Base de Datos:** usualmente administra las bases de datos de un sistema a través de un software DBMS, que se encarga de centralizar y optimizar las operaciones de acceso a la base de datos por medio de los requerimientos de los clientes a través de sentencias SQL.
- **Cliente:** es quién solicita un servicio al servidor y es el consumidor de dichos servicios, usualmente es la entidad que posee la interfase con el usuario final.



Parte de su funcionalidad comprende usualmente la interfase gráfica (GUI) y otras aplicaciones de ‘front end’

- **Protocolo:** es la forma en que se comunican clientes y servidores, es imprescindible que la forma de comunicación sea preestablecida en forma común y que sea compatible entre ambas entidades para que la misma pueda llevarse a cabo.

Dificultades con el Modelo Cliente Servidor

Las aplicaciones típicas Cliente – Servidor generalmente organizan los componentes de manera que la base de datos resida en el Servidor, las interfaces de usuario en el Cliente y la lógica generalmente está distribuida en ambos con posibilidad de variar la carga de alguno [ORF97].

Se pueden identificar cuatro principales componentes que comprometen la performance de un sistema Cliente Servidor.

- Mucho tráfico en la red causado por clientes demasiado inteligentes.
- Incremento de las capas en la arquitectura que se implementa.
- Demoras en las comunicaciones entre procesos por el tráfico de la red.
- Incremento en la cantidad de diferentes arquitecturas que deben ser contempladas en una red de área amplia como Internet.

Vamos a ver a modo de ejemplo, como el incremento de las capas de la arquitectura puede modificar el comportamiento o al menos hacerlo más dificultoso en un esquema Cliente Servidor [EUR95].

El siguiente cuadro muestra el crecimiento de las capas de las que se compone un sistema Cliente Servidor y la tendencia en aumento que se ha ido experimentando con sistema cada vez más complejos.

Terminal/Host	Cliente/Servidor de 2 capas	Cliente Servidor de 3 capas
Terminal firmware	Display driver GUI Application code Toolkit Layer DB Connectivity Client O/S Network Transport	Display driver GUI Application code Toolkit Layer Middleware Client O/S Network Transport LAN/WAN Network Transport Middleware Business logic DB Connectivity Server O/S Network Transport
Terminal Driver	LAN/WAN	LAN/WAN



Forms Management s/w Application code SQL Services DBMS Host O/S	Network Transport DB Connectivity SQL Services DBMS DB Server O/S	Network Transport DB Connectivity SQL Services DBMS DB Server O/S
7 Componentes	13 Componentes	20 Componentes

Con el aumento de la tecnología han aparecido nuevos componentes en las arquitecturas que se manejan en la actualidad, sobre todo si tenemos en cuenta la implementación de estos esquemas sobre Internet, contando con accesos remotos a través de servidores WEB.

Orientación a Objetos

El paradigma de Orientación a Objetos está mucho más cerca de la forma de pensar y ver el mundo real que el paradigma tradicional de diseño y programación de sistemas tradicionales [ORG96]. En el mundo real nosotros tenemos contacto a diario con objetos que tienen propiedades y un determinado comportamiento, tal como lo plantea la teoría de objetos.

Los objetivos principales de la tecnología de objetos son:

- Los programas escritos en función de objetos deben ser más intuitivos y fáciles de manipular que los tradicionales.
- La reusabilidad de los diferentes objetos será una de las mayores ventajas de este modelo. Gran parte del código de los sistemas desarrollado bajo este paradigma puede ser utilizado en otras aplicaciones, pues se independiza el diseño de la implementación de los objetos.
- Las modificaciones y manutención de los programas con objetos pueden ser realizadas en forma modular por objetos, por consiguiente es más fácil y barato de realizar.
- El análisis de los problemas a modelizar en un sistema de computación podrá ser realizado en términos de objetos reales. Es evidente la ventaja de poseer un método conceptualmente unificado de análisis para diseño de software.

Principales Características de los Objetos

Abstracción

Este paradigma pone énfasis en una técnica que permite pensar las aplicaciones como un conjunto de objetos que interactúan entre sí para el objetivo del sistema. Esta división permite una mejor organización y diseño de los sistemas, esto se debe a la forma natural de encarar los problemas de modelado del mundo real que se presentan.



Objeto = Datos + Métodos

En la programación tradicional los datos estaban separados de los algoritmos o funciones. En un objeto se unen los dos conceptos, el objeto contiene datos (propiedades) que lo definen y además las órdenes (métodos) que se le pueden impartir para conseguir un determinado comportamiento o alterar algunas de sus propiedades. Esto asegura un acceso controlado a los datos y que el estado de los mismos sea mantenido siempre en forma consistente.

Encapsulamiento e Información oculta

Este concepto tiene que ver con la forma en que se accede al objeto y como es el comportamiento del mismo. No es necesario, ni deseable, que la forma en que almacena y modifica el objeto los datos básicos sea conocida por todos los que usan el objeto. El diseñador del objeto da acceso a los datos a través de los métodos y mantiene los derechos sobre la forma en que se implementan los mismos.

Parte de la información de los objetos se oculta al usuario final con el fin de lograr mayor nivel de abstracción y alcanzar un sistema más fácil de modificar, mantener y extender.

Del objeto se conoce QUE cosas puede hacer y que propiedades posee, pero no COMO hace las cosas y tampoco como guarda sus propiedades.

Métodos

En el mundo real los objetos hacen cosas o responden a estímulos externos. La forma de modelar este comportamiento es a través de métodos. Los métodos son funciones que pertenecen al objeto y que el usuario puede invocar para obtener del objeto una reacción, comportamiento o cambio. Los usuarios pueden acceder a los métodos invocándolos, pero no tienen acceso a la forma en que están implementados, esto da flexibilidad para modificar y mejorar los métodos de los objetos

Herencia

Es la propiedad que tienen los objetos de ser creados a partir de otros ya existentes, heredando sus propiedades y métodos. Esto permite ahorrar tiempo a diseñadores y programadores, ya que el código puede ser reutilizado.

Debemos destacar también el concepto de Clase, que abarca las características generales de diferentes objetos. A partir de una Clase, se pueden definir objetos particulares que mantienen el estado (datos internos) y el comportamiento (operaciones que puede realizar) de la Clase que le da origen, pero agregar sus características particulares.



Polimorfismo

Es la posibilidad que tienen los objetos de responder a los mismos mensajes con un diferente tipo de comportamiento, que puede deferir por ejemplo de la Clase que le dio origen. Existen también en algunos lenguajes como Java, la posibilidad de que un mismo método pueda ser invocado con diferentes tipos y cantidad de parámetros, de esta manera podemos decir que el método o el objeto puede adquirir diferentes formas (polimorfismo) en función de las necesidades del usuario.

RPC (Remote Procedure Call)

Otras de las características comunes de las arquitecturas analizadas es la implementación de métodos RPC (Remote Procedure Call). Podemos ver a RPC como un caso especial de servicio cliente – servidor, donde el cliente invoca un procedimiento o función que se encuentra físicamente ubicado en un servidor, pero cuya localización es transparente para el cliente [PAC98].

Tanto en el cliente como en el servidor existen programas especiales (stub) que se encargan de hacer transparente la ejecución de estos procedimientos remotos, tanto para quién lo invoca como para el procedimiento propiamente dicho. Los stub pasan los parámetros necesarios al procedimiento en el servidor (cuando se ejecuta el procedimiento remoto el cliente no sabe, ni necesita saber, si la invocación es local o remota) y luego devuelven los resultados al proceso que realizó la llamada RPC.

- **Stub (Cabo):** Programa encargado de implementar una llamada RPC, ocultando a quién invoca la función la información referida a la localización del servicio llamado y los detalles referidos a la red donde opera.

La mayor dificultad en la implementación de una llamada RPC es el pasaje de parámetros por referencia, en especial, punteros a listas complejas, grafos u otras estructuras de datos variables. También es difícil tratar con procedimientos y funciones que son pasadas como parámetros [BINE84].

Es muy importante el manejo de excepciones que se realiza en un esquema RPC, ya que si la llamada RPC falla, no se puede devolver el control a quién realiza la llamada sino al manejador de excepciones del lenguaje. Es necesario reconocer las llamadas exitosas de las que no concluyen en forma correcta para poder tomar los recaudos y las acciones correspondientes.

El otro aspecto que no puede ser soslayado es la semántica de la comunicación, ya que se involucra en cada llamada el proceso en el cliente y servidor, y éstas pueden caerse, pudiendo ocurrir que algunos mensajes se perdiesen, que haya dificultades con los parámetros, etc. El buen manejo de las comunicaciones evitará que un cliente quede en espera infinita o sean informadas excepciones inexistentes durante la llamada RPC [BINE84].



Framework

Los sistemas que estamos analizando presentan un *framework*, que corresponde al marco conceptual de la arquitectura o modelo donde se manejarán términos comunes para unificar el análisis.

Vamos a utilizar la siguiente terminología para unificar las referencias que se realizarán en el análisis de las diferentes arquitecturas [OGR96]:

- **Interfase:** una colección identificada por nombre de operaciones abstractas o métodos que representan una funcionalidad. Si lo relacionamos con los conceptos ya descriptos sobre objetos, podemos decir que la interfase es el diseño conceptual de un objeto, ya que se identifica una entidad y se describe su funcionalidad en ambos casos.
- **Clase:** es la implementación concreta de una interfase. Continuando con la analogía anterior podemos decir que esto es la implementación concreta de un diseño de objeto. En este caso se debe programar la funcionalidad y los datos que contendrá el objeto o clase.
- **Instancia de Objeto:** es la instanciación de alguna clase. En este caso el concepto hace referencia al uso concreto de un objeto o clase. Cuando en una aplicación se utiliza una clase esta toma un valor concreto y tiene entidad y comportamiento propios, es decir, se ha transformado en una instancia de objeto.

Sockets

Los sockets no son más que puntos o mecanismos de comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso, incluso estando estos procesos en distintas máquinas. Esta característica de interconectividad entre máquinas hace que el concepto de socket sea de gran utilidad [ORF97].

Un socket es al sistema de comunicación entre ordenadores lo que un buzón o un teléfono es al sistema de comunicación entre personas: Un punto de comunicación entre dos agentes (procesos o personas respectivamente) por el cual se puede emitir o recibir información. La forma de referenciar un socket por los procesos implicados es mediante un descriptor del mismo tipo que el utilizado para referenciar archivos. Se puede implementar la filosofía CLIENTE-SERVIDOR por medio de sockets: un proceso en esta comunicación actuará de proceso servidor creando un socket cuyo nombre conocerá el proceso cliente, el cual podrá "hablar" con el proceso servidor a través de la conexión con dicho socket [ORF97].

El tipo de sockets describe la forma en la que se transfiere información a través de ese canal de comunicación. Según el grado de confiabilidad y método por el cual se realiza la comunicación se pueden dividir en dos modelos diferentes:



TCP (Transport Control Protocol)

Son un servicio orientado a conexión donde los datos se transfieren sin encuadrarlos en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados. En el lenguaje Java este modelo se implementa con *Sockets Stream*

El protocolo de comunicaciones con streams es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones [MIC98].

UDP (User Datagram Protocol)

Es un servicio de transporte sin conexión. Es más eficiente que TCP, pero no está garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió. En el lenguaje Java este modelo se implementa con *Sockets Datagram*.

El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar algunos datos adicionales cada vez que se realiza una comunicación.

Diferencias entre UDP y TCP

Ahora se nos presenta un problema de seleccionar qué protocolo, o tipo de sockets, debemos usar, UDP o TCP. La decisión depende de la aplicación cliente/servidor que estemos escribiendo. Vamos a ver algunas diferencias entre los protocolos para ayudar en la decisión.

En UDP, cada vez que se envía un datagrama, hay que enviar también el descriptor del socket local y la dirección del socket que va a recibir el datagrama, luego éstos son más grandes que los TCP. Como el protocolo TCP está orientado a conexión, tenemos que establecer esta conexión entre los dos sockets antes que nada, lo que implica un cierto tiempo empleado en el establecimiento de la conexión, que no existe en UDP [MIC98].

En UDP hay un límite de tamaño de los datagramas, establecido en 64 kilobytes, que se pueden enviar a una localización determinada, mientras que TCP no tiene límite; una vez que se ha establecido la conexión, el par de sockets funciona como los streams: todos los datos se leen inmediatamente.



UDP es un protocolo desordenado, no garantiza que los datagramas que se hayan enviado sean recibidos en el mismo orden por el socket de recepción. Al contrario, TCP es un protocolo ordenado, garantiza que todos los paquetes que se envíen serán recibidos en el socket destino en el mismo orden en que se han enviado.

Los datagramas son bloques de información del tipo lanzar y olvidar. Para la mayoría de los programas que utilicen la red, el usar un flujo TCP en vez de un datagrama UDP es más sencillo y hay menos posibilidades de tener problemas. Sin embargo, cuando se requiere un rendimiento óptimo, y no está justificado el tiempo adicional que supone realizar la verificación de los datos, los datagramas son un mecanismo realmente útil.

En resumen, TCP parece más indicado para la implementación de servicios de red como un control remoto (rlogin, telnet) y transmisión de archivos (ftp); que necesitan transmitir datos de longitud indefinida. UDP es menos complejo y tiene una menor sobrecarga sobre la conexión; esto hace que sea el indicado en la implementación de aplicaciones cliente/servidor en sistemas distribuidos montados sobre redes de área local.



Características de las Arquitecturas

Vamos a analizar tres de las arquitecturas más conocidas y usadas actualmente para el desarrollo de sistemas distribuidos. DCOM (Distributed Component Object Model), CORBA (Common Object Request Broker Architecture) y Java RMI (Java Remote Method Invocation) que es el estándar para la invocación de Objetos Java desde JDK 1.1.

El análisis presentado se centraliza en primer término en la descripción de las características de cada una, destacando las principales fortalezas y capacidades, como así también las debilidades y funcionalidades faltantes. Luego se realiza una comparación de los principales aspectos que tienen en común, tales como: orientación a objetos, estructura de cliente - servidor, RPC, interfaces, etc. Finalmente se expresa el ambiente donde mejor se adapta cada una de las arquitecturas.

DCOM: (Distributed Component Object Model)

Framework de objetos creado por Microsoft. Es la extensión distribuida de COM, que construye objetos RPC para soportar y manejar los llamados por requerimientos remotos. Los Clientes COM acceden a un objeto a través de un puntero a una interfase y desde allí pueden acceder a los diferentes métodos del objeto como si estuvieran en un espacio propio de direcciones [MS99].

DCOM funciona como un protocolo que habilita a diferentes componentes de software a comunicarse directamente sobre una red en un modo confiable, seguro y eficiente. Originalmente fue llamado OLE, y está diseñado para trabajar a través de múltiples plataformas de redes, incluyendo por supuesto Internet. Las especificaciones de DCOM indican que estará preparada para trabajar con componentes de dos diseños diferentes, tanto Java como ActiveX.

La comunicación es del tipo cliente – servidor, a través de un requerimiento el cliente invoca un método implementado en un objeto remoto, que actúa como el servidor. El servicio provisto es encapsulado como un objeto y la interfase del objeto es descripta en un lenguaje especial, el IDL (Interfase Definition Language). IDL sirve para ocultar a los clientes la implementación de los objetos, pero provee la encapsulación, polimorfismos y herencia simple. DCOM no soporta herencia múltiple, en su lugar utiliza la noción de múltiples interfaces, lo que en la práctica representa diferentes vistas o comportamientos del mismo objeto [DCP99].

La arquitectura DCOM provee el diseño para la activación de objeto y otros métodos en tres capas de la red. La capa superior llamada básica es la visible para desarrolladores de clientes y programas objetos en el servidor. La capa media es la de arquitectura remota que hace transparente los punteros a las interfaces y las referencias a los objetos entre los procesos. La capa mas baja es referida al protocolo, y se espera que sirva para extender la arquitectura remota para que funcione en diferentes tipos de equipos de computación.



Objetos Locales y Remotos

En el mundo de DCOM todos los objetos son creados con un identificador único y global, compuesto de un entero de 128bits. Cada vez que un usuario quiere hacer referencia a un objeto debe referirse al mismo con este número de identificación. Este mecanismo provee un método libre de colisiones para el manejo de los objetos que se crean en una aplicación simple o en múltiples aplicaciones interactuando [DCP99].

Las librerías DCOM permiten la creación de objetos en otras máquinas. Para que esto sea posible en torno a objetos remotos, DCOM necesita conocer el nombre o la referencia del servidor de red donde solicitar el servicio. Existen dos mecanismos fundamentales para este fin, el primero es una configuración fija dada por el cliente en su sistema de registry, el otro corresponde a parámetros explícitos incluidos en la llamada al servicio.

El primer mecanismo es muy usado para mantener la ubicación en forma transparente para los clientes/usuarios del servicio que no necesitan conocer si el componente que está invocando corre en forma remota o localmente. Además las librerías DCOM se encargan de mantener actualizada la información correspondiente al nombre del servidor en la red.

Nombres para Instancias de Objetos

Existen en los esquemas distribuidos diferentes mecanismos para referenciar a las instancias de los objetos: por medio del nombre de la base de datos y del servidor, a través de una dirección URL o de una página HTML, etc.

DCOM trata de utilizar una forma flexible y extensible de hacer referencias a objetos [MIC98]. Para lograr flexibilidad, el sistema de nombres en sí mismo se compone de instancias llamadas *monikers*. La forma de nombrar los objetos contiene la lógica necesaria para encontrar una instancia de cualquier objeto, esté corriendo actualmente o no. Se define una interfase estándar que une el objeto nombre con la instancia del objeto que se desea referenciar, teniendo en cuenta que el objeto nombre está oculto para el cliente.

Usualmente las instancias de los objetos crean sus objetos *moniker* que definen sus nombres y los dejan disponibles para cualquier cliente que necesite utilizarlos. El sistema mantiene una tabla con las instancias activas de los objetos (ROT - Running Object Table)

Como ejemplo para control y referencia de los mismos podemos citar al URL *moniker* que encapsula varios protocolos estándar de Internet, incluyendo HTTP, FTP, Gopher y otros similares. Microsoft Internet Explorer utiliza este método para encapsular varios tipos de datos de Internet como GIF o HTML.



Llamadas a Métodos Remotos

Cuando un cliente llama un objeto en otro espacio de direcciones, los parámetros para el método llamado deben ser pasados de algún modo desde el cliente. Cuando el proceso comparte el mismo espacio de direcciones esto no es problema porque los parámetros pueden ser dejados en cualquier lugar común para que el proceso invocado los pueda levantar.

En una llamada remota, la librería DCOM se encarga de leer los parámetros dejados por el proceso local que realiza la llamada y enviarlos a través de la red, luego toda la estructura de parámetros, que puede ser muy compleja (ver RPC), debe ser reproducida en el espacio de direcciones del proceso llamado [MIC98].

Para resolver este problema en forma eficiente, DCOM necesita conocer exactamente el método de registro de los parámetros, incluyendo todos los tipos de datos, tipos de estructura, componentes y tamaños de cada array en la lista de parámetros. Para asegurar esto se utiliza una interfase de definición de lenguaje (IDL) que corre sobre el procedimiento RPC para asegurar una correcta implementación del mismo.

Manejo de Conexiones

- **Garbage Collection:** por medio de una interfase, DCOM mantiene una referencia remota de los procedimientos [MS99] que están siendo utilizados bajo RPC. Cuando un cliente hace un pedido de Release (eliminar el objeto) este no se transmite inmediatamente, sino que se espera hasta que todos los clientes que referenciaban a ese proceso hayan dejado de utilizarlo para darlo de baja. Esto optimiza el tráfico en la red y la performance.
- **Pinging:** como no siempre los procedimientos terminan en forma normal, se utiliza este mecanismo para detectar la forma en que los clientes terminaron su ejecución. En cada servidor el cliente tiene un tiempo de *ping*, [MS99] si pasa ese tiempo y no se recibe la referencia del cliente, se considera que el tiempo ha expirado y el procedimiento queda disponible para ser eliminado por el método de garbage collection.
- **Puntos de Conexión:** es común que las aplicaciones actuales necesiten una conexión bidireccional en tiempo real [MS99] entre clientes y objetos. En DCOM se provee un mecanismo para hacer eficiente esta comunicación, a través de punteros de interfaces pasados por el servidor se pueden enviar notificaciones [MS99] o datos entre las dos partes intervinientes. Este método puede ser muy útil para aplicaciones distribuidas donde múltiples clientes se registran en el mismo punto de conexión para recibir notificaciones en forma conjunta.
- **Referrals:** otro de los métodos muy usados para la comunicación entre procesos distribuidos. Cuando un cliente llama un objeto y este llama a un



segundo objeto, al cliente que solicita el servicio se le pasa la referencia directa al segundo objeto. Esto se utiliza para evitar el tráfico innecesario en la red y aún mas, suponiendo que el primer objeto desapareciera, todavía se podrían utilizar los servicios del segundo llamándolo directamente [MS99].

Manejo de Concurrency

DCOM no utiliza threading en el sentido estricto, sino que aprovecha la funcionalidad que los sistemas operativos nativos le brindan para esta funcionalidad. Por ejemplo, win32 provee un espacio memoria compartida para multi thread sincronizado por semáforos.

El modelo de programación de DCOM provee de total libertad para la invocación de métodos thread, tal como lo hace RPC, pero además provee de un mecanismo automático de invocación para thread simples sincronizados. Este método es llamado Single Threaded Apartments (STA) [MS99].

Seguridad en Aplicaciones Distribuidas

DCOM provee un framework extensible o customizable [DCP99] sobre el cual los desarrolladores pueden diseñar las aplicaciones, soporta la interoperabilidad de múltiples soluciones de diferentes proveedores en forma simultánea.

Maneja un concepto de identificación de seguridad principal (cuentas de usuarios), autenticación de seguridad principal (a través de passwords) y una autoridad central que maneja la seguridad principal y sus claves

Se distinguen cuatro aspectos fundamentales en lo que hace a seguridad:

- **Seguridad de acceso:** indica que cuentas de usuarios pueden hacer llamadas a los objetos del sistema. Es el requerimiento mínimo para evitar accesos indebidos, y puede ser configurado a nivel de las aplicaciones, evitando que los nuevos componentes tengan que preocuparse por este control [DC99]. De todos modos, los objetos tienen la libertad de hacer arbitrariamente las validaciones que consideren necesarias.
- **Seguridad para correr procesos:** indica que cuentas de usuarios pueden crear nuevos objetos y nuevos procesos. Como en el ambiente DCOM todos los objetos son potencialmente accesibles, es crítico prevenir que usuarios no autorizados creen instancias de objetos sobre los que no tienen permisos [DC99]. El control puede realizarse sin tener que programar explícitamente nada en el objeto mismo, el servidor puede realizar este control.
- **Identidad:** indica cual es la seguridad de los objetos [DC99] en sí mismos. Como cada objeto puede realizar operaciones en nombre de sus llamadores es necesario establecer cuales serán las capacidades y permisos del objeto



mismo. Los permisos se refieren a acceso a archivos, a la red, a determinados registros, etc.

- **Política de Conexión:** hace referencia a la integridad, privacidad y autenticación [DC99] de los mensajes y objetos. La integridad física es usualmente garantizada por la capa de transporte de la red y DCOM permite a las aplicaciones elegir cada cuanto tiempo se realiza el control lógico de la comunicación. Como privacidad e integridad requieren de autenticación, se unifican en un mismo nivel.

Objeto RPC

El protocolo DCOM reconoce los objetos RPC como un conjunto de definiciones que extienden el protocolo RPC estándar [DC99]. Fue diseñado específicamente para el entorno orientado a objetos de DCOM y especifica como las llamadas deben ser realizadas a través de la red y como las referencias a los objetos deben ser presentadas y mantenidas.



CORBA: (Common Object Request Broker Architecture)

Manejada por un grupo llamado OMG (Object Management Group) sin fines de lucro compuesto por mas de 600 empresas, CORBA fue pensada como un standard que permita interoperabilidad y portabilidad de software orientados a objetos entre distintos ambientes de computación [OMG95A].

CORBA es una arquitectura diseñada para sistemas distribuidos de computadoras que manejan tecnología de objetos. Se compone de especificaciones que modelan la comunicación y funcionalidad entre los componentes de un sistema de objetos, para este fin se define una semántica común como base consistente de entendimiento y terminología.

El Modelo CORBA

CORBA permite a las aplicaciones comunicarse entre sí sin preocuparse por saber donde están localizadas o quién las ha diseñado. El cliente que necesita invocar un objeto no necesita saber dónde está ubicado, ni el lenguaje de programación, ni el sistema operativo o cualquier otro aspecto del sistema que no fuera parte de la interfase del objeto [OMG95A].

En el campo de las aplicaciones Cliente Servidor, el diseñador puede utilizar su propio diseño para definir el protocolo a utilizar entre los dispositivos, el protocolo tiene que ver con el lenguaje, el tipo de la red y otros factores. Sin embargo, en CORBA el protocolo se define a través de interfases simples utilizando un lenguaje común (IDL) y esto permite que tanto el cliente como el servidor puedan estar programados en cualquier lenguaje y aún así puedan operar ya que tienen algo común para comunicarse: las interfaces y el lenguaje en el cual se definen [OMG95A].

Para que los objetos puedan ser utilizados en forma inmediata y transparente, los clientes deben conocer exactamente como llamar a cada objeto y que respuesta van a obtener. Para especificar estos servicios y al objeto mismo, se utiliza un *contrato* que se conoce a través de su interfase con su sistema.

El contrato tiene dos propósitos principales: el primero es informar los servicios del objeto y la forma de invocarlos, el segundo es permitir una estructura de comunicación conocida con el formato de los mensajes que el objeto puede enviar o recibir.

Cada objeto en CORBA necesita un *handle* único para ser identificado a través de la red y por los clientes que lo referencian [OMG95A]. El handle se mantiene a través del tiempo de vida del objeto y durante su transporte sobre la red, esto permite mantener el control y la consistencia del sistema.



Arquitectura del Modelo de Referencia OMA

ORB constituye la base para OMA (Object Management Architecture) que es una arquitectura de alto nivel para ambientes distribuidos. Está compuesto de dos partes principales: componentes orientados a sistemas y componentes orientados a aplicaciones.

ORB cumple la tarea de comunicación entre los componentes de ambientes distribuidos en diferentes plataformas. Para este fin existen diferentes servicios de ORB:

- **Servicios de Objetos:** son interfases de uso estándar que usan la mayoría de los programas de objetos distribuidos, por ejemplo el Servicio de Nombres que se utiliza para localizar objetos a partir del nombre [OMG95].
- **Common Facilities:** son interfaces horizontales orientadas a aplicaciones de usuario final. Por ejemplo, un administrador de un sistema distribuido que permite el intercambio de programas y control de versiones utilizando un modelo común.
- **Interfaces de Dominios:** cumplen el papel del Servicio de Objetos y Common Facilities pero orientados a un dominio específico de aplicaciones, como por ejemplo telecomunicaciones, medicina, universidades, finanzas, etc.
- **Interfaces de Aplicación:** están desarrolladas para aplicaciones existentes, como CORBA no las desarrolla, puede llegar a estandarizarlas a través de una especificación para ese dominio en particular.

Componentes de CORBA

- **Objeto:** está definido por las interfaces y los servicios que representan las características y operaciones que el objeto puede realizar, incluyendo los parámetros de entrada y salida. Para generar un objeto es necesario escribir y compilar una especificación en el lenguaje que brinda CORBA, el IDL (Interfase Definition Language). Esto permite que objetos generados en diferentes lenguajes y sistemas operativos puedan comunicarse a través de los servicios escritos en IDL sobre la arquitectura CORBA [OMG95B]. Esto brinda flexibilidad para cambiar la implementación de los objetos y manejarlos en forma transparente para los usuarios de los mismos.
- **IDL (Interfase Definition Language):** sirve para describir las interfaces de objetos, las operaciones que el objeto ejecutará, los parámetros y los valores que puede devolver como así también las excepciones que pueda generar. No es un nuevo lenguaje de programación, se utiliza para especificar tipos de las interfases, por este motivo es fuertemente tipado, luego un compilador CORBA IDL debe traducir estas especificaciones en lenguaje destino, es



decir, mapear los tipos de datos y su función al lenguaje de implementación [OMG95A].

- **ORB (Object Request Broker):** es un middleware que establece relaciones cliente/servidor entre objetos. Sirve para hacer transparente la ubicación del servidor a un cliente, cada vez que un cliente hace una llamada, ésta es capturada por el ORB, que también es responsable de encontrar la implementación pedida, traduce y envía los parámetros al servidor invocando el método. Debe también, por la misma vía, devolver el resultado al cliente que realiza la llamada [OMG95B].
- Con esta funcionalidad ORB, se provee la interoperabilidad entre diferentes aplicaciones en diferentes sistemas operativos con transparencia de comunicaciones. Cada implementador puede desarrollar los objetos en el lenguaje que prefiera y solamente debe diseñar las interfaces en IDL para que sus objetos puedan ser compartidos.

Elementos de Arquitectura CORBA

- **Implementación de Objetos:** define las operaciones que implementan una interfase CORBA IDL. La implementación puede ser escrita en una variedad muy extensa de lenguajes incluyendo C, C++, Java, Smalltalk y Ada [AKP98].
- **Clientes:** es un programa que invoca una operación de una implementación de objeto. Si el objeto es remoto o local deberá ser transparente para quién está llamando y no deberá diferir la forma de invocación del objeto en uno u otro caso.
- **Object Request Broker (ORB):** provee un mecanismo transparente para comunicar requerimientos de clientes con implementaciones de objetos. Simplifica el mecanismo de comunicación evitando que el cliente tenga que ocuparse de los detalles de invocación.
- **Interfaces ORB:** es una entidad lógica que puede ser implementada de varias formas, por medio de un proceso o a través de un conjunto de librerías. CORBA define una interfase estándar para ORB con el objeto de simplificar su uso.
- **CORBA IDL stubs y skeletons:** tanto stubs como skeletons son el pegamento que une al cliente con el servidor respectivamente. Las especificaciones son traducidas por medio de un compilador que reduce el riesgo de potenciales inconsistencias entre ambos.
- **Dynamic Invocation Interfase (DII):** esta interfase permite a un cliente acceder directamente al mecanismo de pedidos provistos por ORB. Es una alternativa a la implementación del requerimiento a través de interfaces IDL,



a diferencia de IDL, permite tratar por separado las operaciones de envío y recepción de mensajes como así también, establecer las conexiones en una sola dirección [OMB95B].

- **Dynamic Skeleton Interface (DSI):** es la parte análoga a la interfase DII de los clientes, da flexibilidad en cuanto a los tipos de objetos. Permite que en ORB un cliente pueda hacer un requerimiento a un objeto sin saber si la implementación del mismo está utilizando un tipo específico de IDL o usa Skeleton Dynamic [OMG95B].
- **Object Adapter:** asiste a ORB para enviar los requerimientos al objeto y activarlo. Se utiliza para asociar implementaciones de objetos con el modelo ORB. Pueden ser especializados en dar soporte para ciertos estilos de implementación de objetos tales como librerías de objetos no remotos [OMG95B].

Estructura de un Cliente

Cada cliente de un objeto tiene una referencia al objeto que es un *token* que puede ser invocado o pasado como parámetro [Vinoski]. La invocación de un objeto envuelve la especificación del objeto, la operación a ejecutar y los parámetros dados para la operación o para retornar los resultados. Usualmente a través de ORB se realiza la llamada que solamente retorna al cliente cuando se completa la operación y si ésta falla una excepción es la respuesta al cliente.

El cliente tiene acceso a los stubs como librerías comunes, es decir, que aparece un conjunto de rutinas que pueden ser llamadas como si fueran locales y se realizan en el lenguaje de programación del cliente.

Las referencias a los objetos se obtienen usualmente como parámetros provenientes de invocaciones a otros objetos conocidos. Cuando el cliente también es una implementación, recibe las referencias como parámetros de entrada o de invocación, los cuales pueden ser convertidos a strings almacenados para su posterior uso en un archivo o se pueden comunicar a otros objetos.

Estructura de la Implementación de un Objeto

La implementación de objetos provee el estado actual y el comportamiento del mismo. La implementación puede estructurarse en diferentes formas, ya que además de definir métodos para las operaciones, también se especifican los procedimientos para activar y desactivar objetos, como así también la persistencia o no del objeto [Vinoski].

Durante la implementación se interactúa con ORB para establecer la identidad del objeto, crear nuevos objetos y obtener servicios dependientes de ORB. Para cada nuevo objeto que se crea, ORB debe ser notificado para que se pueda referenciar la implementación a los clientes, pero además la implementación también se registra con



una particular interfase y especifica como inicializar la implementación si aún no está activa.

Estructura de un Object Adapter

El Object Adapter en el medio primario que tiene la implementación de un objeto para acceder a los servicios de ORB. Se exporta una interfase pública de la implementación y una privada para el skeleton [Vinoski]. El Object Adapter es responsable por las siguientes funciones:

- Generación e interpretación de referencias de objetos.
- Invocación de métodos.
- Seguridad de interacciones.
- Activación y desactivación de objetos.
- Mapear las referencias de objetos con la correspondiente implementación.
- Registración e implementación. Para esto se utilizan componentes de ORB.

Integración de Objetos Foráneos

Todos los objetos de otros sistemas externos a ORB deben tener la posibilidad de ser accedidos. Para los sistemas que simplemente necesitan mapear sus objetos y recibir invocaciones a través de ORB, pueden ser implementados como objetos ORB. Manejando los requerimientos que recibe y actuando como cliente para los requerimientos que provienen de su sistema.

En otros casos un Object Adapter podría ser creado para los objetos foráneos en conjunto con ORB. Otra alternativa es que el sistema externo desee crear sus objetos sin consultar a ORB y en este caso los objetos pueden registrarse ellos mismos cuando pasan a través de ORB y no antes.



Java RMI (Java Remote Method Invocation)

Para la comunicación básica, Java posee manejo de los elementos básicos como son los sockets, los cuales son flexibles y poseen suficiente funcionalidad para las comunicaciones generales. Pero las aplicaciones requieren que se programe, tanto el servidor como el cliente, para codificar y decodificar los mensajes [JAV99].

La posibilidad de utilizar RPC como alternativa para la invocación de procedimientos remotos como si estuvieran en el ambiente propio, necesita de RMI para coordinar y matchear la semántica de invocación de objetos.

Objetivos de RMI

Los principales objetivos de soporte para objetos distribuidos en lenguaje Java son:

- Soportar la invocación en forma transparente de objetos en diferentes máquinas virtuales.
- Soportar respuestas de llamados por parte de los servidores hacia los Applets.
- Integrar el modelo de objetos distribuido dentro del lenguaje Java en forma natural, manteniendo la semántica de objetos del lenguaje.
- Hacer transparente la diferencia entre un modelo distribuido de objetos Java y un modelo de objetos local.
- Poder generar aplicaciones distribuidas confiables y en la forma tan simple como fuera posible.
- Preservar el ambiente de seguridad provisto por el entorno de ejecución de Java.

Modelo Distribuido de Objetos Java

En este modelo un objeto remoto es aquel cuyos métodos pueden ser invocados desde otra máquina virtual Java, potencialmente en un host diferente. Un objeto de este tipo es descrito por una o más interfaces remotas que son una extensión de las interfaces de Java [JAV99].

- **Remote method invocation (RMI):** es la acción por la que se invoca un método de una interfase remota en un objeto remoto, pero lo más importante es que la llamada se realiza de la misma forma que si el objeto fuera local.

Diferencias entre Modelo Distribuido y no Distribuido

Los dos modelos son similares en los siguientes aspectos:



- La referencia a un objeto remoto puede ser pasado como argumento o retornada como resultado de otra invocación a un método ya sea local o remoto.
- Un objeto remoto puede ser asignado a cualquier otro conjunto de interfaces remotas soportadas por la sintaxis de Java.
- Java tiene un operador que puede ser utilizado como test para probar las interfaces remotas soportadas por un objeto remoto.

Los dos modelos son diferentes en los siguientes puntos:

- Los clientes de objetos remotos interactúan con las interfaces remotas, nunca con la implementación de dichas interfaces.
- Ni los parámetros pasados al objeto remoto ni los resultados obtenidos del mismo pueden ser pasados por referencia. Solamente es usado el método de referencia cuando se trabaja en una única máquina virtual Java.
- Los objetos tienen parte de la semántica que es exclusiva de los métodos para dichos objetos.
- Los clientes deben considerar que la invocación a objetos remotos tiene mas posibilidad de fallar, por lo que deben estar preparados para manejar las excepciones que se puedan producir.

Componentes

El sistema RMI consta de tres capas: stub/skeleton, referencias remotas y transporte. El alcance de cada capa está definido por interfaces y protocolos específicos, y cada una es independiente de las otras y puede ser alterada su implementación sin afectar a las demás [PAC98].

La capa Stub/Skeleton

Esta capa es la interfase entre la capa de aplicación y el resto del sistema RMI. Esta capa no interactúa con ninguna capa de transporte pero transmite datos a la capa de referencia remota a través de una abstracción de *marshal streams*. Para este fin utiliza el mecanismo de serialización provisto por Java para que los objetos puedan ser transmitidos.

El *stub* es quién actúa del lado del cliente como responsable del objeto remoto. Como implementa las interfaces soportadas por el objeto remoto tiene las siguientes responsabilidades:

- Inicializar una llamada al objeto remoto, para esto debe hacer una llamada a la capa de referencia remota.
- Preparar los argumentos para formar el *marshal stream*, para que puedan ser transmitidos.
- Informar a la capa de referencia remota para que realice la invocación.



- Preparar los argumentos que retornan como resultado de la llamada para que el cliente pueda accederlos.
- Avisar a la capa de referencia remota que el llamado se completó.

El *skeleton* es quién actúa del lado del servidor para la llamada a un objeto remoto. Es responsable por las siguientes tareas:

- Desempaquetar los argumentos que provienen del *marshal stream*.
- Hacer la llamada a la implementación actual del objeto en el servidor.
- Preparar los datos resultados utilizando *marshal stream* o una excepción en caso de ocurrir un error.

La capa de Referencia Remota

Esta capa se encarga de interactuar con el nivel mas bajo de la interfase de transporte. También es responsable de mantener un protocolo específico de referencias remotas que es independiente de stubs y skeletons [AKP98].

Cada implementación de un objeto remoto elige su propia subclase de referencia que maneja su comportamiento. Varias invocaciones a protocolos son atendidas en esta capa:

- Invocaciones point to point.
- Replicaciones para grupos de objetos.
- Soporte para estrategias específicas de replicación.
- Soporte para una referencia permanente a objetos remotos
- Estrategias de reconexión si los objetos remotos se vuelven inaccesibles.

Esta capa tiene dos componentes que cooperan entre sí. La parte del cliente y la parte del servidor que durante cada invocación se encargan de la semántica específica para la referencia remota. Los datos son transmitidos a la capa de transporte por medio de una abstracción de un *stream* orientada a conexión. El transporte se encarga de los detalles que aseguran la implementación segura de la conexión.

La capa de Transporte

Esta capa es responsable de las siguientes tareas:

- Inicializar la conexión con el espacio de direcciones remoto.
- Manejar y administrar la conexión.
- Realizar el control de que la conexión se mantenga viva.
- Escuchar por llamadas entrantes.
- Mantener una tabla de objetos remotos que reside en el espacio de direcciones.



- Localizar a quién realiza el pedido remoto y pasar la conexión a quién lo debe recibir.

La representación de la referencia remota consiste en un puntero denominado *endpoint* y un identificador, la capa de transporte puede usar el puntero para establecer una conexión con el espacio de direcciones donde reside el objeto. En el servidor la capa de transporte utiliza el identificador del objeto para localizarlo.

Abstracciones básicas de la capa de Transporte:

- El *endpoint* es la abstracción para denotar un espacio de direcciones o una máquina virtual Java. En la implementación los *endpoint* pueden ser mapeados en la capa del transporte.
- El *canal* es la abstracción para conectar dos espacios de direcciones. Se utiliza para el manejo de la conexión entre el espacio de direcciones locales y el espacio de direcciones remoto.
- La *conexión* es la abstracción para transferencia de datos (input y output)
- El *transporte* es la abstracción para manejar los canales. Cada canal es una conexión virtual entre espacios de direcciones y dentro de la abstracción de *transporte* solo puede existir un canal para cada par de direcciones. Esta abstracción establece el canal entre un *endpoint* dado y la dirección remota. También recibe llamadas provenientes del espacio de direcciones y las direcciona para las capas superiores del sistema.

Garbage Collection de Objetos Remotos

Dado un sistema distribuido, en cada uno de los sistemas locales es deseable que los objetos remotos que ya no son referenciados por ninguno de los clientes sean borrados en forma automática. Esto libera a los programadores de mantener la referencia de los clientes de objetos remotos [JAV99].

Para implementar el *garbage collection*, el runtime de RMI mantiene el control de todas las referencias a objetos dentro de cada máquina virtual Java. Se incrementa el contador de referencias cada vez que un nuevo requerimiento ingresa a la máquina virtual Java y cuando se verifica un mensaje que libera una referencia se decrementa el contador de dicho objeto. Cuando la última referencia ha sido descartada se envía un mensaje al servidor donde reside el objeto para que sea eliminado. También existe un mecanismo de control para asegurar que los objetos no sean borrados en forma prematura.

Cuando un objeto remoto no es referenciado por ningún cliente, el runtime de RMI utiliza un tipo de referencia especial para permitir que la máquina virtual Java realice el *garbage collection* si no existe otra máquina virtual Java que lo utiliza.



Hay que notar que si existe una partición de la red entre el cliente y el servidor remoto, puede ser posible que se borre en forma prematura algún objeto. Esto se debe a que la capa de transporte puede creer que el cliente ya no existe. Por este motivo, las referencias remotas no pueden garantizar integridad de referencia, es decir, que una referencia remota puede de hecho no estar apuntando a ningún objeto existente.

Carga de Clases Dinámicas

En un sistema RPC (remote procedure call) el código del stub correspondiente a la parte del cliente debe ser generado y conectado al cliente antes de que la llamada remota pueda ser hecha. Este código puede ser conectado en forma estática o en forma dinámica por medio de librerías locales o sobre un sistema de archivos de red. RMI generaliza esta técnica usando un mecanismo llamado *dynamic class loading* que se encarga de cargar las clases que manejan RPC [JAV99].

Seguridad

En Java, cuando una clase es cargada desde las librerías locales, estas clases son consideradas confiables y no tienen restricciones por parte del manager de seguridad. Pero cuando la clase se carga desde la red, el manager de seguridad debe controlar la clase por si se produce una excepción [JAV99].

El manager de seguridad debe comenzar con la primera acción del programa Java, para que pueda regular las siguientes acciones. Las aplicaciones deben definir su propia seguridad o utilizar el manager estándar.

Applets

Típicamente, las clases necesitan ser provistas por un server HTTP o por un server FTP haciendo referencia a direcciones URL dentro de páginas HTML conteniendo los applets. El servicio RMI utilizado por el applet debe ser el server de donde se bajó dicho applet, ya que los applets solamente pueden hacer conexiones con el host desde el cual ha sido bajado [JAV99].

Por ejemplo, un escenario normal para un applet está determinado por el uso de un host para server HTTP que provee la página HTML, el código del applet, los servicios RMI y el registro de inicio. Todas las clases son pasadas al applet desde el servicio RMI con lo que se convierte en una aplicación segura ya que no toma clases desde la red.

Aplicaciones

Las aplicaciones escritas en lenguaje Java, a diferencia de los applets, pueden conectarse a cualquiera host, de esta forma tienen mas alternativas para configurar el



origen de las clases. Típicamente un server HTTP será usado para proveer clases remotas, mientras que las aplicaciones basadas en RMI están ellas mismas distribuidas en la red, en servidores o corriendo sobre las máquinas de los usuarios.

Si una aplicación es cargada localmente, entonces las clases usadas directamente en dicho programa deben ser también disponibles localmente. En este escenario las únicas clases que pueden ser bajadas desde la red son las clases de interfaces remotas, clases stub, las extensiones de clases y los valores que retornan de invocaciones a métodos remotos [JAV99].

Para habilitar que desde la red se baje el código para una aplicación, cada objeto remoto debe ser configurado con una propiedad especial de Java que especifica donde residen las clases de la aplicación y los stubs/skeletons generados.

El RMI de Java está pensado para operar en el entorno de Java y asume la homogeneidad de la Máquina Virtual Java y puede aprovechar todas las ventajas de los objetos Java y su funcionalidad.

Cuando un programa Java realiza una llamada a un objeto remoto, puede obtener la referencia al objeto a través de dos formas diferentes: por un servicio de “naming” provisto por RMI o si recibe la referencia como argumento o retorno de una llamada a un procedimiento. Cuando un cliente realiza una llamada remota en un servidor, puede ocurrir que este servidor sea cliente de otro servidor para satisfacer el requerimiento. RMI utiliza “serialización” como método propio para organizar el pasaje de parámetros y estandarizar tipos y objetos soportando polimorfismo. Otra de las técnicas del sistema es la carga dinámica de stubs que es usada en la implementación de las interfases del lado del cliente.

Principales Ventajas de Java RMI

- **Orientado a Objetos:** RMI puede pasar objetos completos como argumento y retorno de valores, no solo de tipos predefinidos. Esta característica, contrariamente a la mayoría de los sistemas RPC actuales, permite enviar objetos a través de la red sin tener código extra en el cliente.
- **Comportamiento flexible:** en RMI se pueden mover las implementaciones de clases desde el cliente hasta el servidor y viceversa. Por ejemplo, cuando se produce un cambio en la política de seguridad de un sistema, la restricción puede ser chequeada en el host en la parte del cliente proveyendo un rápido feedback a los usuarios sin recargar al servidor. Esto brinda la flexibilidad de poder hacer un cambio en el sistema de seguridad solamente escribiendo una nueva clase e instalándola en el servidor, sin modificar los sistemas de los usuarios [PAC98].
- **Patrones de Diseño:** la posibilidad de transferir objetos completos permite utilizar todas las virtudes de la tecnología en computación distribuida. Se utilizan los Patrones de Diseño para poder pasar tanto la implementación



como los tipos de objetos y así poder utilizar toda la potencia de los objetos en forma transparente [PAC98].

- **Seguridad:** RMI permite utilizar los mecanismos de seguridad que aseguran que los sistemas continúen siendo seguros cuando los usuarios bajan las implementaciones desde la red. El manager de seguridad se encarga de evitar que tanto los applets hostiles como cualquier otro código no autorizado pueda ser ejecutado.
- **Fácil de escribir y usar:** RMI hace que tanto los servidores como los clientes puedan ser escritos en Java de forma fácil. Una interfase remota es una interfase normal en Java y un servidor tiene apenas algunas líneas adicionales de código para declararse como tal, y el resto es como cualquier otro objeto Java. Esta facilidad hace que sea muy adecuado para implementar sistemas rápidamente, tales como prototipos y pruebas de versiones iniciales de software [PAC98].
- **Conexiones con los sistemas Existentes / Legacy:** Java posee métodos nativos (JNI) que a través de sus interfases pueden interactuar con dichos sistemas. Esto permite a clientes escritos en Java comunicarse con servidores ya existentes o cambiar parte de los servidores existentes implementando los cambios en código Java. La conexión con las bases de datos se realiza a través de JDBC sin necesidad de modificarlas en ningún aspecto.
- **Multiplataforma:** todo el código escrito en Java es portable entre cualquier plataforma que posea una *Java Virtual Machine*. Utilizando la combinación RMI/JDBC se puede compilar el código de las aplicaciones y ejecutarlo en diversas plataformas, sin necesidad de reescribirlo ni recompilarlo [JAV99].
- **Garbage Collection distribuido:** esta característica es análoga a la provista por el lenguaje Java a nivel local y permite que se definan tantos objetos servidores como se necesiten sabiendo que serán borrados cuando ya nadie de los clientes necesite acceder a ellos, facilitando la tarea de los programadores y diseñadores [JAV99].
- **Computación paralela:** como RMI es multi-thread, permite que los servidores de objetos puedan utilizar mejor el procesamiento concurrente de requerimientos de clientes a través de los threads de Java [JAV99].
- **Solución distribuida de Java:** como RMI es parte del core de Java, existe en todas las versiones de JDK, por lo tanto, el sistema RMI utiliza el mismo protocolo público de Java y puede comunicarse con cualquier otro sistema escrito en Java directamente y con gran eficiencia.



Tabla Comparativa

	CORBA	DCOM	RMI
Interfases	Definidas en IDL. Luego deben ser compiladas al lenguaje destino.	Definidas en IDL, con función de búsqueda para interfases adicionales, también maneja punteros a interfases.	Definidas en Java. Todos los métodos deben ser declarados en librerías de Java.
RPC	Genera stub/skeleton. Para facilitar la implementación utiliza una interfase dinámica.	Genera proxy/stub. Especifica la forma de realizar la llamada y como se mantienen las referencias.	Utiliza serialización de objetos Java.
Servidor	Deriva e implementa skeleton por herencia.	Implementa el servidor a través de una clase. Puede ser de configuración fija o con parámetros explícitos.	Implementado como una aplicación especial Java.
Programa principal del Servidor	Crea un nuevo objeto a través de una función en ORB.	Lo crea por medio de una nueva clase y se maneja con la librería COM.	Implementado con applets.
Clientes	Se conectan a los objetos remotos y realizan las llamadas. El acceso se realiza a través de librerías comunes.	Utilizan una función especial para crear un puntero a las interfases.	Implementado con applets o aplicaciones comunes.
Seguridad	Provee encriptamiento, autenticación y autorización para objetos, datos y servicios.	Puede ser provista por terceros. Maneja el concepto de cuentas, identificación y autoridad central.	Tiene un manager de seguridad que controla solamente las clases que no son locales. Las aplicaciones pueden definir su propia seguridad.
Objetos	Se deben declarar en ORB para que se realice la conexión de la llamada con la implementación.	Administración centralizada de objetos a través de instancias monikers.	Provee un servicio de naming para referenciar un objeto. La serialización se utiliza para el envío de mensajes y objetos.
Protocolos	Utiliza el protocolo Internet Inter – ORB	DCOM actúa en sí mismo como un protocolo de tres capas que conecta diferentes componentes de software.	Propio de Java. Aunque posee un método nativo (JNI) para comunicarse con sistemas existentes en otra plataforma.



Manejo de Conexiones	Cada objeto invocado necesita un handle que se mantiene a través del ciclo de vida del objeto y el transporte sobre la red.	Utiliza garbage collection y ping para eliminar procesos.	Se realiza a nivel de máquinas Java virtuales, utiliza garbage collection para eliminar procesos.
Concurrencia	No especifica procesamiento paralelo.	Utiliza la concurrencia provista por los sistemas operativos nativos donde corre.	Es multi thread, por lo tanto permite procesamiento paralelo.
Arquitectura	Abierta.	Semi Abierta.	Cerrada.
Lenguaje	Neutro.	Visual Basic preferentemente.	Java.



Principales Ventajas de Cada Sistema:

CORBA

- Amplio soporte para elementos heterogéneos de computación: vendedores, sistemas operativos, lenguajes de programación.
- Modelo de programación relativamente fácil.
- Soporta herencia múltiple en las interfaces.
- Provee amplia interoperabilidad entre sistemas.
- Posee un muy útil standard de especificaciones para su implementación.
- Plataforma abierta mantenida y actualizada.
- Verdadera orientación a objetos.
- Tiene a Object Request Broker como proveedor centralizado que brinda y garantiza el servicio.

DCOM

- Gran cantidad de usuarios y componentes en el mercado.
- Integración de software por código binario.
- Actualización de software on line.
- Interfaces múltiples por objeto.
- Amplia variedad de herramientas de programación disponibles.
- Plataforma propietaria de Microsoft.
- Es una extensión para Component Object Model.
- Objetos asignados digitalmente.
- Control de versiones embutido.

RMI

- Lenguaje simple.
- Amplia y fácil integración con Java.
- Gratis.
- Modelo de programación fácil de implementar.
- Posibilidad de ejecución en browsers standard de Internet.
- Interfaces múltiples para los objetos.
- Real capacidad de ejecutar los objetos polimórficos.
- Objetos accionados remotamente.
- Objetos clonados y transportados.
- Comunicación orientada a conexión o sin conexión.
- Buen manejo de la seguridad.



Conclusiones

Las principales similitudes se encuentran en los aspectos referidos a

- **Interfases:** concepto que todos manejan y que si bien tiene una declaración diferente utilizando JDK, poseen una funcionalidad similar.
- **RPC:** implementado por todas las arquitecturas, destacándose JDK, utilizando la serialización de Java como la forma de implementarlo.
- **Servidor:** presente en todas las arquitecturas como una implementación especial que cumple con esas funciones.
- **Seguridad:** en todos los casos la seguridad se puede implementar en niveles aceptables y son utilizadas sus funciones sin dificultades. En su versión nativa, CORBA tiene mayor nivel de seguridad que DCOM y JDK.
- **Manejo de conexiones:** es implementada en todos los casos manteniendo control de los objetos involucrados y verificando el fin de la conexión. Tanto CORBA como JDK utilizan el método de garbage collection para liberar de la memoria los objetos que ya no se utilizan.

Las diferencias principales son:

- **Clientes:** se diferencian en la forma de implementarlos, ya que CORBA lo hace por medio de librerías comunes, DCOM por medio de punteros a interfaces y JDK implementando con applets/aplicaciones.
- **Objetos:** en CORBA se utilizan las implementaciones declaradas en ORB, en tanto que DCOM centraliza los objetos por instancias llamadas monikers y JDK utiliza naming de objetos y serialización.
- **Protocolos:** CORBA tiene definido su propio protocolo llamado Inter-ORB, DCOM actúa en sí mismo como un protocolo de tres capas y JDK tiene un método nativo para comunicarse con otras plataformas.
- **Concurrencia:** no está especificada en CORBA, DCOM utiliza la nativa de los sistemas operativos donde corre y en JDK se implementan métodos especiales llamados thread que cumplen con esta funcionalidad.
- **Arquitectura:** CORBA es abierta, DCOM es semi abierta y JDK cerrada.
- **Lenguaje:** en CORBA es neutro, para DCOM preferentemente Visual Basic y en JDK se utiliza JAVA.



Podemos concluir que no hay una arquitectura que se distinga como la mejor en forma contundente sobre las demás. Lo más destacable de los aspectos que estuvimos analizando y donde podemos enfatizar nuestro estudio y decisión, es el área donde se pueden adoptar cada una de las aplicaciones para optimizar sus funciones. CORBA se destaca en la comunicación entre empresas, DCOM es lo mas apropiado para computadoras personales y Java RMI aparece como lo mas indicado en los desarrollos de Internet.



Comparación Java con C++

El lenguaje JAVA presenta varias ventajas y desventajas con respecto C++. Una desventaja bastante notoria es la performance de ejecución, dado que en la mayoría de los casos C++ ejecuta los programas en un tiempo considerablemente menor. Un análisis más profundo del mismo se puede observar en el anexo de la presente Tesis. Para poder contrarrestar esto, se están desarrollando “chips” JAVA que permiten de alguna manera disminuir el tiempo de ejecución. También se pueden realizar otras mejoras como ser la de generar un código nativo, pero esto va en contra de cómo trabaja la portabilidad de los applets y de las aplicaciones. Otra técnica similar es traducir Java a C con Java2c, pero esto también va en contra de la portabilidad. También hay una técnica llamada “just in time” que se está siendo considerada, que consiste en hacer la compilación dinámica dentro del intérprete bytecode, y la próxima vez que el bytecode es ejecutado, el actual código de máquina para la plataforma dada es sustituido en su lugar.

También hay que tener en cuenta que las diferencias no son muy importantes si por ejemplo realizamos un applet altamente interactivo que opera a través de Internet, puesto que el tiempo incurrido por la espera de respuesta de una persona y de la red será más importante que el tiempo puro del lenguaje.

El bytecode de Java hace posible la filosofía de “write one, run anywhere”. Se puede compilar un programa Java a bytecode en cualquier plataforma que contenga un compilador Java. Luego, ese bytecode puede ser ejecutado en cualquier implementación de la máquina virtual de Java. Esto va en detrimento del tiempo de ejecución del programa.

El lenguaje C y C++ son similares, pero Java es notoriamente diferente. Algunas de estas diferencias son las siguientes:

- Java no tiene archivo de encabezamiento.
- Java “conoce” ciertas características del sistema como ser la clase String y la output stream (System.out).
- No hay funciones globales como en C y C++, pero en su lugar todas las funciones (métodos) deben ser parte de la misma clase.
- main() es un método de la clase, pero este no opera específicamente como una instancia de la clase, y en su lugar es llamado “método de la clase” que es parte de la clase para el propósito de la aplicación. En C++ la equivalencia podría ser una “static member function”.
- La clase que define un programador, debe ser definida en un archivo llamado con el mismo nombre de la clase, pero con extensión java. No hay un requerimiento equivalente en C o C++.
- Para pasar un estado de salida desde el programa, Java usa System.exit(n). En C y C++ hay una función de librería llamada exit(), o uno puede retornar un valor desde main().

Por otro lado, Java tiene un buen manejo de memoria, pues usa la técnica de Garbage Collection, haciendo que el programador de aplicaciones Java se olvide de manejarla, evitando de esta manera acceder a espacios de memoria inadecuados. Esto



presenta una ventaja considerable frente a otros lenguajes, dado que muchos errores en tiempo de ejecución se deben al mal manejo de la memoria. La manera que Garbage Collection trabaja es simple, limpia automáticamente toda la memoria cuando en la misma no existen referencias de utilización. Un lado negativo del Garbage Collection es que hace a Java más lento que C++. También se puede decir que la facilidad de manejo de memoria de Java es de más alto nivel que C++.

Java realiza una validación de los tipos de datos en el momento de compilación, haciendo de esta manera, que los programas sean más seguros y confiables, dado que se evitan errores en tiempo de ejecución.

Java también presenta una ventaja importante con respecto al tamaño de los datos primitivos, dado que los mismos son uniformes en todas las aplicaciones, pues una vez que el mismo es compilado, cada intérprete de la Máquina Virtual Java lo puede interpretar sin ningún inconveniente.

En lo que respecta la manejo de errores, Java presenta una forma fácil y segura de manejarlos. El manejo de estos errores se realiza a través de excepciones. Para capturar las excepciones, las mismas se deben especificar en los módulos donde se lo desee utilizar y en caso de no poder atraparse un error en ninguno de los niveles superiores, el programa termina inmediatamente.

El disparo de una excepción es el mismo en C++ y en Java, con una excepción. En Java uno sólo puede disparar objetos de clases descendientes “throwable”, pero en C++ uno puede disparar valores de cualquier tipo.

Java no tiene variables o funciones globales como otros lenguajes comunes. Toda variable o función debe ser parte de una misma clase. Dentro de una clase las variables o funciones (métodos) pueden ser miembros regular de la clase, o puede ser una variable de una clase, o un método de una clase. Los métodos y las variables de la clase no operan sobre una instancia particular de una clase, pues un método de clase es típicamente usado como función dentro de la clase, mientras que las variables de clase están compartidas por todas las instancias de la clase.

C++ presenta herencia múltiple pero Java no. Sin embargo tiene una manera de hacer algo parecido que es lo que se llama interfase.

En C++ hay una noción de un destructor, una función llamada cuando una instancia de clase de objeto sale de escena. Los objetos de la clase Java no tienen un comportamiento similar. Ellos son dinámicamente alocados, en este punto un constructor es ejecutado sobre la instancia del objeto, y más tarde del garbage collection será el encargado de eliminarlo definitivamente. Java no tiene la noción de un método destructor que corre cuando la instancia del objeto deja de ser válida, pero en su lugar tiene un método finalize.

La mayoría de las otras plataformas son descriptas como una combinación de sistema operativo y hardware. La plataforma Java tiene dos componentes de software:



- La máquina virtual Java (Java VM).
- Java Application Programming Interface (Java API)

Java no maneja los tipos de datos enumerados como sí lo hace C++, pero los mismos se pueden simular. Con el uso de interfases Java puede manejar herencias múltiples.

Java es un lenguaje dinámico en el sentido que la carga de las clases se hace en runtime, es decir, en el momento que se necesitan, a diferencia del lenguaje C++, que realiza el linking en el tiempo de compilación.

Java presenta como novedad importante que en la ejecución de un programa este debe ser invocado con el nombre de la clase y no con el nombre del programa. No nos parece tan ventajoso, pues normalmente cuando uno quiere ejecutar un programa lo hace con el nombre del ejecutable. Esto se podría utilizar para restringir la ejecución de una aplicación o applet y que la misma sólo sea ejecutada por quien la conoce, permitiendo de esta manera, un cierto nivel de seguridad extra.

Java permite la asignación de datos primitivos más chicos a datos primitivos más grandes y por lo tanto es válido una asignación de un int a un short, pero no permite la asignación de un dato primitivo de mayor tamaño a uno de menor tamaño.

Para el caso de llamada a un método, Java busca todos aquellos que tienen el mismo nombre que el método invocado, luego busca de estos los que coinciden en la misma cantidad de parámetros, y luego los parámetros que son del tipo que se pueden asignar según los valores de los argumentos. Si sólo matchea un único método, entonces este es el método que será invocado. Si hay más de un método que coincide en todos los datos descriptos arriba, busca los métodos para los cuales todos los tipos de los parámetros pueden ser asignables a otros parámetros de otros métodos, y remueve esos otros métodos del conjunto de métodos considerados. El método removido es considerado “menos específico”. Si luego de este descarte queda un sólo método, éste es invocado normalmente. Si queda más de un método, entonces la llamada es ambigua.

Java tiene arreglos que en su uso son como los arreglos convencionales de C/C++, pero hay una diferencia en Java cuando se refiere a los manejos de los arreglos. Una gran diferencia es que Java chequea los subíndices, pues si uno recorre un arreglo y se trata de acceder a un elemento no contenido en la definición del arreglo, Java lanza una excepción. Java “pierde” tiempo cuando se recorren arreglos pues siempre realiza un chequeo para saber si no se está accediendo a un elemento indebido.



Capítulo 2

Sistema a Implementar

Presentamos someramente el Sistema de Distribución de Objetos (ODS) [RNL 97], que es una especificación general para la distribución de objetos, intentando optimizar recursos, tiempos de respuesta y comunicaciones en un esquema distribuido de computación.

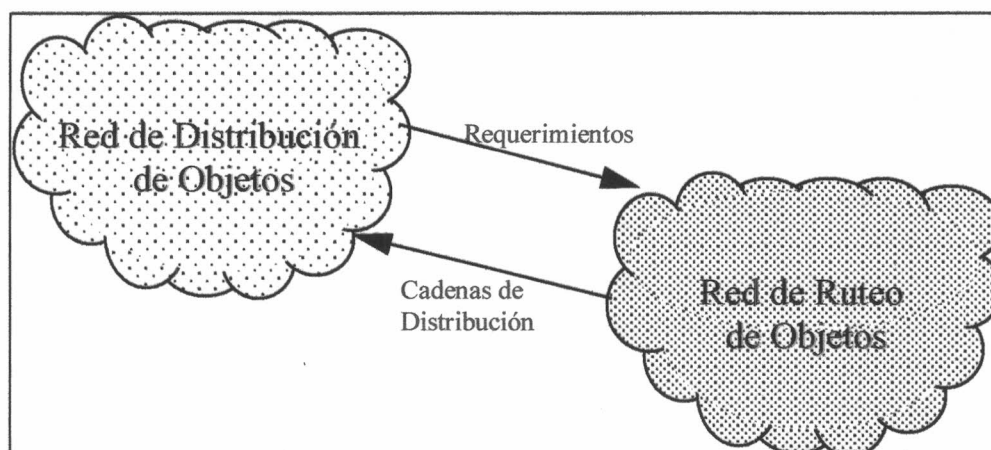
La especificación toma como modelo redes de distribución de la vida real y trata de aplicar ese paradigma a un ambiente de computación distribuida, logrando un esquema sencillo y novedoso para mantener objetos distribuidos actualizados, con ciertas restricciones en cuanto a las versiones de los objetos en el sistema.

El modelo aparece especificado en una forma muy flexible debido a la división de sus componentes en unidades lógicas - funcionales concretas y al análisis de los correspondientes protocolos de comunicaciones. Además deja abiertos temas de implementación, que hacen que un desarrollo que trate de aproximarse a dicha especificación tenga a la vez, posibilidades creativas para completar la especificación detallada y un importante desafío en cuanto a la implementación real del sistema definido.

Sistema de Distribución de Objetos (ODS) [RNL 97]

El objetivo de ODS es crear un modelo de distribución similar a los utilizados en la vida real como las cadenas de distribución de alimentos, distribución de publicaciones, etc. Las fábricas tienen logísticas de distribución para los productos que fabrican y el consumidor final accede a los mismos en los negocios minoristas, que por lo general están más cercanos a sus casas.

El Sistema de Distribución de Objetos está formado por dos redes independientes: La Red de Distribución de Objetos y la Red de Ruteo de Objetos.





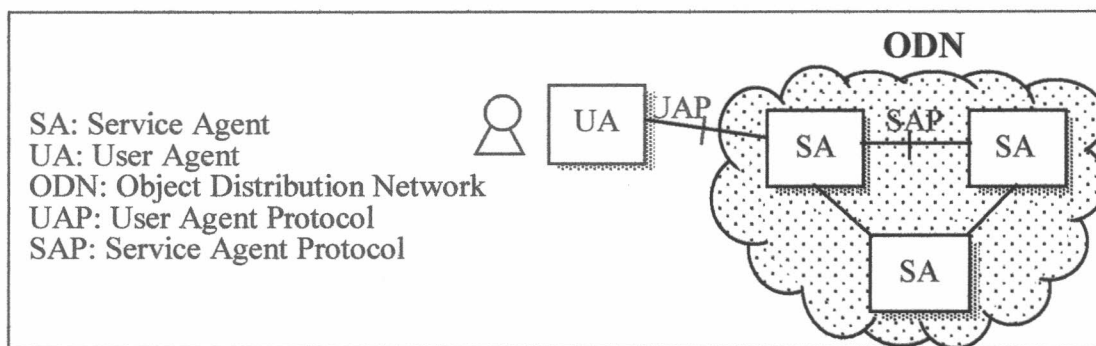
La Red de Distribución está compuesta por un conjunto de Agentes de Servicio que cooperan entre sí para lograr que los objetos puedan ser replicados en forma transparente para los usuarios del sistema.

La Red de Ruteo es la encargada de generar las redes lógicas de comunicaciones que se utilizarán para realizar la distribución de los objetos entre los usuarios del sistema. Su función es informar a la Red de Distribución la forma en que se deben rutear los objetos, para este fin se generan las cadenas de distribución.

Ambas redes funcionan en forma independiente, para esto se definió un protocolo de comunicaciones por medio del cual se realizan las peticiones y se reciben las respuestas por parte de los componentes de las redes. Los algoritmos para establecer las Redes de Distribución son independientes de los objetos que se estén distribuyendo, esto aumenta la independencia entre ambas redes

El sistema esta pensado para acceder a versiones de objetos que pueden no ser las últimas. Esto no resulta aceptable en ciertas aplicaciones como las necesarias para tomar decisiones en tiempo real o una videoconferencia. Pero en cambio, es adecuado para objetos que no sufren modificaciones constantes, como por ejemplo, documentos o papers.

Red de Distribución de Objetos (ODN)

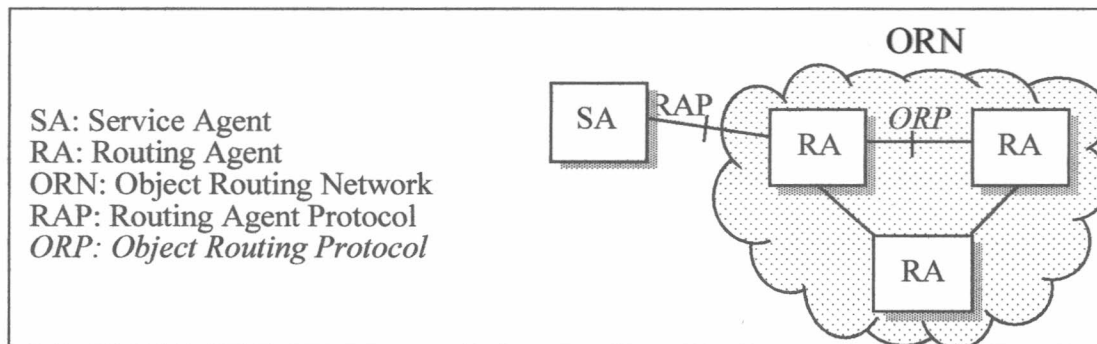


Los Agentes de Servicio (SA) que componen la red de distribución colaboran para replicar los objetos según sean requeridos por los usuarios. Utilizan para la comunicación un protocolo especial donde hacen las veces de cliente o servidor según el flujo de información establecido por la Red de Ruteo.

El Agente Usuario (UA) es el módulo que permite el acceso de los diferentes usuarios a la red de distribución. El Agente Usuario no pertenece a la red de distribución y se comunica con la misma a través de un protocolo establecido para darle independencia y poder ser implementado con mayor flexibilidad.



Red de Ruteo (ORN)



Los diferentes Agentes de Ruteo (RA) que componen la Red de Ruteo colaboran para generar las cadenas de distribución que son luego pasadas a la Red de Distribución, que es la que se encarga de replicar los objetos.

Se comunican a través de un protocolo común para compartir la información necesaria de toda la red a fin de establecer las cadenas de distribución. Cada Agente de Ruteo puede tener uno o más Agentes de Servicios a los cuales les informa la distribución que deben realizar con los objetos que reciben o que generan.

Conclusiones

El Sistema de Distribución de Objetos (ODS) es un esquema a la vez sencillo y novedoso para mantener objetos distribuidos actualizados y optimizar tanto las comunicaciones entre los componentes de la red como el manejo de los demás recursos del sistema.

La división en dos redes funcionalmente dependientes pero conceptualmente distintas del sistema brinda la flexibilidad de poder implementar una o ambas en un intento de probar en forma práctica la factibilidad de dicho desarrollo.

Los protocolos de comunicaciones están muy bien definidos y establecen un modo de comunicación que habilita que los diferentes componentes del sistema, Agentes de Servicio, Agentes de Ruteo y Agentes de Usuario, puedan ser implementados en diferentes lenguajes o arquitecturas con la sola condición de que puedan compartir el canal de comunicación para intercambiar mensajes.

En la especificación del Sistema de Distribución de Objetos (ODS) [RNL 97], no se establece detalladamente el nivel de seguridad que se debe adoptar para una eventual implementación. Tampoco hay especificaciones estrictas sobre la forma de almacenar los documentos ni los datos de control correspondientes al sistema en sí mismo. Estos dos aspectos, unidos a la flexibilidad que brinda para la implementación del Agente de Usuario, tanto en lo que respecta a funcionalidad como apariencia, hacen que la especificación de ODS represente un gran potencial de desarrollo y el desafío de terminar de especificar y concretar las interesantes ideas que se exponen en el trabajo.



Capítulo 3

Necesidades del sistema

Las necesidades que deseamos cubrir del sistema son: buen manejo de la base de datos, es decir, que sea independiente del lenguaje elegido para el desarrollo de ODS, a su vez, necesitamos que sea portable, o sea, que una vez compilado se pueda ejecutar en otra máquina sin necesidad de volver a compilarlo en esa máquina. También debe ser seguro, dado que se va a desarrollar en un ambiente como Internet. Otro punto a tener en cuenta, es que maneje diferentes tipos de conexiones, tales como punto a punto, datagramas, etc.

Bases de Datos

En la misma se van a almacenar los datos relacionados con la identificación de los objetos y las tablas internas de control del sistema. La base de datos debe manejar sentencias SQL (Structured Query Language) por ser este el standard utilizado, y a la vez, debe ser una base de datos relacional.

El ser independiente del lenguaje a utilizar, permite la flexibilidad de poder cambiar el motor de base de datos, y que los cambios que se deban efectuar al sistema sean mínimos, por ejemplo pequeños cambios en un archivo de configuración. Con esta característica del lenguaje nos permite realizar mejoras en nuestro sistema cuando se efectivizan mejoras en el motor de la base de datos. A su vez, todo cambio que se haga, no requerirá cambiar el cliente, siempre y cuando se respete el protocolo de comunicaciones entre el cliente y el servidor.

Alternativas de Comunicaciones

En este punto, es deseable que el lenguaje que se vaya a utilizar para la implementación de ODS, pueda manejar diferentes tipos de conexiones, como ser punto a punto, datagramas, etc. Si el lenguaje maneja diferentes tipos de conexiones, nos permite realizar por ejemplo, una conexión punto a punto entre el cliente y el servidor, y establecer conexiones de datagramas entre los servidores.

Es importante una eficiente detección de los errores producidos en la transmisión, para poder tomar la medida necesaria al respecto. Los errores a manejar deben contemplar entre otras cosas, detección de la caída de un nodo cuando se esta realizando una transmisión de mensajes en una conexión punto a punto, falta de respuesta de un nodo al intentar establecer una comunicación, etc. Los mismos deben ser fácilmente atrapados para su interpretación y tratamiento y así poder actuar en consecuencia.



También es deseable que se puedan interpretar URL's, TPC y UDP sockets, y direcciones IP dado que son requerimientos indispensables para la interconexión entre redes.

Portabilidad y Mutiplataforma

El lenguaje no debe depender de la implementación de la especificación, cualquiera sea el tipo de datos que se utilice, el comportamiento del lenguaje no debe depender del compilador.

Por las características de este sistema se desea que el lenguaje sea portable, el programa ya compilado debe funcionar sobre distintos sistemas operativos como ser Unix, Windows'95 o NT.

Estas características son imprescindibles dado que ODS se desarrolla en un ambiente como Internet donde los clientes que acceden a la red pueden trabajar sobre distintos sistemas operativos, y por ende, necesitamos un lenguaje que sea portable.

Seguridad

Como ODS está pensado para ambientes de redes como Internet, es necesario poner mucho énfasis en la seguridad. El lenguaje debería controlar el stack en tiempo de ejecución, hacer un buen manejo de la memoria - liberar la misma en forma automática cuando deja de usarla -, validar los tipos de datos en el momento de compilación para evitar sorpresas en el momento de ejecución. También debe poseer cierta facilidad para el manejo de los errores que se pueden producir durante la ejecución.

Otros Aspectos

Otros aspectos a tener en cuenta para calificar al lenguaje de programación elegido para la implementación son: capacidad para transmitir gran volumen de información, manejo de objetos, fault tolerance, arquitectura cliente / servidor, programación orientada a objetos, manejo de herramientas gráficas, etc.

Conclusiones

Es importante contar con un lenguaje de programación que brinde solución a estas necesidades. Si contamos con estas características, existe la posibilidad de desarrollar mejores aplicaciones y con mayor facilidad.

El desarrollo de ODS también requiere que el lenguaje pueda tener una interacción fluida con diferentes bases de datos, por la necesidad funcional de intensivo manejo objetos almacenados en la base de datos y para no atar la implementación a un determinado motor de base de datos.



En el análisis de la mejor herramienta de desarrollo tendrá vital importancia la facilidad de comunicaciones entre procesos, ya que este es otro de los usos intensivos que ODS hará del lenguaje elegido.

La implementación del sistema sobre Internet también requiere de importantes capacidades de portabilidad y multiplataforma, dada la posibilidad de la existencia de diferentes sistemas operativos en Clientes y Servidores.



Capítulo 4

Decisiones de Implementación del Prototipo

Luego de la comparación entre las arquitecturas y lenguajes, vistos en el capítulo I, se decidió en un principio implementar el prototipo de ODS utilizando la arquitectura RMI, bajo el entorno de JDK, utilizando el lenguaje de programación Java.

Se utilizó en la aplicación una base de datos standard y de fácil acceso. También se tuvo en cuenta en la elección de la misma que no sería necesario almacenar un gran volumen de información, ya que los documentos propiamente dichos no se guardan dentro de la base de datos. El motor de base de datos elegido fue Access.

La alternativa que vamos a implementar es una de las opciones que se puede obtener limitando el alcance del sistema ODS [RNL 97] de objetos en general, a la Distribución de Documentos sobre Internet.

Parte de las decisiones de implementación incluyen el diseño detallado y la programación de los Agentes de Servicio que son el componente básico para el sistema de distribución de documentos. También vamos a implementar un Agente Usuario para brindar a los usuarios una herramienta específica para acceder al sistema.

Se incluye el diseño y la estructura de los documentos en el sistema, destacándose que cada componente del sistema tiene acceso a diferentes partes del objeto, según su función y el nivel de información que necesita administrar en el sistema.

Se establece el alcance y se detallan las decisiones de diseño que se tuvieron en cuenta para los demás módulos del sistema. La administración del sistema no se desarrolla en el prototipo, ésta debe realizarse en forma manual. El diseño de la base de datos incluye todas las tablas necesarias para la administración y operación del sistema, los documentos se almacenan en forma externa en el formato propio de cada uno.

Para las comunicaciones se realizaron dos diseños diferentes, uno orientado a conexión que garantiza la transmisión entre los componentes, y otro menos seguro para comunicaciones que no necesitan garantizar la seguridad.

El esquema de publicaciones de documentos es sencillo, cada usuario con permiso de editor puede ingresar sus publicaciones, las cuales quedan a disposición de los demás participantes del grupo de interés al que pertenece.

JDK – RMI

Dada las características del desarrollo a implementar, consideramos que esta arquitectura es la que mejor se adapta a los requerimientos del sistema ODS.



Teniendo en cuenta que el prototipo va a funcionar sobre Internet, y dado que la arquitectura RMI es la que presenta mayores virtudes en este tipo de entorno, es uno de los motivos de la decisión en la elección del mismo.

Una característica diferencial que presenta RMI frente a las demás arquitecturas, es la posibilidad de procesamiento paralelo. La implementación se realiza por medio de threads, que generan procesos concurrentes e independientes.

Otro de los aspectos importantes que presenta esta arquitectura y que influyó en la decisión de su elección, es la capacidad de portabilidad y multiplataforma que nos permite emplear el mismo código compilado en diferentes sistemas operativos

Java

Una vez analizadas las características de Java y C++, decidimos realizar la implementación del prototipo utilizando el lenguaje Java, dado que presenta algunas ventajas que consideramos importantes frente a C++.

Java utiliza la técnica de Garbage Collection para el manejo automático de memoria, permitiendo una fácil administración de la misma, evitando errores en tiempo de ejecución, haciendo de esta manera los sistemas más confiables, y facilitando la programación.

Otras de las características por las cuales decidimos implementar el prototipo utilizando Java son:

- Lenguaje fuertemente tipado, evitando errores en tiempo de ejecución ya que se detectan en el momento de compilación.
- Librerías dinámicas que permiten modificar las mismas sin necesidad de recompilar la aplicación.
- Capacidad de manejar applets, permitiendo la utilización de browsers estándares de Internet.
- Facilidad para el manejo de errores por medio de excepciones.

La decisión de implementar RMI nos restringe el uso del lenguaje, puesto que éste debe ser desarrollado utilizando Java.

Siendo Java uno de los primeros lenguajes para el entorno de Internet, y considerando que aún no existe claro horizonte sobre el futuro del lenguaje, nos pareció desafiante utilizarlo para el desarrollo de la aplicación ODS.

Motor de Base de Datos

Como Java no presenta una base de datos nativa, pero sí drivers para conectarse a diferentes motores, y teniendo en cuenta las necesidades del sistema, decidimos utilizar Access.



Utilizando la base de datos Access, obtenemos algunas ventajas tales como:

- Bajo costo de adquisición, dado que la misma se distribuye en el paquete Office de Microsoft y no necesita licencias extras para su utilización.
- Este motor maneja sentencias estándares de SQL, pudiendo ser reemplazada la misma con un bajo costo de modificación de la aplicación.
- La capacidad de almacenamiento y tiempo de respuesta estaban acorde a las necesidades del prototipo.

Distribución de Documentos

La distribución de documentos se adapta y cumple las características necesarias para el esquema de ODS, ya que estadísticamente los documentos se adaptan a las limitaciones que el sistema plantea:

- Bestavros [Bestavros 95b] plantea que el porcentaje de actualización de los documentos que denomina 'globalmente populares' es a lo sumo de un 0,5% por día, restringido además a un pequeño subconjunto de ellos.
- Blaze [Blaze 93] plantea que la probabilidad de que un documento vuelva a modificarse va disminuyendo a medida que transcurre el tiempo desde su última modificación.

Como la implementación y pruebas se realizará en el ámbito universitario, creemos que será la distribución de documentos muy útil en el ambiente y además se adapta al tipo de comunicaciones usuales entre los diferentes actores que componen Internet en la actualidad.

Alcance del Prototipo

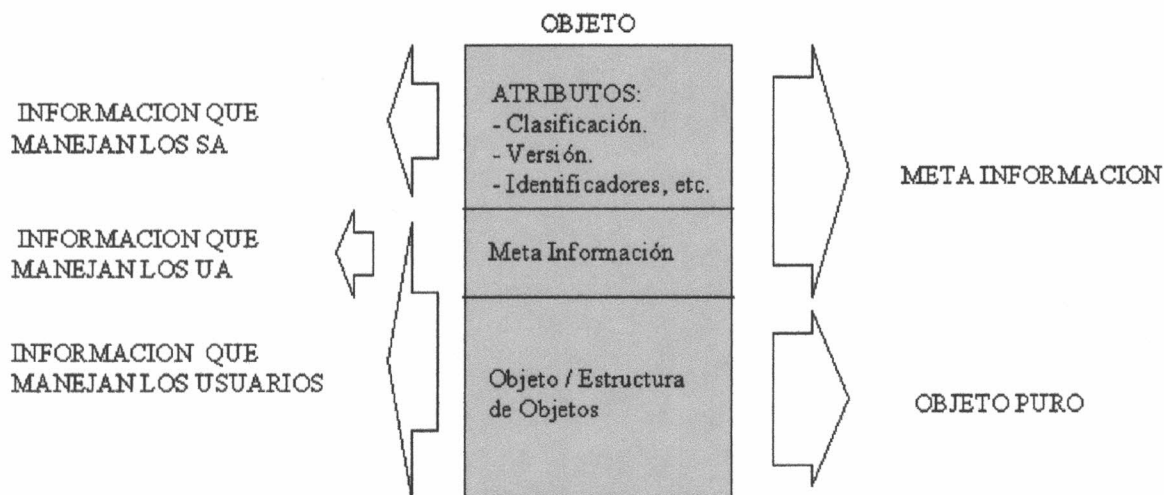
Vamos a implementar el Agente de Servicio que es el componente básico para poder utilizar la red de distribución, desarrollando toda la funcionalidad descrita en ODS [Righetti97] para los Agentes de Servicio.

Vamos a implementar un Agente Usuario que permita el acceso de los usuarios a la Red de Distribución para poder colocar y consultar objetos de la misma. No es imprescindible realizar este desarrollo ya que se podría acceder a la Red a través de un navegador estándar de Internet, pero nos pareció adecuado desarrollar uno ad hoc para probar la implementación y funcionamiento de los protocolos correspondientes a esta parte del sistema, así también como test para las capacidades gráficas del lenguaje.



Estructura de los Objetos

Se presenta el tipo de objeto que se diseñó para el prototipo. Esta estructura incluye las necesidades de control para la administración de dichos mensajes en la red y la codificación de los mismos según el tipo



La información del objeto propiamente dicho, se encuentra en lo que denominamos Objeto Puro, todo el resto de información que corresponde a información de control y meta información del objeto, se denomina Meta Información, ya sea para el uso de los Agentes de Servicio como para los Agentes de Usuario.

La información que manejan los Agentes de Servicio corresponde a los atributos generales de los documentos, por ejemplo, tema al cual pertenecen los documentos, autor, versión, origen, etc. Toda esta información es necesaria para administrar la distribución de los mismos.

Esta estructura permite el manejo de objetos compuestos. Es decir, un objeto compuesto puede representar una estructura de objetos que a su vez pueden ser objetos compuestos o simples. Esta definición brinda las facilidades necesarias para componer volúmenes o documentos cuyos capítulos son redactados por diferentes autores. También se habilita la posibilidad de modificar o agregar un capítulo de un documento sin necesidad de cambiar todo el documento, sino solamente la parte que fue modificada o agregada.

Cada usuario con permisos de productor puede publicar documentos.

Base de Datos

Es fundamental el diseño de la base de datos y del módulo de administración de la misma, ya que los demás módulos del sistema en cada Agente de Servicio se comunican entre ellos a través de la Base de Datos. Por lo tanto, hemos tenido especial



cuidado en la definición de las características que debe reunir y en el diseño de las tablas que hacen a la comunicación interna de los módulos.

Se definió una base de datos de uso interno del sistema para administrar todos los elementos que componen la Red de Distribución y los usuarios. La parte de información pura de documentos que se administra no forman parte de la base de datos sino que se almacenan en el formato propio de cada documento implementando un sistema de identificación que hace que cada documento posea un identificador único.

La base de datos se utiliza para almacenar todos los datos referidos a la meta información de los objetos, el envío y recepción de ofertas, la actualización de clientes por tema de interés, los permisos de los usuarios y todas las tablas de control del sistema.

División de los Módulos

Se estableció la división en módulos tomando en cuenta la funcionalidad de la Red de Distribución y el Agente de Usuario.

La subdivisión de las diferentes funciones en módulos facilita la definición e implementación de los mismos pero hace más complejas las interfaces entre ellos. Surge también la necesidad de programar un módulo principal que administre y gestione el funcionamiento de los demás módulos.

Además esta decisión permitió que se pudieran implementar y probar en forma paulatina los módulos en la medida en que se iban terminando.

Métodos de Comunicación Establecidos

Sabemos que en Internet la comunicación estándar es TCP/IP. El nivel de comunicaciones donde se va a manejar la aplicación que vamos a desarrollar sobre Internet se corresponde con el nivel de Red y Transporte OSI y controla el direccionamiento de la información.

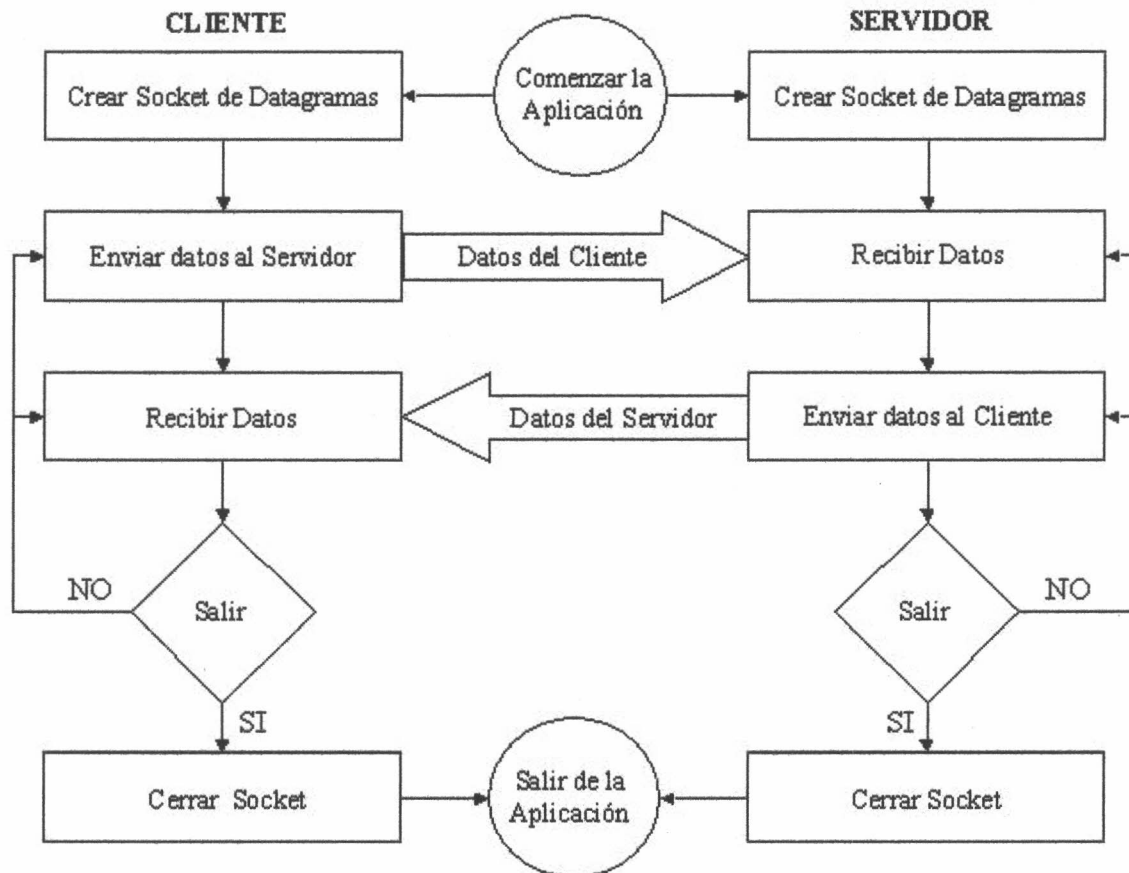
El nivel de Red que se corresponde con el IP se trata de un protocolo para el intercambio de datagramas en modo no conectado. Esto no garantiza la llegada de mensajes (cosa que se hará con el TCP). El algoritmo de direccionamiento de Internet se basa en tablas de direccionamiento de los datagramas difundidos por los gateways.

Nivel de Transporte en Internet se corresponde con el de Transporte OSI, garantizando la seguridad de la conexión y el control del flujo. Incluye el Protocolo de Control de Transmisión (TCP) y el Protocolo de Datagrama de Usuario (UDP).

La comunicación entre los Agentes de Servicio se dividió en dos tipos dependiendo del grado de confiabilidad necesaria.



ODP: comunicación orientada a datagramas, no es del todo confiable, cada emisor envía paquetes con la información que desea transmitir y no recibe certificación de la otra parte para verificar si el paquete fue recibido correctamente. Corresponde al nivel UDP de Internet y es utilizado en el sistema para las comunicaciones en que no es necesario garantizar el envío de la información, por ejemplo, en el envío de ofertas.



Especificaciones del Server de Datagramas

1. Crea el socket de Datagramas en un port específico.
2. Espera en el port especificado por paquetes que lleguen.
3. Responde a los paquetes recibidos de acuerdo al protocolo establecido.
4. Volver al paso 2, al paso 3 o continuar al paso 5.
5. Cerrar el socket de datagramas.



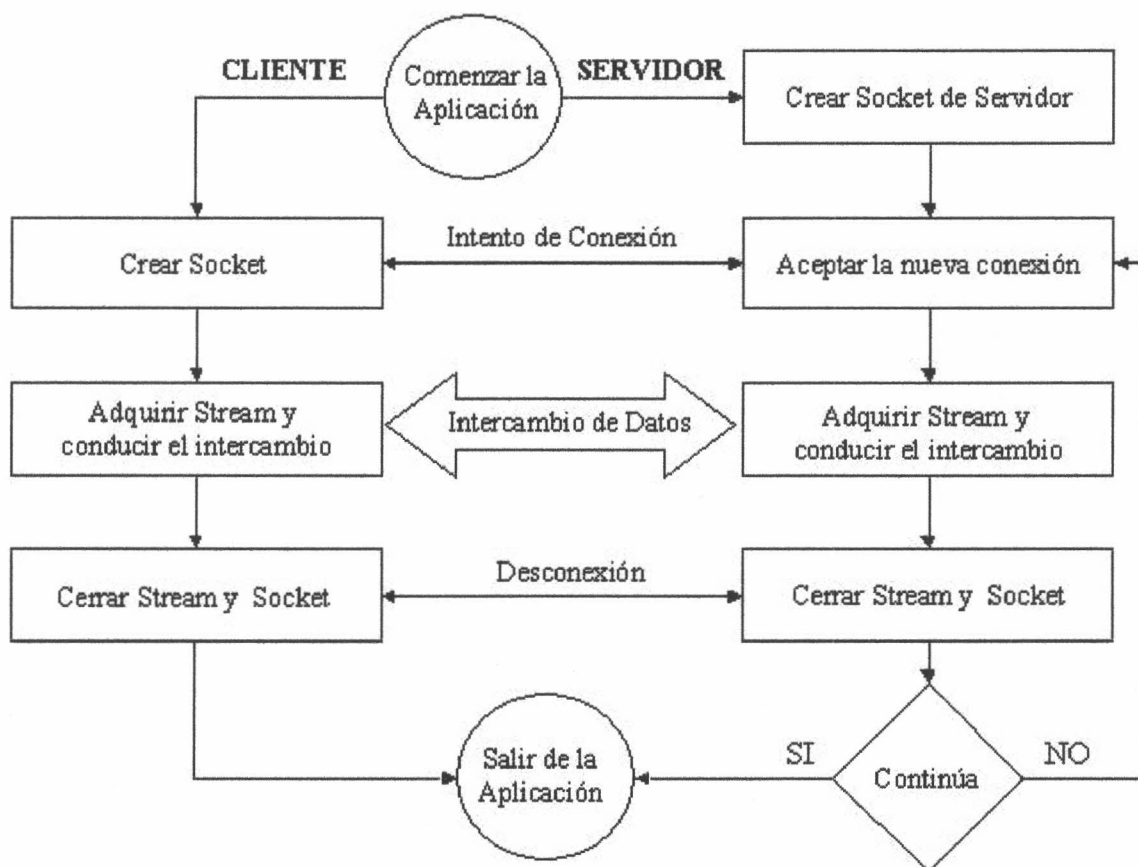
Especificación del Cliente de Datagramas

El cliente correspondiente tiene los mismos principios que el servidor con la excepción que el cliente es quién inicia la conexión.

1. Crea el socket de datagramas en un port específico.
2. Crea la dirección para enviar datos.
3. Envía los datos de acuerdo al protocolo establecido.
4. Espera para recibir datos por el port.
5. Va al paso 3 (para enviar mas datos), al 4 (esperando para recibir) o al 6 (salida).
6. Cerrar el socket de datagramas.



OTP: comunicación orientada a conexión, es confiable, se establece la comunicación entre las dos partes interesadas y no se suspende ni se corta dicha conexión hasta que no se terminan los datos a comunicar. Corresponde al nivel TCP orientado a conexión en Internet y es utilizado en el sistema para las comunicaciones que necesitan asegurar la información enviada, por ejemplo, cuando se está transmitiendo un documento.



Especificación del Servidor con Sockets

1. Crear un socket y comenzar a escuchar por mensajes en el mismo.
2. Llamar al método `accept()` para obtener una nueva conexión cuando llega un requerimiento.
3. Crear input y output streams para el socket.
4. Intercambiar datos de acuerdo al protocolo de la aplicación.
5. Cerrar el streams de cliente y el socket.
6. Ir al paso 2 o continuar con el paso 7.



7. Cerrar el socket del Servidor.

Especificación del Cliente con Sockets

1. Crear la conexión para el socket del cliente.
2. Crear los streams para leer y escribir por el socket.
3. Utilizar el stream para el intercambio de datos de acuerdo al protocolo de la aplicación.
4. Cerrar el streams.
5. Cerrar el socket.

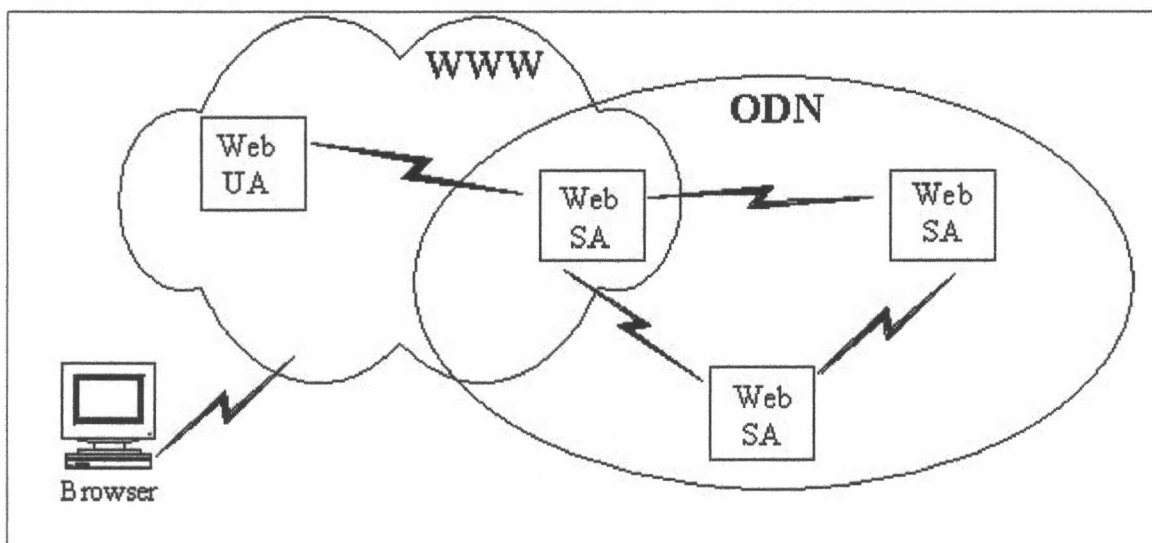
En la especificación de los módulos del sistema se especifican los casos en los cuales se utilizó cada uno de estos métodos. También se eligieron ports para la comunicación entre los módulos del sistema que no son usualmente usados por otros servicios en Internet.

Alternativas de Implementación

El modelo implementado cuenta con la posibilidad de comunicarse a través de las redes existentes utilizando el protocolo IP.

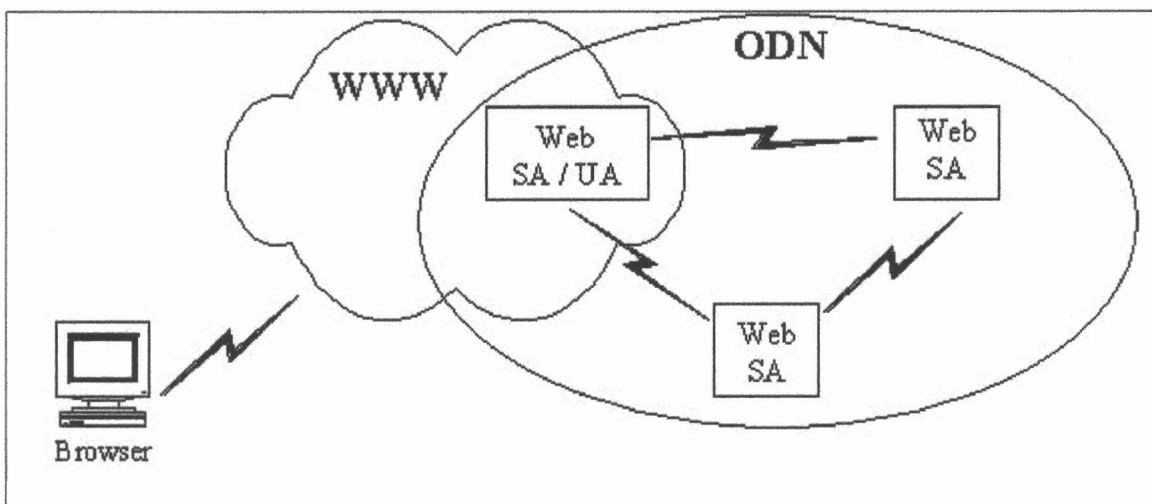
Tanto los Agentes de Servicio entre sí, como los Agentes de Servicio y Agentes de Usuario, cuentan con funcionalidad para utilizar Internet/Intranet, utilizando cualquiera de los protocolos definidos: ODP u OTP.

Una alternativa posible es la de transformar los Agentes de Usuario y los Agentes de Servicio para que puedan ser ejecutados en un Servidor Web y accedidos por medio de Browsers a través de una página HTML. En nuestra implementación no presenta ningún inconveniente ya que como los Agentes están implementados en lenguaje Java es muy fácil transformarlos en Applets y ejecutarlos desde páginas HTML.



Esta alternativa tiene la ventaja de poder dejar disponible el sistema a cualquier usuario autorizado que posea acceso a Internet, sin necesidad de instalar en su computadora el Agente de Usuario y configurarlo con algún Agente de Servicio. A la flexibilidad que brinda este esquema hay que agregarle la economía de actualización de versiones, ya que solamente se actualizan los sistemas en los servidores.

Una variante importante a este esquema es la posibilidad de unificar el Agente de Servicio y el Agente de Usuario en una sola aplicación.



A las ventajas antes mencionadas en lo que respecta a los accesos por Internet a las aplicaciones hay que agregarle que este esquema presenta mayor cohesión entre los módulos, evitando el protocolo de comunicaciones entre el Agente de Servicio y el Agente de usuario, con la consiguiente ganancia en seguridad y performance.

Por otro lado hay que aclarar que este esquema presenta la desventaja de tener menos flexibilidad que el anterior, ya que no pueden ser implementados Agentes de Usuario con diferentes lenguajes o funcionalidad.



Conclusión

Considerando las necesidades para el desarrollo del prototipo y el ambiente donde el mismo va a funcionar, es que decidimos la utilización de RMI y Java como arquitectura y lenguaje de programación respectivamente. Como motor de base de datos se utilizará Access.

Para la implementación del prototipo se decidieron algunas restricciones sobre el modelo planteado en el Sistema ODS [RNL 97] y se especificaron con criterio propio las opciones que no se encontraban con suficiente detalle o estaban sin definir.

Se implementarán tanto el Agentes de Servicio como el Agente Usuario que componen la Red de Distribución y el front end con los usuarios.

El detalle de la estructura de los objetos que maneja el sistema fue diseñado teniendo en cuenta la funcionalidad de los diferentes módulos que actúan en el ciclo de un documento, y para conservar tanto la información de control como los datos propiamente dichos dentro de una misma estructura.

La publicación de documentos se implementa en forma sencilla, cada autor puede editar publicaciones en los temas de interés en los cuales está autorizado.

La administración del sistema se estableció en forma centralizada en cada Agente de Servicio, pero teniendo en cuenta la coherencia entre los permisos de usuarios en los grupos de interés y las relaciones entre los diferente Agentes del Servicio.

Se diseñó un nivel de comunicación confiable que garantiza la conexión entre los componentes comunicados y otro nivel donde la información se fragmenta y no tiene la mayor confiabilidad.



Capítulo 5

Desarrollo del Agente de Servicio

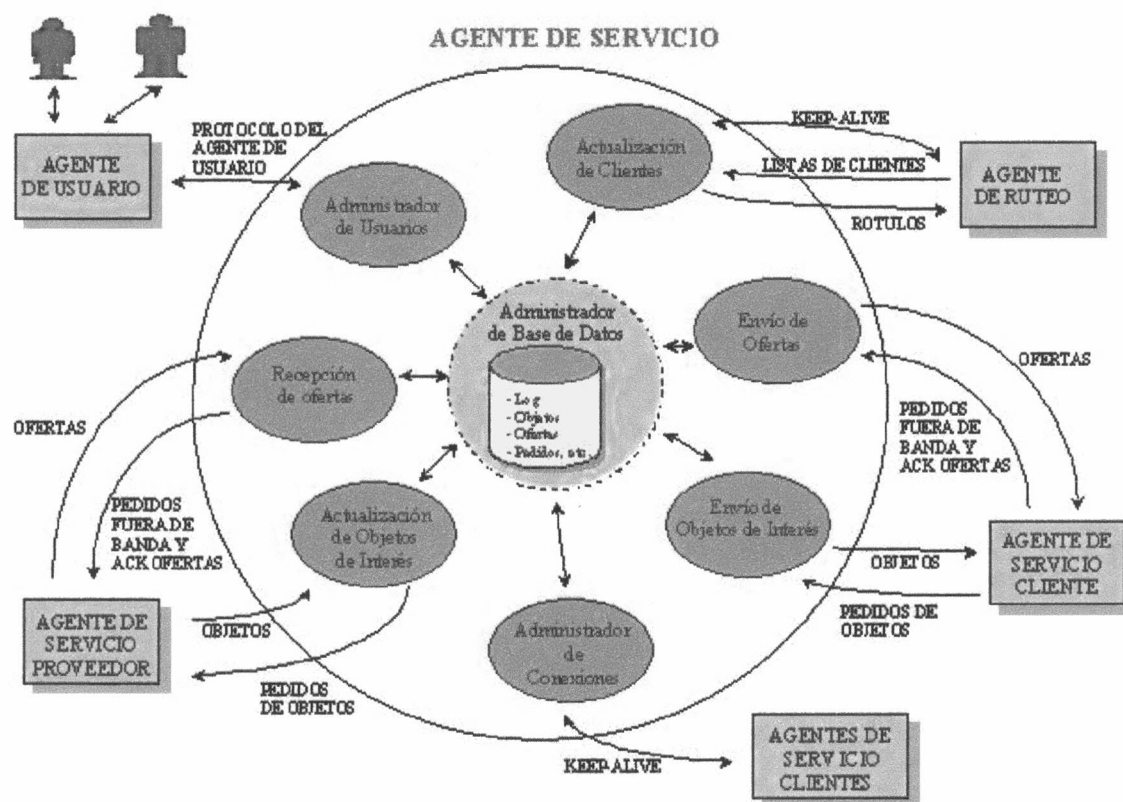
El agente de Servicio fue desarrollado en diferentes módulos independientes, pero relacionados y complementarios funcionalmente, para cumplir con todas las tareas que se diseñaron en los Agentes de Servicio. Se presenta en esta parte el diseño y las decisiones adoptadas para cada uno de los módulos del sistema.

El desarrollo en diferentes módulos permite darle al sistema mayor flexibilidad y robustez. Además, como pruebas iniciales del prototipo pudimos testear los módulos que completábamos mientras realizábamos la implementación a pesar de no contar con todo el sistema en funcionamiento. Esto contribuyó en gran medida a corregir los errores iniciales de programación y comunicaciones, y demostró ser un camino válido para encarar este tipo de sistemas con un lenguaje que no había sido probado integralmente todavía.

Se ha realizado una distribución de los diferentes sistemas de comunicación para asegurar que se optimizan los recursos del sistema y de la red. Se utiliza una conexión segura y otra menos segura, según las necesidades de cada módulo y su función.

Esquema General de los Módulos

Se presenta a continuación un esquema con los diferentes módulos en que se dividió el sistema. El criterio para establecer la división se basó en la funcionalidad que deseábamos darle a los Agentes de Servicio, tratamos que cada función específica estuviera implementada en un módulo diferente.



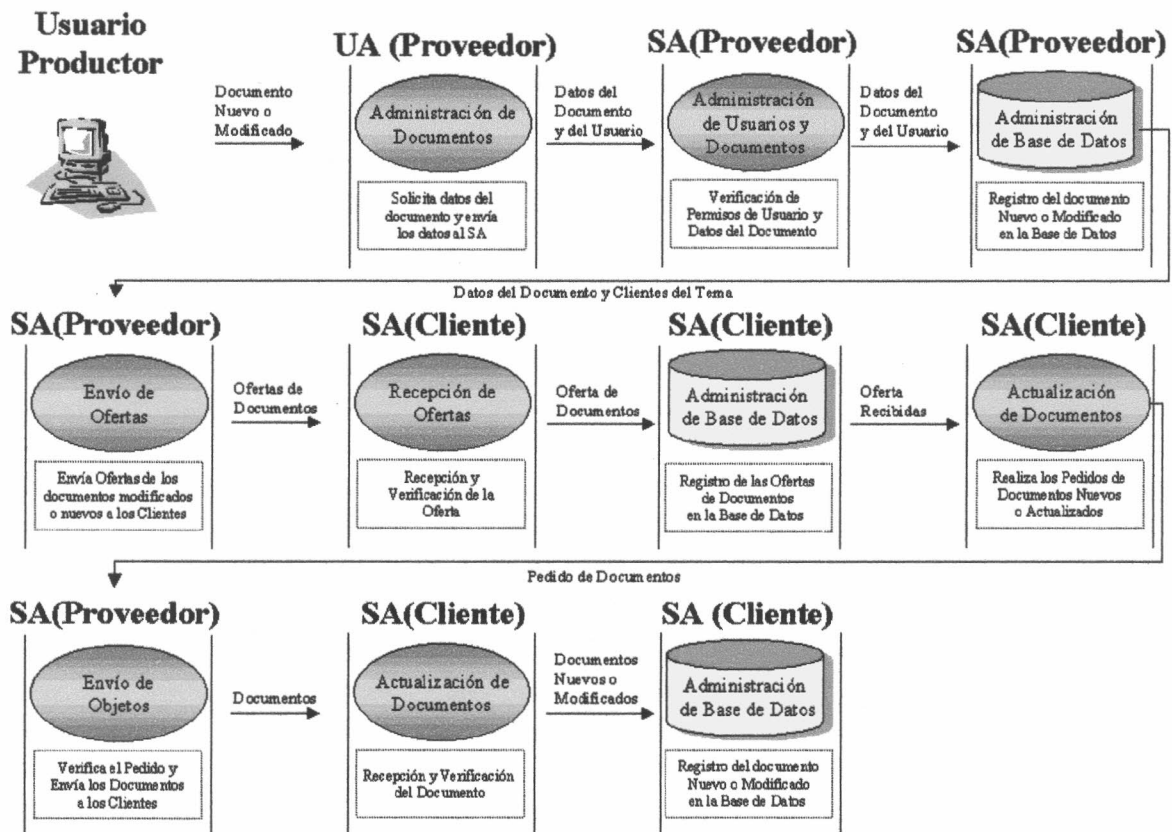
El hecho de que los módulos del Agente de Servicio fueran implementados en forma independiente también tiene la ventaja que no necesariamente deben residir todos en el mismo servidor, es más, si tenemos cuidado en definir y establecer las diferentes funciones en forma lógica podemos tener más de un módulo del mismo tipo funcionando en la misma configuración de Agente de Servicio. Por ejemplo, el módulo de Administración de Objetos y Control de Usuario puede estar repetido varias veces para acelerar el acceso de los usuarios en el mismo Agente de Servicio sin cambiar o modificar el comportamiento del sistema en general.

Como se puede apreciar en el esquema funcional, los módulos del Agente de Servicio pueden ser, según su propósito específico, clientes o servidores de otros módulos. También varía el tipo de comunicación en función al tipo de información que están manejando y a la necesidad de asegurar que la misma llegue en forma segura.

El módulo correspondiente al Agente de Usuario es especificado en el siguiente capítulo.



Inserción de un Documento al Sistema



El gráfico indica los pasos que sigue un documento cuando es ingresado al sistema y antes de que esté disponible para los usuarios interesados en él. El mismo camino se recorre cuando se produce la modificación de un documento ya existente.

Este esquema ejemplifica la forma en que los distintos componentes se relacionan, dentro de un mismo Agente de Servicio y entre diferentes Agentes de Servicio relacionados por el tema de interés del documento que se está actualizando.

Especificaciones de los Módulos

Envío de Ofertas

Este módulo del Agente de Servicio es el encargado de generar y enviar las ofertas de los documentos nuevos o actualizados a los clientes correspondientes. También envía los mensajes de confirmación (ACK) indicando que las ofertas fueron recibidas correctamente para los Agentes de Servicio proveedores, y los pedidos fuera de banda que son enviados a los proveedores, por la necesidad de actualización de algún documento especial.



Consta de dos módulos separados que en forma conjunta completan la funcionalidad para el manejo de las ofertas. Cada módulo tiene ports específicos para la transmisión y recepción de datos que pueden ser seteados por medio del archivo de configuración.

Toda la comunicación de este módulo se realiza por medio de datagramas. El envío de ofertas se realiza cuando un documento nuevo ingresa al sistema o se actualiza uno existente, y esta operación se repite hasta que el cliente destinatario de la oferta envía la confirmación (ACK) de que la oferta fue recibida correctamente.

Este módulo se comunica con el módulo de Recepción de Ofertas de los Agentes de Servicio que son clientes y que se encuentran activos, cada vez que corresponde enviar ofertas de documentos nuevos o actualizados.

Cada cierto tiempo, seteado por medio del archivo de configuración, también se ejecutan dos procesos, uno a continuación del otro, para generar y enviar las ofertas correspondientes.

El Primer Proceso es el que genera las ofertas en la tabla de ofertas: recorre el tabla de documentos (objetos), buscando aquellos documentos cuyo estado corresponde al de un nuevo documento o a un documento actualizado. De acuerdo al grupo de interés de cada objeto se obtienen los clientes que son consumidores o consumidores de meta información de ese tema, en la tabla de ofertas se debe registrar si se enviará todo el documento o sólo la meta información. La dirección IP del cliente se obtiene de la tabla de clientes.

Teniendo como referencia el mayor número de oferta generada, se generan las nuevas ofertas, registrando todos los datos correspondientes a dicha oferta. También se verifica si hay nuevos clientes a los que todavía no se le ha enviado ninguna de las ofertas de los documentos actuales.

También se verifica la tabla de pedidos fuera de banda, y para todos los pedidos que se recibieron y para los que todavía no se generó oferta, se registra la nueva oferta identificando a la misma para que sea tratada como una oferta generada en forma ordinaria. También se registra la actualización del estado de los pedidos fuera de banda para indicar que la oferta ya se generó.

El Segundo Proceso es el que efectivamente realiza el envío, por medio de datagramas, de las ofertas de documentos a los clientes activos. Este proceso verifica la tabla de ofertas generadas y envía tanto las nuevas ofertas como las que todavía no tienen confirmado que han sido recibidas por parte de los clientes. Este proceso también envía las confirmaciones a los proveedores de la recepción de las ofertas.

La información que se transmite cuando se envía una oferta incluye identificador de oferta, identificador de documento, dirección del servidor que generó el objeto, versión del objeto, identificador del grupo de interés, servidor que envía la oferta y tipo de oferta. Estos datos son imprescindibles para que el receptor pueda verificar la validez de la oferta e identificar el documento, y la versión para decidir si tiene que pedir el



documento. Recordemos que cada cliente puede tener mas de un proveedor del mismo tema (esta redundancia es necesaria porque no está garantizada la conectividad en la red) y pueden llegar ofertas del mismo documento desde diferentes proveedores.

Además se verifica el archivo de pedidos fuera de banda para enviarlos si es que hay nuevos pedidos generados y que deben ser atendidos. Estos pedidos en general provienen del módulo de Administrador de Usuarios, ya que cuando un usuario necesita un documento que por alguna causa no está en el Agente de Servicio se genera este pedido extraordinario.

Recepción de Ofertas

Este módulo del Agente de Servicio es el encargado de recibir las ofertas de los documentos nuevos o actualizados de los proveedores correspondientes. También recibe las confirmaciones (ACK) indicando que las ofertas fueron recibidas correctamente por los Agentes de Servicio Clientes, y los pedidos fuera de banda que son enviados por proveedores que necesitan la actualización de algún documento especial.

Este módulo está residente en el Agente de Servicio esperando por los mensajes que le envían los proveedores. Recibe y registra en la base de datos, luego de realizar las verificaciones correspondientes a versión, temas y origen de los documentos, las ofertas recibidas. En el caso especial en que la oferta ya existe, se registra en la base de datos la necesidad de enviar la confirmación de recepción de la misma.

Si se trata de un pedido fuera de banda, los datos que se reciben son: identificador del documento y dirección del servidor que lo generó. Luego de registrar este pedido especial se establece el estado como pendiente para que se genere la oferta correspondiente.

En el caso en que el mensaje recibido sea un ack de oferta, se registra la novedad en el archivo de ofertas, registrando que el cliente ya recibió la oferta para que la misma no se repita.

Administrador de Conexiones

Cada Agente de Servicio del sistema posee un registro de todos los clientes y mantiene el estado de actividad de los mismos. A través de mensajes especiales, cuyo único objeto es verificar los clientes, se determina si estos se encuentran en actividad o no, con el objeto de evitar enviar mensajes con ofertas, documentos, etc. que nunca podrán ser recibidos. Esto es necesario para la comunicación que se establece a través de datagramas, ya que la comunicación punto a punto chequea la actividad de los pares involucrados antes de establecer la comunicación.

El módulo se encarga de mantener actualizado el estado de actividad de los clientes y de la comunicación con los proveedores para indicar a estos que el Agente de Servicio está activo.



Consta de dos módulos para realizar la administración de Clientes y la comunicación con los Proveedores. Cada módulo tiene ports específicos para la transmisión y recepción de datos que pueden ser seteados por medio del archivo de configuración.

Módulo de envío

Envía los mensajes de respuesta a los proveedores y los requerimientos a los clientes del Agente de Servicio.

Cada cierto tiempo, seteados por el archivo de configuración, se realiza una consulta en la tabla de clientes y se envía un keep-alive (requerimiento para establecer si un servidor está activo) a cada cliente para saber si aún está recibiendo los mensajes. El mensaje consiste en una cabecera indicándole al cliente que es un mensaje de uno de sus proveedores y la dirección del servidor que envía el requerimiento.

También se recorre la tabla de proveedores para ver a que proveedor hay que enviarle una respuesta a un keep-alive enviado, para indicarle que está activo. Este mensaje solamente indica que se trata de un cliente respondiendo a un requerimiento de un proveedor.

Módulo de recepción

Tiene como función recibir los mensajes de los proveedores preguntando si el servidor se encuentra activo y de los clientes informando que se encuentran en actividad.

Cada mensaje que recibe este módulo está identificado en su cabecera indicando si se trata de un requerimiento (enviado por un proveedor) o de una respuesta (enviada por un cliente), además de la dirección del servidor que lo envía.

Si el mensaje corresponde a un cliente, indica que el cliente sigue activo, ya que el paquete es una respuesta a un keep-alive enviado como requerimiento por el servidor, entonces se actualiza el cliente como activo.

Si el mensaje corresponde a un proveedor, se toma la dirección del servidor que nos envía el mensaje del mensaje mismo y se registra dicha dirección en la tabla de proveedores, para enviarle una respuesta que llevará una indicación al proveedor que somos su cliente y que seguimos activos.

Otras consideraciones

El estado a partir del cual determinamos que un cliente está inactivo puede ser seteado en forma dinámica por el archivo de configuración en función a la seguridad de la red y los tiempos de demora de los mensajes.



Actualización de Objetos de Interés

La función de este módulo es mantener actualizados los documentos del Agente de Servicio para los temas de interés en los cuales existen usuarios finales interesados.

Los pedidos de actualización de documentos se pueden producir por la aparición de ofertas de documentos nuevos o de actualizaciones de versiones de documentos ya existentes. También puede ocurrir que se introduzcan usuarios o temas que requiere que se actualicen todos los documentos correspondientes a ese nuevo tema o usuario.

La comunicación de este módulo se realiza por medio de una conexión directa TCP/IP (ODP) ya que es una de las más delicadas del sistema y necesita que entre los dos servidores se establezca una conexión segura.

Este módulo, cada cierto tiempo, seteado por el archivo de configuración, verifica la tabla de ofertas comparándolos con la de documentos actuales, para determinar si se debe hacer un pedido de documento. Las ofertas que se piden son las que en cuyo estado se indica que el documento no fue transmitido, controlando que la versión sea mayor a la actual o que el documento no exista entre los documentos registrados en el Agente de Servicio.

Los datos necesarios para realizar el pedido se toman de las ofertas recibidas, ya que se debe referenciar una oferta recibida para establecer claramente con el proveedor de documentos a que objeto nos estamos refiriendo. Los datos necesarios para identificar la oferta recibida y que se transmiten en el pedido del objeto son: identificador de la oferta, identificador del objeto, dirección del servidor que generó el objeto, versión del objeto, tipo de oferta (para saber si se solicita el documento completo o sólo meta información).

Siempre se realiza la recepción de la meta información (información mínima que se puede disponer de un objeto), y si corresponde se recibe también la información, o sea el archivo físico. Durante y al finalizar la transmisión se realiza un chequeo de integridad del documento recibido.

Luego de la transmisión, el módulo actualiza la tabla de control de objetos para indicar que ya se dispone del archivo físico correspondiente al nuevo objeto, se debe grabar el archivo físico en el disco del servidor y registrar en la tabla de ofertas recibidas que ya se completó el pedido correspondiente a la misma.

Envíos de Objetos de Interés

Este módulo tiene como función el envío de los archivos físicos correspondientes a los documentos del sistema, como así también los datos correspondientes a la meta información de los documentos.



La comunicación de este módulo, se realiza por medio de una conexión directa TCP/IP (ODP) ya que es una de las más delicadas del sistema y necesita que entre los dos servidores se establezca una conexión segura.

El módulo que recibe pedidos, queda activo esperando por un requerimiento en un port determinado por el archivo de configuración, pero debe corresponderse con el que utiliza el cliente para realizar el pedido. Cuando un pedido es recibido por este módulo, se crea un thread (proceso especial que tiene toda la funcionalidad del módulo pero se crea especialmente y desaparece cuando termina la transmisión) que atiende el pedido, enviando la meta información y el documento si corresponde. El módulo principal que maneja los thread solamente espera por conexiones y cada vez que recibe una, activa el thread correspondiente y queda esperando una nueva conexión.

El primer mensaje del cliente tiene el número de oferta que se le envió. Se realiza la verificación con la oferta enviada, identificador del objeto y tipo de oferta. Si todas los datos de la solicitud son correctos se le envía una confirmación, luego el objeto y/o la meta información. Se registran estos datos en la tabla de ofertas enviadas para saber que la misma fue satisfecha.

Actualización de Clientes

Este módulo se encargará de la comunicación con el Agente de Ruteo. El mismo estará escuchando permanentemente si hay novedades enviadas por el Agente de Ruteo que informan los cambios efectuados en la topología de la red lógica.

Un evento importante ocurre cuando el Agente de Servicio actualiza uno de sus grupos de interés, informando explícitamente al Agente de Servicio el cambio y recibiendo del Agente de Ruteo la lista de los clientes para ese grupo de interés.

Este es el módulo que debe implementar el chequeo periódico con el Agente de Ruteo para saber si ambos continúan activos.

Administrador de Base de Datos

Este módulo se encarga de la comunicación de todos los módulos con la base de datos. En este caso han sido utilizados todos los servicios usuales que administran y utilizan los motores de bases de datos más comunes respetando la norma ANSI del lenguaje SQL standard.

Administrador de Usuarios

El administrador de usuarios en un módulo que se activa cuando se levanta la aplicación del Agente de Servicio. Su función es atender los pedidos de conexión que solicitan los clientes, lanzando un proceso, que atiende en forma exclusiva al cliente.



La conexión está activa hasta que el Agente de Usuario cierre la aplicación, es decir, que es el cliente quien solicita una finalización de la conexión. También, puede ocurrir que se corte la comunicación por otro motivo, pero en este caso no hay un pedido formal de fin de conexión.

El administrador de usuario reside en el servidor, y está siempre escuchando si algún cliente realiza una llamada. El agente de servicio puede atender a varios usuarios al mismo tiempo, existiendo de esta manera una escalabilidad horizontal. Cada vez que un cliente realiza una llamada, el servidor lanza un proceso “thread” que es el encargado de atender en forma exclusiva a ese usuario.

El servidor intenta leer un archivo de configuración donde lee variables que indican donde se encuentran los documentos físicamente en el servidor, el tamaño de la trama para los mensajes que se transmiten entre el cliente y servidor. Si por algún motivo no se puede leer este archivo de configuración, no se establece la conexión, avisándole al usuario de tal situación. Si todo anda bien, el servidor se queda a la espera de un pedido de *inicio de conexión* solicitado por el cliente.

Una vez que se establece un *pedido de conexión*, el cliente envía un pedido de *tamaño de la trama*. El mismo se utiliza para determinar el tamaño de la trama de los mensajes que se envían entre el cliente y el servidor.

Para poder ingresar al sistema, el usuario debe ingresar su nombre y password, y éstas son validadas por el servidor. Para ello, el cliente envía un pedido de *control de password* y el servidor chequea los datos enviados por el cliente contra la base de datos, y le envía el resultado al cliente. Si la respuesta es correcta, el servidor responde con el número de usuario, y si la misma falla, le responde con el valor 0.

El cliente solicita un pedido de *armado del treeview*, que se utiliza para armar dinámicamente las operaciones que puede realizar el usuario sobre los documentos, que esta dado según su perfil. El servidor al recibir este pedido arma un Query dinámico con los datos pasados como parámetros, como la cantidad de días de novedad que tendrán los documentos, el id del usuario y la dirección del servidor. Este Query está compuesto en 6 partes, uno hace referencia a la carpeta de *Alta* donde se cargan los nombres de los grupos de los cuales el usuario es productor, pudiendo el usuario no tener carpeta de *Alta* si es que el mismo no es productor de ningún grupo. En la carpeta correspondiente a *Baja* se hace referencia a todos los documentos creados por el usuario. Lo mismo sucede con respecto a la de modificación. Las otras tres partes que siguen corresponden a la carpeta de *Consulta*. La primera hace referencia a los documentos que por su fecha de actualización corresponden a la carpeta de *Novedades*, es decir, contiene las ultimas novedades existentes en el servidor. La segunda parte hace referencia a los documentos que el usuario puede acceder para consultar cuando éstos no fueron creados por el usuario. Por último, están todos los documentos creados por el propio usuario. Una vez resuelto el Query por el servidor, le envía el resultado a cliente, y éste se encargará de cargarlos en la aplicación del usuario.



Cuando el servidor recibe un pedido de *“fin de conexión”* por parte del cliente, corta la comunicación matando el proceso “thread” que atiende al cliente en forma exclusiva, enviando previamente el OK de fin de conexión al cliente.

Cuando el usuario está realizando una operación de alta, el cliente envía un pedido *solicitando los objetos que pertenecen la mismos grupo*, enviando como parámetro el nombre del grupo al cual pertenece el documento a dar de alta. El servidor recibe este requerimiento y arma una sentencia SQL consultando los nombres de los otros documentos que pertenecen al mismo grupo y el servidor envía el resultado al cliente. El cliente usa esta información para que el usuario pueda vincular el nuevo documento a crear con los ya existentes. Cuando el usuario presiona el botón de alta, el cliente envía un pedido de *alta de documento*. El servidor arma una instrucción SQL con los datos recibidos y la ejecuta, dando de alta el documento en la base de datos y enviando el resultado de la operación al cliente y éste chequea el valor, y si todo anduvo bien, el cliente le envía el documento físico al servidor, realizando para un ello un pedido de *ingreso de documento físico*; donde el servidor almacena dicho archivo cuyo nombre está formado por la dirección del servidor + “_” + id del objeto. El servidor almacena el documento físico según en parámetro que el servidor toma cuando arranca el agente de servicio. Este parámetro se llama “remote_directory” que se encuentra en el archivo de configuración OpenConfig.ini.

Dado un requerimiento por parte del cliente de un pedido de *baja de documento*, el servidor arma una sentencia SQL y la ejecuta para poder dar de baja el documento, pero lo que hace realmente, es sólo agregar una fecha en el campo fecha_baja de la tabla objeto, enviándole el resultado de la operación al cliente. Para poder recuperar documentos dados de baja, sólo lo puede hacer el *administrador* poniendo en blanco la fecha_baja de la tabla objeto.

Cuando el cliente envía al servidor un pedido de *modificación de documento*, éste arma una instrucción SQL y la ejecuta, actualizando todos los datos modificados por el usuario y le devuelve el resultado de la operación al cliente. Si la operación es exitosa y el usuario actualizó el documento físico, entonces el cliente envía un pedido de *actualización del documento físico*, y el servidor se encargará de reemplazar el existente, devolviéndole al cliente el resultado de toda la operación.

Cuando el servidor recibe un pedido de *extracción de documento físico*, éste busca el documento solicitado y le envía el documento físico al cliente y éste se encarga de almacenarlo en el lugar especificado por el usuario oportunamente.

En una operación de solicitud del documento físico, puede ocurrir que el documento no esté disponible por algún problema en el servidor o no se pueda extraer. En este caso, le envía al cliente tal novedad y este último se encargará de avisarle al usuario de tal inconveniente, dándole el cliente la oportunidad al usuario que realice un pedido *fuera de banda*, es decir, sí el usuario lo desea, genera un pedido especial del documento físico, y luego el servidor se encargará de realizar el pedido correspondiente al servidor padre, quien fue que el que envió el documento original si es que el mismo no pertenece al servidor donde se conecta el usuario. Si el documento faltante pertenece al servidor donde se conecta el usuario, entonces al realizarse un pedido *fuera de banda*



hará que cuando ingrese al sistema el dueño del documento se le avisará que dicho documento está faltante y se le pedirá que lo actualice. De esta forma el usuario que originó el pedido fuera de banda podrá consultar ese documento en otro momento. En caso que el propio dueño de un documento quiera extraer el documento físico y por algún motivo no lo puede hacer el servidor, se le avisará de tal situación y se le pedirá al usuario que actualice el mismo, no generando un pedido *fuera de banda*.



Especificaciones de Protocolos

Envío de Ofertas → Recepción de Ofertas
• Ofertas ("F", id_oferta, id_objeto, dir_server, version_obj, id_gi, server_oferta, tipo_of)
• Of_recibidas ("A", id_oferta).
• Fuera_de_Banda ("P", id_objeto, dir_server).

Administrador de conexiones → Administrador de conexiones
• Requerimiento ("P", dir_server)
• Respuesta ("C", dir_server)

Actualización de Objetos → Envío de Objetos	Envío de Objetos → Actualización de Objetos
• Pedido_Objetos (id_oferta, id_objeto, dir_server, server_oferta, tipo_of)	• Ok_pedido_objeto ("OK", id_oferta)
	• Envio_MetaInfo (titulo, version, id_grupo, fecha_baja, fecha_creacion, fecha_recepcion, tipo_doc, extension, autor, coautores, comentarios, abstract, doc_asociados)
•	• Envio_ObjetoFísico(id_objeto, dir_server, objeto_físico)

Agente de Usuario → Agente de Servicio	Agente de Servicio → Agente de Usuario
• Pedido_Conexion ("IC")	
• Pedido_Trama ("TR")	• Respuesta_Trama (Tamaño_trama)
• Ingreso_Usuario ("PA", usuario, password)	• Respuesta_Ingreso (user_id)
• Perfil_Usuario ("TV", user_id, dias_novedad, dir_server)	• Respuesta_Perfil (lista < id_grupo, nombre_grupo, titulo, id_objeto, dir_server, id_owner, tipo .>)
• Alta_Doc ("IN", meta_información, objeto_físico)	• Resultado_Alta (codigo_resultado)
• Baja_Doc ("DE", id_objeto, id_grupo, dir_server)	• Resultado_Baja (codigo_resultado)
• Modificacion_Doc ("UP", meta_información, objeto_físico)	• Resultado_Modificacion (codigo_resultado)
• Consulta_Doc ("RD", id_user, id_objeto, id_grupo, dir_server)	• Envio_Datos (meta_informacion, objeto_físico)
• Pedido_Fuera_Banda ("FB", id_objeto, dir_server)	• Resultado_Fuera_Banda (codigo_resultado)
• Fin_Conexion ("FC")	



Conclusiones

Consideramos que la división en módulos hace que el sistema sea a la vez flexible y robusto ya que tanto el diseño de la forma en que interactúan los módulos como el mecanismo de recuperación de módulos caídos hace que el sistema tenga una degradación gradual de su funcionamiento y la posibilidad de recuperar o cambiar los módulos caídos del sistema.

Cada módulo tiene un esquema propio de comunicaciones de acuerdo al nivel de confiabilidad que se necesita por su función. Esta comunicación se realiza con otros módulos de los Agentes de Servicio con los cuales se está comunicando ya que la relación con los procesos propios de Agente de Servicio al que pertenece se realiza a través de la base de datos.

Dentro de cada Agente de Servicio los módulos interactúan para el objetivo final del sistema que es el ingreso y transmisión de documentos a través de la red en forma óptima y transparente para los usuarios. Esto significa que la funcionalidad está repartida en los diferentes módulos que componen el sistema.

Se considera en el sistema que es necesario mantener un control de los clientes activos para optimizar las comunicaciones. De esta manera, cada Agente de Servicio sabe que clientes están activos y esperando datos y cuales no, evitando de esta manera enviar a través de la red paquetes de información que sabemos que no llegarán a destino.

Podemos mencionar como parte de la flexibilidad que tiene el sistema, el mecanismo excepcional para pedir documentos perdidos, esto es considerado en el sistema como un pedido fuera de banda. Se utiliza este método en los casos en que un Agente de Servicio haya perdido en forma accidental o por algún problema del sistema operativo alguno de los documentos físicos que debería poseer. Dentro de la información que se almacena de cada objeto se mantiene el último Agente de Servicio que tiene el documento físico para poder implementar esta funcionalidad.

El esquema de los documentos permite que la meta información siempre pueda ser consultada ya que puede ser transmitida en forma independiente del cuerpo principal y si la información de esta parte del documento interesa a los usuarios, se puede pedir el documento en sí mismo. De alguna forma optimiza el tránsito en la red y evita que se almacene información que no se utiliza en los Agentes de Usuario.

Las facilidades de incluir los documentos compuestos es otra de las características más destacables de la implementación. Estos documentos, compuestos por otros documentos, que a su vez también pueden ser compuestos, permite la incorporación de documentos tipo libro con capítulos. Los capítulos pueden ser actualizados o agregados sin necesidad de editar nuevamente todo el libro, esto optimiza el tráfico y almacenamiento.

Con la creación del módulo Administración de Usuarios, se logra seguridad con respecto al acceso a la Base de Datos, ya que el mismo es el único módulo que



interactúa con el cliente. A su vez, si en algún momento se decide cambiar el motor de base de datos, no habrá problemas siempre y cuando, se respeten los formatos de los mensajes entre el administrador de usuario y el Agente de Usuario, por lo cual, cualquier modificación que se realice en la base de datos será transparente para el usuario.

El Administración de Usuarios tiene la virtud de atender simultáneamente a varios usuarios, por medio del lanzamiento de un proceso Thread, que atiende en forma exclusiva a cada cliente que solicita un pedido de conexión con el servidor, estableciendo para ello, una conexión punto a punto entre el cliente y el servidor.



Capítulo 6

Desarrollo del Agente de Usuario

El agente de usuario es un módulo que se encarga de interactuar con el usuario. Por medio del mismo el usuario puede realizar todas las operaciones válidas sobre los documentos. A su vez, éste módulo se comunica con el Agente de Servicio, quien se encarga de realizar las operaciones solicitadas por el Agente de Usuario. El usuario puede a través de este módulo dar de alta documentos pertenecientes a los grupos de los cuales el usuario es productor, dar de baja y modificar documentos propios y consultar todos los documentos que pertenecen a los grupos donde el usuario es consumidor de información o de meta información.

En cada documento se indica el grupo al cual pertenece el documento, título, autor, fecha de creación y resumen del mismo, siendo estos datos de carácter obligatorio, en tanto que comentario, documentos vinculados, y coautores son de carácter optativo.

Desarrollo

Nuestro objetivo primordial era, cuando se decidió implementar ODS, realizar sólo los Agentes de Servicio, pero para poder ingresar los datos de los documentos a la base de datos para luego ser distribuidos entre los distintos Agentes de Servicio, se lo debe hacer directamente sobre la base de datos provocando no sólo un ingreso engorroso sino que además se perdía la seguridad de la base de datos, dado que varios usuarios podrían ingresar para efectuar modificaciones a sus propios documentos causando pérdida de información o daño en la misma. Decidimos implementar el Agente de Usuario como medio por el cual el usuario puede ingresar sus documentos, haciendo de esta forma que el acceso a los documentos existentes en el Agente de Servicio sea sencilla, amigable y segura, ya que éste representa el único medio para acceder a la información.

El Agente de Usuario es la interfase que le permite al usuario realizar operaciones sobre documentos. Este se encarga de comunicarse con el Agente de Servicio en forma transparente para el usuario, es decir, el mismo no tiene porque conocer donde reside el Agente de Servicio, logrando así una transparencia de localización.

El Agente de Usuario funciona como un cliente en un sistema Cliente-Servidor. El mismo envía requerimientos al Servidor (Agente de Servicio), éste recibe el mensaje, procesa el mismo (en su mayoría sentencias SQL – Structured Query Language-) y devuelve el resultado de la operación al Cliente (Agente de Usuario). Entre el cliente y el servidor se establece una comunicación punto a punto.

La conexión está activa hasta que el Agente de Usuario cierre la aplicación. En algunos casos, puede llegar a ocurrir que se corte la comunicación mientras el usuario tiene la aplicación abierta; en este caso, cuando el usuario intente realizar un



requerimiento al servidor, el cliente intenta restablecer la comunicación y si esto es factible, todo sigue normalmente sin que el usuario se haya enterado de la situación. Si la misma no se logró restablecer, se le avisa al usuario de tal situación y el mismo deberá intentar conectarse más tarde.

El Agente de Usuario está desarrollado para poder ser ejecutado sobre un ambiente Windows 95 / NT.

Cuando arranca la aplicación, el cliente solicita un *pedido de inicio de conexión*. Luego, el usuario debe ingresar su nombre y password, y éstas son validadas por el servidor. Para ello el cliente envía un pedido de *control de password*. En caso de que el nombre del usuario y la password no fueran válidos, se le avisará al usuario que se ha denegado el acceso y el mismo puede volver a intentarlo nuevamente, con un máximo de tres intentos, llegado a esto, se sale de la aplicación.

A continuación el cliente envía un pedido de *tamaño de la trama*. El mismo se utiliza para determinar el tamaño de la trama de los mensajes que se envían entre el cliente y el servidor. Luego, el cliente solicita un pedido de *armado del treeview*, que se utiliza para armar dinámicamente las operaciones que puede realizar el usuario sobre los documentos, que esta dado según su perfil. El perfil lo asigna el administrador del sistema. Una vez resuelto el requerimiento por el servidor, le envía el resultado al cliente. A medida que el cliente va recibiendo la información va armando el TreeView y a su vez almacena parte de esta información en una matriz que se utiliza cuando el usuario desea traer los datos del documento a pantalla. Cabe recordar, que de acuerdo al perfil del usuario y a los documentos disponibles en ese momento en el servidor, se arman las carpetas, pudiendo en algún momento no tener todas las carpetas o documentos posibles.

El usuario puede dar de alta documentos que pertenezcan a los grupos que tiene habilitados para crear documentos, es decir, si es productor del grupo. Sólo puede dar de baja documentos creados por él. Al dar de baja el documento, el mismo deja de estar disponible tanto para el propio dueño como para los demás usuarios que son consumidores. De la misma forma, sólo puede modificar documentos creados por él. Al modificar el documento, el mismo aparecerá como novedad para los usuarios que son consumidores.

Las consultas que puede realizar el usuario son de documentos pertenecientes a determinados grupos, a los cuales está habilitado como consumidor de información, consumidor de meta información o productor. Los objetos que están en la carpeta de "Novedades" son aquellos que fueron creados o modificados dentro de un determinado tiempo. Este tiempo se toma de un archivo de parámetros que se lee cuando se abre la aplicación, e indica la cantidad de días que puede tener el documento desde su creación o modificación. Si la fecha de creación o modificación con respecto a la actual (fecha del servidor) está dentro de los días indicados en el archivo de configuración, el documento será mostrado en la carpeta "Novedades", dentro del grupo al cual pertenece el documento. Estos documentos pueden haber sido creados o modificados por otros usuarios que pertenecen al mismo servidor donde se conecta el usuario ingresado o a otro servidor.



En la carpeta “*Objetos*” están todos los documentos a los cuales el usuario tiene acceso para consulta y que no son propios. Aquí también aparecen los que figuran en la carpeta de “*Novedades*” y que no son propios.

En la carpeta “*Objetos Propios*”, figuran todos los documentos creados por el propio usuario. En la opción de consulta sólo se puede extraer el documento físico si es consumidor de información o productor y ver todos los datos existentes en los respectivos “*tabs*”. Cabe aclarar que el usuario que es consumidor de meta información no puede extraer el documento físico.

Mientras el usuario realiza operaciones, otros usuarios pueden actualizar sus propias documentos, y estos cambios no se ven reflejados en el “*TreeView*” del usuario, para evitar esto hay una opción dentro del menú (File – Menú Refresh) que permite reflejar las novedades existentes en el servidor. Cuando un usuario da de alta, baja o modifica el título de un documento, no se refleja en el “*TreeView*”, para ver estos cambios se debe refrescar el “*TreeView*”.

Para salir del sistema se lo puede hacer desde el menú (File – Exit) o dando un click en el extremo superior derecho (X). En este caso el cliente envía al servidor un pedido de “*fin de conexión*”, que hace que el servidor corte la comunicación, enviando previamente el OK de fin de conexión, y cuando el cliente recibe éste, cierra la aplicación del usuario.

Alta de Documentos

Para proceder a dar de alta un documento, se debe ingresar en la carpeta “Alta” y elegir el grupo al cual pertenece el documento a dar de alta. En ese momento el cliente envía un mensaje al servidor para solicitarle un *pedido de los objetos que pertenecen al mismo grupo*, y el servidor le responde con los nombres de los otros objetos que pertenecen al mismo grupo, siendo estos cargados en la tabla de *documentos disponibles* y que luego el usuario podrá usar si así lo desea para vincular al documento creado por él con estos, como se describe más adelante.

Para la opción de alta es necesario ingresar un título, un autor, la versión, la extensión, el abstract y el documento físico en sí. Los datos de los co-autores y comentarios (comments) son opcionales. El documento a ingresar puede tener otros documentos que están vinculados a él. Para ello se debe elegir el “*tab*” “*Documents Compounds*”, en el cual aparecen dos tablas, una que corresponde a los documentos disponibles y otra a los documentos vinculados al documento a dar de alta. En el primero figuran todos aquellos documentos que pertenecen al grupo al cual se quiere dar de alta el documento. Puede ocurrir que dicha ventana este vacía, y esto es porque el servidor no posee documentos que pertenezcan a ese grupo. Para el caso que ya existan documentos, y se quiere seleccionar uno de ellos, se debe elegir el documento, haciendo que el nombre del documento desaparezca de dicha tabla y aparezca en la tabla de documentos ingresados, quedando de esta manera el documento vinculado. Para el caso de querer desasociar un documento, se debe seleccionar el mismo, haciendo que



desaparezca el nombre del documento de la tabla de documentos vinculados y que aparezca en la tabla de documentos disponibles. Se debe aclarar que un documento puede estar vinculado a varios documentos (siempre perteneciente al mismo grupo), realizando el procedimiento descrito arriba para cada documento al cual se lo desea vincular.

Para ingresar físicamente el documento, se debe presionar el botón consulta y aparecerá un menú de diálogo que le permite al usuario buscar o localizar el documento a dar de alta. Para ejecutar la operación de alta se debe presionar al botón para este fin, el cual validará que estén todos datos requeridos como obligatorios y en caso de faltar alguno se le avisará al usuario los datos faltantes, posicionándose luego el cursor en el dato faltante. Una vez completado este paso, el cliente arma una cadena con toda la información ingresada por el usuario, y ésta se envía al servidor solicitando un pedido de *alta de documento*. El servidor ejecuta el requerimiento enviando el resultado de la operación al cliente y éste chequea el valor, y si todo anduvo bien le envía el documento físico al servidor realizando para un pedido de *ingreso de documento físico*; donde el servidor almacena dicho archivo. Luego de completada esta operación de deshabilitan los botones de alta y consulta. Hay que tener en cuenta que el documento dado de alta no aparecerá en el TreeView hasta que el usuario no refresque el mismo desde la opción del menú.

Baja de Documentos

Para proceder a dar de baja un documento, se debe ir a la carpeta de baja y elegir el documento. El cliente envía al servidor una solicitud de *requerimientos de los datos*, el servidor le devuelve el resultado y el cliente es el encargado de cargar los datos en los campos correspondientes de la pantalla. Luego, el cliente solicita al servidor un requerimiento de *objetos vinculados*, éste le envía el resultado al cliente, pudiendo ser vacía la información recibida por el cliente si es que el documento no tiene ningún documento vinculado, el cliente los carga en la tabla de *documentos ingresados*, y por último el cliente realiza otro requerimiento solicitando los *documentos que pertenecen al mismo grupo*, el servidor ejecuta dicho requerimiento y le devuelve el resultado al cliente, pudiendo ser nula la información si aún no existen documentos en ese grupo, el cliente los carga en la tabla *documentos disponibles*.

En la opción de baja toda modificación que se haga no será tenida en cuenta. Se puede extraer el documento si es que el usuario así lo desea. Para darlo de baja se debe presionar botón para tal fin, el cual dará de baja realmente el documento para el usuario. El cliente envía en mensaje al servidor de solicitud de *baja de documento*. El servidor envía el resultado de esta operación informándole del mismo al usuario. Hay que recordar que este documento no desaparece del TreeView hasta que se refresque el mismo desde el menú.



Modificación de Documentos

Para proceder a modificar un documento, se debe elegir uno de la misma manera que se describió para el caso de la baja, pero trabajando sobre la carpeta “Modificación”. Se pueden modificar todos los datos del documento, vincular otros documentos, el documento físico, etc.

La versión se debe modificar, pues con esto se indica que el documento ha sido modificado y deben ser enviado a los servidores clientes. Para realizar realmente la operación de modificación se debe presionar el botón para este fin. En esta opción también se validará que estén completos los datos obligatorios. El cliente envía al servidor un pedido de *modificación de documento*, éste ejecuta el requerimiento devolviendo el resultado al cliente y si todo anduvo bien, y el usuario actualizó el documento físico, entonces el cliente envía un pedido de *actualización del documento físico*, y el servidor se encargará de reemplazar el existente, devolviéndole al cliente el resultado de toda la operación.

Consulta de Documentos

Para realizar una opción de consulta, se debe ingresar en la carpeta de consulta, y elegir el documento a consultar. Si el usuario es consumidor de información del grupo al cual pertenece el documento elegido, no podrá extraer el documento físico para consultarlo, pero si podrá hacerlo si es consumidor de información o productor del grupo. En esta opción toda modificación que hace el usuario no tendrá validez.

Para la extracción del documento el cliente solicita al servidor un pedido de *extracción de documento físico*, el cual le envía el documento y el cliente se encarga de almacenarlo en el lugar especificado por el usuario oportunamente. En caso que al realizar el pedido del documento físico este no esté disponible por algún motivo o no puede ser leído, se le avisará al usuario de tal situación y se le dará la oportunidad de realizar el pedido *fuera de banda*, es decir, sí el usuario lo desea, genera un pedido especial del documento físico, y luego el servidor se encargará de realizar el pedido correspondiente al servidor padre, quien fue que el que envió el documento original si es que el mismo no pertenece al servidor donde se conecta el usuario. Si el documento faltante pertenece al servidor donde se conecta el usuario, entonces el realizar un pedido *fuera de banda* hará que cuando ingrese al sistema el dueño del documento se le avisará que dicho documento está faltante y se le pedirá que lo actualice. De esta forma, el usuario que originó el pedido fuera de banda podrá consultar ese documento en otro momento. En caso que el dueño de un documento quiera extraer el documento físico y por algún motivo no lo puede hacer el servidor, se le avisará de tal situación y se le pedirá al usuario que actualice el mismo, no generando un pedido *fuera de banda*.

Conclusiones

Con el Agente de Usuario se puede conectar al Agente de Servicio en forma transparente para el usuario, ya que se configura en un archivo de configuración



llamado *OpenConfig.ini* la dirección del Agente de Servicio y por lo tanto cada vez que el usuario abre la aplicación se establece una conexión punto a punto entre ambos módulos. Esta conexión queda establecida hasta que el usuario salga de la aplicación.

Para ejecutar el Agente de Usuario, el usuario debe tener instalado un intérprete Java ya que este módulo no es un ejecutable por sí mismo sino que es un bytecode en lenguaje Java.

Todas las operaciones sobre los documentos se deben hacer a través de este módulo. Los documentos a los cuales tiene acceso, son los que están asignados según su perfil, tanto para actualizar como para consultar. Cada vez que el usuario se conecta, o actualiza el TreeView, puede consultar todos los documentos existentes en el servidor al cual se conecta el usuario, ya sean pertenecientes a dicho servidor o pertenecientes a otros servidores.



Conclusiones

Se desarrolla en esta parte de la tesis la conclusión sobre los aspectos fundamentales de este trabajo. Se pone el énfasis en la investigación realizada sobre las arquitecturas y lenguajes y su aplicación práctica en el sistema de transmisión de documentos.

Sobre las Arquitecturas (CORBA, DCOM y RMI-JDK)

Básicamente son arquitecturas muy parecidas en tanto sirven para la implementación de comunicaciones entre componentes de una red, permitiendo desarrollar aplicaciones que soporten esquemas cliente-servidor, todas tienen una funcionalidad parecida, aunque la implementen de diferente manera.

Las principales diferencias pueden verse conceptualmente en que tanto CORBA como DCOM integran elementos heterogéneos de software, a través de objetos comunes o interfaces de comunicaciones. En tanto, RMI-JDK brinda soluciones parecidas de conectividad y manejo de recursos compartidos por medio de un lenguaje nuevo y propietario, Java, que corre sobre diferentes sistemas operativos.

Podemos decir que no hay una arquitectura que se distinga como la mejor en forma contundente sobre las demás, cada una cubre diferentes necesidades y requerimientos:

- **CORBA** se destaca cuando se necesita comunicación entre empresas.
- **DCOM** es lo más apropiado para conexión de computadoras personales.
- **JDK-RMI** aparece como el más indicado en los desarrollos de Internet.

Sobre los Lenguajes (JAVA y C++)

Podemos decir que el tiempo y la aceptación de C++ en el mercado todavía representan una supremacía que Java no ha podido superar, a pesar de las expectativas que se tienen sobre el uso masivo del lenguaje. Creemos que Java tiene mejores perspectivas en los desarrollos de aplicaciones en Internet.

La mayor desventaja de Java es la performance de ejecución, dado que en la mayoría de los casos C++ ejecuta los programas en un tiempo considerablemente menor.

Lo mas destacado de Java es la implementación de la filosofía “write one, run anywhere”. Se puede compilar un programa Java en una plataforma que contenga un compilador Java y luego ser ejecutado en cualquier otra máquina virtual Java.



Sistema Implementado

El sistema que se implementó corresponde a un subconjunto de las especificaciones del sistema ODS [RNL 97] sobre distribución de objetos en una red de área amplia.

La implementación incluye el diseño detallado y la programación del Agente de Servicio que es el componente básico del Sistema de Distribución de Documentos y de un Agente Usuario para brindar a los usuarios un front end.

Esta implementación parece adecuada ya que nos permitió verificar en forma práctica la funcionalidad de JDK (Java Development Kit) para este tipo de desarrollo. A la vez, pudimos implementar una hipótesis de distribución de documentos desarrollada para Internet por medio de las formulaciones en ODS [RNL 97]. Queda fuera del alcance de la presente tesis las conclusiones definitivas en cuanto al grado de cumplimiento y efectividad del sistema ODS.

Logros más Significativos

Dentro de las experiencias recogidas durante el desarrollo y la investigación, se puede destacar el buen entendimiento de las diferentes arquitecturas disponibles para desarrollo de sistemas distribuidos. La comparación deja en claro las similitudes y diferencias que existen y una proyección de lo que puede deparar el futuro para cada una, como así también cuales son sus puntos fuertes y débiles, concluyendo las áreas donde cada una puede predominar sobre las demás.

También el análisis detallado de las características de Java y C++, es uno de los objetivos logrados y que hacen un verdadero aporte en esta materia, ya que es muy profusa la literatura indicando las similitudes, pero una verdadera comparación de estos lenguajes deja muy en claro el alcance de cada uno y las diferencias conceptuales desde el punto de vista del diseño de aplicaciones que poseen.

El desarrollo de un sistema Cliente – Servidor para Internet, implementado enteramente en Java representa otro de los logros que podemos citar. Creemos que la habilidad y los conocimientos necesarios y adquiridos durante esta implementación nos dejan una muy buena experiencia, en tanto que ejercitan las habilidades para resolver problemas de desarrollo con las limitadas herramientas de un lenguaje nuevo de programación.

Con la implementación del prototipo pudimos aplicar parte de los conocimientos adquiridos en la investigación de arquitecturas y lenguajes. Logramos complementarlo con nuestro aporte personal que se ve reflejado a través del diseño detallado de la especificación ODS [RNL 97] y la programación del sistema. Pudimos hacer realidad un sistema concreto para compartir publicaciones en Internet, que puede ser utilizado en forma práctica por grupos con este tipo de necesidades.



Pudimos comprobar que el sistema resultó robusto y confiable siendo una aproximación adecuada para distribución de documentos sin alto grado de actualización. La ventaja de acceder a copias locales de los documentos disminuye el tráfico en la red, ya que el documento original se actualiza en los servidores locales solamente si hay usuarios interesados en el documento y el mismo fué modificado, y la transmisión se realiza en forma automática.

Sobre el desarrollo técnico realizado podemos destacar que al modelo cliente servidor utilizado es el adecuado para este sistema, permitiendo la posibilidad de intercambiar componentes de software en forma flexible. Esto incluye la capacidad de conectar Agentes de Usuario o componentes de los Agentes de Servicio nuevos o con distinta funcionalidad, siempre que se respete el protocolo de comunicaciones establecido.

La programación en lenguaje Java nos permite ejecutar el código generado en diferentes plataformas que posean la máquina virtual de Java. Otras de las características positivas de Java son que en general necesita menos líneas de código que otros lenguajes, tiene un modelo de manejo automático de memoria, facilidad en la administración de errores y la posibilidad de convertir funciones en applets para ser ejecutados en Browser de la Web.

De las deficiencias encontradas en Java podemos mencionar lentitud en la ejecución, falta de facilidades gráficas propias, entorno nativo de programación DOS y falta de documentación.

Haber elegido un lenguaje que luego se consolidó como uno de los más utilizados en ambientes de redes de área amplia, especialmente en Internet, también puede destacarse como un acierto al momento de definir las herramientas a probar.

Otras Vías por Donde Seguir Desarrollando

Una alternativa sería crear un sistema más complejo de distribución de documentos con la inclusión de roles de supervisor, redactor, redes locales de documentos sin editar. Esto por un lado aumentaría la complejidad de la distribución pero garantizaría que el documento puesto en la red tenga toda su información validada, dado que la distribución desarrollada en la presente tesis no lo garantiza, porque el documento creado por el usuario no es controlado por nadie.

Otro de los temas que puede ser desarrollado con mas profundidad es la consideración de nuevas alternativas y herramientas para la implementación de sistemas basados en Internet. Las arquitecturas analizadas son de las mas conocidas y usadas en este tipo de desarrollo, pero están apareciendo alternativas que pueden ser consideradas con gran potencial en el futuro y pueden ser también objetos de investigación y estudio.

Una de las alternativas que pueden llegar a ser implementadas sin demasiados cambios respecto del sistema actual es la inclusión de Agentes de Servicio y Agentes de Usuario, que puedan ser ejecutados directamente a través de un Browser, utilizando



applets de Java. Otra opción complementaria a esta es la de unificar los Agentes de Servicio y Usuario en una misma aplicación

Extender la funcionalidad de documentos a otros objetos en la red es también uno de los posibles caminos para continuar con el desarrollo de este trabajo.



Glosario

Applet: es una pequeña aplicación o programa escrito en lenguaje Java que será ejecutado cuando se invoque la página web donde reside.

Arquitectura: para un sistema de información es el esquema que define como todas las piezas del sistema de computación interactúan y trabajan. No se refiere en particular a ningún proceso, pero sí a la forma en que se relacionan los procesos en un sistema.

Browser: la más importante pieza de software de los últimos años, permite la navegación por la World Wide Web, siendo los mas conocidos Netscape e Internet Explorer. Permite la interpretación de páginas HTML, que es el lenguaje común de Internet.

Cliente/Servidor: sistema informático basado en una aplicación que funciona como cliente, la cual requiere servicios de otra aplicación que funciona como servidor y ofrece esos servicios.

CORBA (Common Object Request Broker Architecture): es un conjunto de definiciones de cómo los objetos deben interactuar sobre la red usando objetos ORB.

Datagrama: paquete de información que, a diferencia de IP o de IPX, no está orientado a la conexión. Un datagrama contiene información de encabezamientos (headers) que proporcionan a los encaminadores (routers) las direcciones de destino imprescindibles para su distribución.

DCOM (Distributed Component Object Model): es una arquitectura definida por Microsoft. Actualmente solo está disponible para plataformas propietarias de Microsoft.

Duplex: Comunicación de datos que se mantiene bidireccional y simultáneamente. Más conocido por la denominación: 'full duplex' y, técnicamente, por duplex integral.

Encapsulamiento: define para cada objeto como se empaquetan los datos y procesos dentro de una interfase para permitir que los detalles de implementación estén ocultos y protegidos de posibles accesos indebidos.

Front End: parte de un desarrollo de software que será utilizado por usuarios finales y que sirve para comunicar, usualmente en forma amigable, a los sistemas con dichos usuarios.

FTP: File Transfer Protocol. Uno de los servicios que brinda Internet y que es utilizado para la transferencia de archivos binarios o de texto. Utilizando FTP se pueden bajar archivos de una computadora que se encuentra en un lugar remoto, también se pueden enviar archivos a otra computadora remota desde la propia.

Gateway: computadora o sistema que transfiere datos entre sistemas o redes que normalmente son incompatibles.



GUI: (Graphic User Interface) Interfase gráfica por medio de la cual los usuarios pueden interactuar con las computadoras en forma más amigable.

Herencia: define la relación entre clases, permitiendo que puedan utilizarse parte del código existente, a través de adquirir datos y propiedades, una subclase de la clase de la cual depende o hereda.

HTML: Hyper Text Markup Language. Es el lenguaje con el cual se escriben la mayoría de las páginas o documentos en la WWW.

IOP (Internet Interoperable ORB Protocol): es un subconjunto de CORBA y provee un camino estándar para comunicaciones ORB. Es un estándar de hecho para comunicaciones inter objetos utilizando protocolos TPC/IP.

IDL (Interface Definition Language): es una de las partes claves de CORBA, se compone del lenguaje para realizar las definiciones de las interfaces de los objetos.

Infraestructura: parte de un sistema de computación que provee una función específica en el sistema donde opera. Típicamente el 70% de una aplicación comprende su infraestructura (Por ejemplo como se acceden los datos, como es la comunicación, etc.).

Internet: la red más grande del mundo y en constante expansión. El conjunto de computadoras que conforman la red utilizan el protocolo IP preferentemente. Internet provee servicios como login remoto, ftp, correo electrónico, telnet, news y muchos más.

IP: Internet Protocol. Es el más importantes de los protocolos sobre los que descansa Internet. Este protocolo permite que los paquetes de información atraviesen las distintas redes que se encuentran en camino hacia su destino final.

Java OS: un programa Java puede correr en cualquier computadora que soporta el Java runtime, con el código común provisto por un compilador Java.

Keep- alive: mensaje codificado muy pequeño que se utiliza normalmente en sistemas de comunicaciones entre procesos o servidores para verificar que ambos componentes continúan en contacto.

Lenguaje Java: está basado en un subconjunto simplificado de C++, es orientado a objetos y está definido para múltiples plataformas, posee seguridad integrada y otros mecanismos que lo hacen mas apropiado para la programación en ambientes distribuidos.

Modelo de Objetos: define las interfaces y protocolos que manejan los componentes de un modelo que soporta objetos.

Multithread: sistema por el cual varios programas iguales pueden estar ejecutando en la misma CPU, repartándose el tiempo de ejecución.



Multiplexar: establecer sobre un mismo port o canal de comunicación físico, varias comunicaciones lógicas que se alternan en el uso de dicho canal.

Object Request Broker (ORB): una infraestructura que maneja la comunicación entre objetos y componentes. Permite a los objetos comunicarse entre diferentes e independientes sistemas de computación.

Objeto: entidad compuesta de datos y métodos. Los métodos definen el tipo de operaciones que se pueden realizar sobre el objeto. Las propiedades básicas de un objeto son encapsulamiento, polimorfismo y herencia.

Polimorfismo: define la habilidad de seleccionar un método basados en el objeto destino permitiendo que el mismo requerimiento pueda ser enviado a múltiples objetos, activando diferentes respuestas.

Port: puerta o canal de comunicaciones que utilizan las computadoras para comunicarse con los demás componentes de la red a la que pertenecen. Pueden ser físicos o lógicos.

RPC: (Remote Procedure Call) sistema por el cual se invoca un procedimiento o función que se encuentra físicamente ubicado en un servidor remoto

Runtime: aplicación que se utiliza para ejecutar programas previamente compilados pero que no pueden ser ejecutados por sí mismos. Se utilizan con el objeto de mantener una sola aplicación que pueda ser ejecutada en diversos ambientes con la ayuda de los programas de runtime.

Stub: (Cabo) programa encargado de implementar una llamada RPC, ocultando a quien invoca la función la información referida a la localización del servicio llamado.

TCP: (Transfer Control Protocol) es un protocolo que se encarga de las transmisiones, junto con IP representan el protocolo más importante de Internet, el TCP/IP.

Telnet: protocolo de terminal virtual que se ocupa de mantener y administrar el flujo de datos en ambas direcciones entre la terminal y la computadora central.

Thread: se puede traducir como canal, y se lo puede definir dentro de un sistema como el espacio donde se ejecuta un programa.

UDP: User Datagram Protocol. (Protocolo de Datagrama de Usuario) Protocolo abierto en el que el usuario (programador) define su propio tipo de paquete.



Bibliografía

[AKP98]

www.Akpublic.research.att.com.

[Bartlett 97]

Bartlett. Leslie. Simkim. - "JAVA Programming", 1997. Editorial Coriolis Group.

[BINE84]

Birrel y Nelson, 1984.

[Cornell 06]

Cornell. Cay. Horstmann. - "Core JAVA", 1996. The Sunsoft Press Java Series.

[Damasceno 98]

Américo Damasceno- "JAVA en Internet", 1998. Editorial Metodos.

[DC99]

DCOM Solutions in Action

"DCOM and Dev Issues" section of the PDC 97 Conference Papers in the MSDN Library.

[DCP99]

Distributed Component Object Model Protocol—DCOM/1.0 (MSDN Library, Specifications)

[EUR95]

EuroSTAR '95, 27-30 November 1995, London UK.

[JAV99]

www.java.sun.com

[MIC98]

www.microsoft.com/oledev/olecom.

[MS99]

The Component Object Model Specification (MSDN Library, Specifications)

[OMG: 95a]

Common Object Request Broker Architecture, OMG, July, 1995.

[OMG: 95b]

Common Object Services Specification, OMG 95-3-31, 1995

[Orchard 96]

Comparison of CORBA/DCOM/Java. Dave Orchard, 1996



[ORG96]

Object Management Group www.org.com.

[ORF97]

The Essential Client/Server Guide, John Wiley & Son, 1997

[PAC98]

www.Pacificspirit.com.

[RNL 97]

Claudio Righetti. María Eva Lijding. Leandro Navarro Moldes.

"A new Large-Scale Distributed System" Proceeding of International Conference for Computer Communications - ICCC'97. 19-12 Noviembre 1997, Cannes, Francia.

[SEG98]

www.Seg.cs.hut.fi.

[Tanenbaum 91]

Andrews Tanenbaum - "Computers Networks", 1991.

[Vinoski]

Steve Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, February, 1997.



Anexo I

Experiencias de la Implementación

JDK-RMI

Las facilidades de exportación del código generado por JKD a diferentes plataformas, con la única condición de contar con el runtime y los nuevos conceptos que surgen con el lenguaje Java, hacen que podamos definirlo como el entorno que mejor se adapta a las necesidades de desarrollo para una aplicación cliente-servidor en una red de área amplia.

Originalmente se pensó en utilizar RMI para la comunicación entre el Agente Usuario y el Agente de Servicio. Esto brinda facilidades de ejecución de procesos remotos sin que el cliente tenga que manejar la complejidad de una llamada remota. El cliente invoca los procedimientos que necesita y el agente RMI se encarga de ejecutarlos en forma local o remota según donde se localice el proceso, haciendo transparente este trabajo a los ojos del programador del Agente de Usuario.

Realizamos en una primera implementación RMI. Nos pareció sencillo de usar y robusto en las pruebas iniciales realizadas. En el primer prototipo las comunicaciones entre el servidor y el cliente utilizaban RMI para la transferencia de datos de control y documentos propiamente dichos. La programación fue relativamente fácil y logramos una performance aceptable a pesar de estar en etapa de desarrollo.

Por otro lado, esta utilización no es flexible para la implementación definitiva del Agente de Usuario en otro lenguaje de programación, ya que con este esquema se estaría obligando a usar un lenguaje que maneje RMI, y esto sólo lo hace Java. Con el objetivo de no limitar los posibles desarrollos futuros en este sistema se tomó la decisión de no implementar bajo la característica de RMI la comunicación entre Cliente y Servidor en el desarrollo.

Finalmente, dada la implementación Cliente – Servidor utilizada para la comunicación entre el Agente de Usuario y el Agente de Servicio, creemos que es la más conveniente, dado que si el Agente de Usuario se implementa en otro lenguaje sólo le basta conocer los tipos de mensajes en entre ambos agentes. De esta forma, se independiza el lenguaje de programación del Agente de Usuario.

Sobre el Lenguaje Para el Desarrollo

Fortalezas

Cuando realizamos la comparación con C++, observamos que generalmente en Java se necesitan menos líneas de código para realizar el mismo programa. Esto nos permite ganar tiempo en el desarrollo de aplicaciones.



Una ventaja interesante es que Java utiliza las funciones de librerías en el momento de ejecución y esto permite realizar por ejemplo una actualización de las librerías sin necesidad de compilar la aplicación nuevamente.

Java es un lenguaje fuertemente tipado, lo cual permite encontrar errores en el momento de compilación, evitando sorpresas durante la ejecución del mismo. Esto permite a su vez que los programadores trabajen más correctamente desterrando los vicios que muchos de ellos presentan de no especificar el tipo de dato que utilizan en las variables.

El buen manejo de la memoria nos hizo independizar de como administrar la misma, siendo de gran ayuda en el momento de la programación.

La comunicación a través de sockets nos resultó sencilla y confiable. Realizamos pruebas de caída de uno de los nodos para el caso de conexiones punto a punto y el otro nodo se enteraba al instante, dado que esta caída provoca el lanzamiento de una excepción.

El manejo de errores nos resultó práctico y confiable, sobre todo para la comunicación entre procesos y otras validaciones que necesitamos durante el desarrollo.

La capacidad del lenguaje de manejar thread nos permitió sin mayores esfuerzos programar procesos para atender múltiples procesos clientes en forma concurrente. Cada uno de estos procesos thread, atienden en forma exclusiva al proceso cliente. En todos los casos, cuando el proceso cliente no necesita realizar más requerimientos al proceso servidor, éste cierra la comunicación y el proceso servidor thread es finalizado.

Como todos los browsers actuales tienen la capacidad de ejecutar aplicaciones Java por medio de applets, entonces existe la posibilidad de transformar el Agente de Usuario en un applet sin mayores esfuerzos, y ejecutar a través de una página de Internet todas las consultas y actualizaciones que el cliente desee realizar sobre el Agente de Servicio.

Debilidades

El manejo de memoria en forma automática por medio de la técnica de Garbage collection provoca lentitud en la ejecución de los programas, dado que Java se debe encargar de realizar el manejo de la memoria.

Cuando realizamos el desarrollo del Agente de Usuario, nos encontramos que la interfase gráfica nativa era muy limitada, así que recurrimos a otras herramientas desarrolladas en Java, pero no por Sun, que permitían un mejor manejo de las interfases gráficas.



Durante el desarrollo y aprendizaje del lenguaje, la falta de ejemplos claros para programación de las funciones mas complicadas nos hizo perder un tiempo considerable. La mayoría de otros lenguajes, presentan en su documentación una buena explicación de la funcionalidad de las funciones y ejemplos claros. Esto en Java no está tan presente como hubiésemos deseado.

Los productos nativos de Java presentan cierta dificultad para el debugging. Ya que Java es un lenguaje de última generación, nos parece que debería tener mejores herramientas para manejar este aspecto.

Notamos una falta de un entorno nativo más amigable de programación y compilación. Es un poco ilógico que un lenguaje que funciona entre otras cosas en Internet, se deba compilar desde el DOS. Ultimamente, han mejorado este aspecto ya hay herramientas que no son desarrolladas por Sun que permiten que un programa escrito en Java pueda ser compilado directamente desde esta herramienta.

Sobre la Base de Datos

El manejo de la base de datos no resultó bastante sencillo, pues Java presenta varios drivers que permiten conectarse a diferentes motores de base de datos. El utilizado por nosotros nos permitió comunicarnos con la base de datos elegida en forma sencilla y segura. Uno puede cambiar entre los diferentes motores con solo cambiar el driver, y como a su vez permite el manejo de sentencias SQL, el cambio es sencillo, pero puede ocurrir que se tenga que realizar alguna modificación en la sentencia SQL, por que un motor de base de datos no llegue a interpretar una función que maneja otro. Dado que Java presenta varios drivers, brinda la posibilidad a los programadores utilizar la base de datos que deseen.

Java nos presenta, como dijimos anteriormente, varios drivers pero no posee una base de datos nativa. La desventaja es que la comunicación con la misma se deba realizar a través de un “driver” y no en forma directa. Por las pruebas realizadas no podemos decir que el acceso a la base sea lento.

La base de datos utilizada para la aplicación de ODS es Access, dado que es una herramienta estándar. Si tenemos en cuenta que solamente el manejo y control de la información se almacena en la base de datos, la elección de Access es adecuada. Los documentos propiamente dichos no están almacenados dentro de la base de datos, sino que se mantienen en un espacio elegido en el Agente de Servicio.

El resultado de la ejecución de una sentencia SQL se almacena en un cursor, y el mismo puede ser recorrido por medios de métodos (funciones). Si una sentencia está mal escrita, o el resultado de la misma es inválido, se dispara una excepción, permitiendo de esta manera poder atrapar el error y actuar en consecuencia. Las sentencias SQL, pueden variar entre los distintos motores de base de datos, principalmente en la forma o nombre de una función, como puede ser la función SUBSTR de ORACLE, que se llama MID en ACCESS. En las sentencias que no



devuelven filas como es el caso de un INSERT, DELETE, UPDATE, se puede obtener la cantidad de filas procesadas por éstas, como así también si hubo un fallo en la ejecución de la misma. Como en el caso de la sentencia SELECT, el error puede ser atrapado.

La limitación de la complejidad de las sentencias SQL varían entre los distintos motores de base de datos. Las sentencias SQL utilizadas en el desarrollo de ODS son de baja complejidad para minimizar los cambios ante un posible cambio en el motor de la base de datos.

Decisiones más Importante Durante la Implementación

Antes de comenzar con la implementación de ODS, estuvimos analizando los lenguajes Java y C++. Después de esta comparación y viendo cuales son las ventajas y desventajas que presenta cada uno de los lenguajes, decidimos realizar esta implementación en Java ya que no solo a nuestro criterio nos pareció que reunía las mejores características para este proyecto, sino también el desafío de desarrollarlo en un lenguaje que nació para Internet.

En el proceso de análisis para elegir la base de datos que se utilizaría para almacenar los datos de los objetos (documentos), nos inclinamos por Access, ya que la misma está prácticamente en todas las PC bajo Windows y además, cumplía con los requisitos necesarios para almacenar los datos requeridos. Por otro lado, si hubiésemos elegido otro motor de base de datos que no fuese de uso masivo, se tendría que pagar por las licencias donde se utilice la misma, es decir, por cada Agente de Servicio

Para el caso de los documentos físico (archivo), estos no se almacenan en la base de datos, se los almacena en un directorio, y el mismo es configurable en cada agente de servicio. Utilizamos esta política de almacenamiento del documento, para que la base de datos no incremente su volumen y no obligar en la elección de la base de datos a que ésta tenga la capacidad de manejar un tipo de datos binarios para poder almacenar los documentos.

Un punto importante en la distribución de objetos, es el que cumple la red de ruteo de los mensajes, que nos indica en forma dinámica por donde debe viajar el documento para llegar a su destino. Para poder manejar la red de ruteo, se tienen que desarrollar algoritmos complejos que no era el objetivo de nuestra tesis.

Durante el análisis de ODS, se planteó la posibilidad de que los documentos antes de estar disponibles, deben pasar por la autorización de una persona de nivel superior que se encarga de verificar si el documento ingresado cumple todos los requisitos, es decir, que el mismo contenga información sobre el tema del grupo al cual pertenece. Para implementarlo se tiene que asignar una persona por cada Agente de Servicio y cada grupo de interés. A su vez, el autenticador puede rechazar el documento y el dueño del mismo tiene que volver a modificarlo para su aprobación. Esto puede ocurrir varias veces hasta que el mismo fuese considerado válido para su difusión.



Otra alternativa pensada, era que el documento sea aprobado por el grupo de usuarios que pertenecen al grupo del documento a ingresar en un determinado servidor. Una vez aprobado por estos, el documento estaba listo para ser transmitido al resto de los Agentes de Servicios.

Estas dos formas nos parecieron muy engorrosas, entonces utilizamos una forma de distribución más sencilla, y decidimos que cada usuario productor de un grupo de interés es el responsable del contenido del documento y su divulgación. De esta manera, se simplifica la distribución de documentos, con la desventaja de que el mismo queda sólo a criterio del productor.

Principales Dificultades

Java permite enviar a través de una conexión, tanto punto a punto como datagramas, objetos, pero la desventaja que presenta es que estos objetos deben ser serializados y no todos los objetos son serializables.

A pesar de que la base de datos Access es SQL estándar, presenta ciertas particularidades en el manejo de funciones que nos obligarían ante un cambio del motor de base de datos, a re-escribir algunas sentencias. Esto en sí, no representa un gran problema, pero una molestia al momento de encarar el cambio en lo que respecta al tiempo que se tiene que invertir para su reemplazo. Cabe aclarar que la cantidad de sentencias SQL que pueden llegar a tener problemas son muy pocas.

En el momento de la instalación del programa cliente o servidor en un equipo bajo Windows, se debe realizar una registración de Java, si es que la máquina no posee Java. Por otro lado, para el caso de los agentes de servicio y también bajo el entorno Windows, se debe registrar en el ODBC 32 bits el nombre de la base de datos y donde se va a almacenar la misma. Esto último no se tiene que hacer cuando se instala un Agente de Usuario ya que el mismo no accede directamente a la base de datos. Sin estas registraciones no se puede utilizar ni el Agente de Servicio ni el Agente de Usuario.

Java no crea una librería con las funciones que utiliza en cada aplicación, sino que debe distribuir todas las librerías por más que sólo se use una función. Esto presenta un problema de espacio a la hora de instalar una aplicación Java.



Anexos II

Comparación Java con C++

Una manera para comenzar a entender la diferencia entre los lenguajes es escribir el programa “Hola”

En C:

```
#include <stdio.h>
main() {
    printf("Hola, mundo \n");
    return 0;
}
```

En C++:

```
#include <iostream.h>
main() {
    cout << "Hola, mundo \n";
    return 0;
}
```

En Java:

```
public class hola (
    public static void main (String args[]) {
        System.out.println("Hola, mundo");
        System.exit(0);
    }
}
```

El lenguaje C y C++ son similares, pero Java es notoriamente diferente. Algunas de estas diferencias son las siguientes:

- Java no tiene archivo de encabezamiento.
- Java “conoce” ciertas características del sistema como ser la clase String y la output stream (System.out).
- No hay funciones globales como en C y C++, pero en su lugar todas las funciones (métodos) deben ser parte de la misma clase.
- main() es un método de la clase hola, pero este no opera específicamente como una instancia de la clase, y en su lugar es llamado “*método de la clase*” que es parte de la clase para el propósito de la aplicación. En C++ la equivalencia podría ser una “static member function”.
- La clase hola debe ser definida en un archivo llamado hola.java. No hay un requerimiento equivalente en C o C++.



- Para pasar un estado de salida desde el programa, Java usa `System.exit(n)`. En C y C++ hay una función de librería llamada `exit()`, o uno puede retornar un valor desde `main()`.

Una de las grandes diferencias es una buena noticia para los programadores de C/C++ que pasan a Java, pues Java no tiene operaciones de manejo de memoria ya que usa una técnica conocida como garbage collection para este fin.

La manera que garbage collection trabaja es simple, limpia automáticamente todo la memoria cuando en la misma no existen referencias de utilización. Veamos un ejemplo:

```
Public class hola {  
    static void f () {  
        int x [] = new int[1000];  
    }  
    static void g () {  
    }  
    public static void main(String args[]) {  
        f();  
        g();  
    }  
}
```

Después de llamar al método, `f()` retorna y ubica dinámicamente el vector `X[]` como un posible desperdicio y es un candidato para recobrar memoria. Típicamente comenzará a usarse sólo cuando es necesario, esto es, cuando la memoria libre es baja. También es posible forzar el garbage collection.

El lado negativo del garbage collection es que hace a Java más lento que C/C++. También podemos decir que la facilidad de manejo de memoria de Java es de más alto nivel que C/C++. Esto es una ventaja para los programadores, pues hay menos problemas por el mal uso de la memoria.

Tamaño de los Tipos de Datos Primitivos

En Java el tamaño de los tipos de datos es uniforme en todas las implementaciones. Específicamente, los tamaños en bit son los siguientes

Boolean	N/A
Byte	8
Char	16
Short	16
Int	32
Long	64
Float	32
Double	64



El tipo Boolean es integral y no está listado el tamaño. Este puede tener el valor *true* o *false*. El tipo carácter es de 16 bits usando el conjunto de caracteres Unicode.

Las ventajas de tamaños uniformes son obvias.

Java no tiene el operador `sizeof()` como en C/C++. Con el tamaño uniforme de datos, y el manejo del compilador de detalles de cálculo para el tamaño de espacio necesario para ubicar sentencias como:

```
long x[] = new long[189];
```

Como se puede ver en esta sintaxis, no es necesario tal operador.

Una Forma de Hacer Inicialización de Clases

En C++ se pueden usar constructores para inicializar una instancia de clase de un objeto cuando es creado y emplea datos estáticos que son inicializados cuando el programa comienza. Pero ¿si le gustaría que el mismo código sea ejecutado una vez que la clase este dada, qué clase de cosas tendría que hacer?. Una forma de hacer esto en Java es:

```
public class A{
    static {
        System.out.println("got to static initialization");
    }
    public static void main(String args[]) {
        System.out.println("start of main()");
        A c1 = new A ();
        A c2 = new A ();
    }
}
```

No hay problemas en lo que respecta a cuantas instancias del objeto de la clase sean creadas, el bloque de código de arriba será ejecutado sólo una vez. Esto sirve como gancho para hacer el seteo de la clase al comienzo de la ejecución.

Variables Globales y Funciones en Java

Java no tiene variables globales o funciones como alguno de los otros lenguajes comunes. Toda variable o función debe ser parte de una misma clase. Dentro de una clase las variables o funciones (métodos) pueden ser un miembro regular de la clase, o puede ser una variable de clase, o un método de clase.



Los métodos y las variables de la clase no operan sobre una instancia particular de una clase. Un método de clase es típicamente usado como función dentro de la clase, mientras que las variables de clase están compartidas por todas las instancias de la clase.

Usar métodos y variables de clase, es algo similar a las variables globales y funciones.

Como Enganchar Listas en Java

Java no usa punteros como C/C++, entonces ¿cómo se pueden enganchar listas? Cada elemento necesita engancharse con el próximo ¿cómo se hace?

```
public class link {
    public int value;        // value of element
    public link next;       // reference to next

    // constructor
    public link(int n, link ln){
        value = n;
        next = ln;
    }

    public static void main(String args[]){
        // initialize list head
        link head = null;

        // add some entries to list
        for (int i = 1; i <= 10; i++){
            head = new link(i, head);
        }

        // dump out entries
        link p = head;
        while (p != null) {
            System.out.println(p.value);
            p = p.next;
        }
    }
}
```

Como ya dijimos, Java no tiene punteros, pero en su lugar usa referencias implícitas, como en la línea:

```
link next;
```

No declara actualmente una instancia de la clase link, pero en su lugar hace referencia a una instancia de un objeto. La línea:



```
head = new link (i, head);
```

crea un nuevo elemento para la lista, setea el valor, y luego setea el valor de referencia en “next”, que apunta al viejo elemento de head (esto significa que el último elemento a ser insertado será el primero de la lista). Diciendo:

```
p = p.next;
```

simplemente se toma el “next” del nodo actual y hace que el elemento actual exista trabajando así.

Sobrecarga de Operadores

En C++ el siguiente código:

```
class A {  
public:  
    void operator+=(int);  
};  
  
void f() {  
    A a;  
    A += 187;  
}
```

es ilegal. El operador “+=” ha sido sobrecargado para un especial significado cuando su primer operando es una clase objeto. Esto se puede usar con una sentencia equivalente a:

```
a.operator+=(187);
```

esto es, una llamada función miembro sobre una instancia de un objeto.

En Java no tiene operador de sobrecarga. Pero puede tener una secuencia como:

```
Public class test1 {  
    public static void main (String args[]) {  
        String s1 = “abc”;  
        s1 += “def”;  
    }  
}
```

la cual es perfectamente legal. Dado que en C/C++ “+=” sólo trabaja operaciones aritméticas o sobre punteros, y en Java no tiene operador de sobrecarga, porque esto es legal?



La respuesta es que en Java, el tipo String es parte del propio lenguaje. El método String es definido en librerías, como en otros lenguajes, pero los String están actualmente en el lenguaje mismo. Cuando “+=” es encontrado, es sustituido por una llamada al propio método.

Múltiple Herencia

C++ maneja múltiples herencias pero Java no. Sin embargo tiene una manera de hacer algo parecido que es lo que se llama interfase. Por ejemplo

```
Public interface Orderable {  
    // return < 0 if P1 < p2, 0 if equal, > 0 if p1 > p2  
    public int compareTo (Object p1, Object p2);  
}
```

Esto se lo puede mirar como una clase, excepto que todos los métodos son abstractos, esto es, no son implementados inmediatamente pero sirven como contenedores de lugar.

¿Para que se usa una interfase?. Supongamos que se quiere asegurar que una clase particular tenga ciertas propiedades, y en el presente caso se define el método compareTo para ordenar objetos. Se quiere respetar esto por requerimiento al implementar la clase Orderable

```
public class xxx implements Orderable {  
    public int compareTo (Object p1, Object p2) {  
        int n1 = ((Integer)p1).intValue();  
        int n2 = ((Integer)p2).intValue();  
        return n1 - n2;  
    }  
}
```

Java no tiene por default una manera arbitraria de ordenar dos objetos. Si se define una interfase Orderable y en la implementación se define el tipo de objeto, se podría hacer un vector de objetos y armar una rutina general para ordenarlos. En cambio C++ tiene una rutina de comparación propia llamada “qsort”.

Sobrecarga de Métodos

Tanto en C++ como en Java es posible definir funciones/métodos con el mismo nombre más de una vez. Por ejemplo en C++:

```
class A{  
public:  
    void f (int i) {}  
    void f (char* s) {}
```



```
};
```

el compilador elegirá la opción correcta basado en el tipo de dato que se utiliza en la llamada a la función. Java se comporta de forma similar.

Una pregunta natural que se presenta es como se elige la llamada a la función o método correcto. Por ejemplo, con este código en C++:

```
void f(short s){}  
void f(long l){}
```

y una llamada:

```
char c = 0;  
f(c);
```

no es claro cual de las funciones hay que llamar, pues el char puede ser representado como un short o un long (En este caso particular la llamada es en efecto ambigua en C++).

Java usa algoritmos que se conocen como “más específicos” para encontrar el método adecuado en la llamada. Antes de explicar esto, veamos un poco de la asignación, que tipos de datos se pueden asignar a otro tipo de dato. Veamos para ello un ejemplo:

```
public class test 1 {  
    public static void main (String args[]) {  
        short s = 0;  
        int i = 0;  
        float f = 0.0f;  
  
        i = s; // OK  
        f = i;  
  
        s = i; // error  
        i = f;  
  
        s = (short) i; // OK pero se debe manejar con cuidado  
        i = (int) f;  
    }  
}
```

Se pueden asignar datos primitivos más chicos a otros datos de mayor tamaño, como por ejemplo asignar un short(16 bits) a un int(32 bits). Por lo tanto, es válida la asignación de un int a un float. Por otro lado, no se puede realizar la asignación de un tipo de datos de mayor tamaño a uno de menor tamaño, y es por eso que está marcado como error la asignación de un int a un short. A su vez, se puede realizar un “cast” para pasar de un tipo de mayor tamaño a uno de menor tamaño, realizándolo con mucho



cuidado, pudiendo con esto perder precisión en la asignación. También se pueden realizar asignaciones de objetos a clases básicas, pero no hay otra manera que a través del casting.

Ahora ya que tenemos una idea de la forma de cómo se realizan las asignaciones en Java y dada la actual forma de llamar a un método, ¿cómo hace Java para resolver esto?. Primero de todo, busca todos los métodos que tienen el mismo nombre, luego busca de éstos los que coinciden en la misma cantidad de parámetros, y luego los parámetros que son del tipo que se pueden asignar según los valores de los argumentos. Si sólo un método matchea, entonces este es el método que será invocado.

Si hay más de un método que coincide en todos los datos descriptos arriba, busca los métodos para los cuales todos los tipos de los parámetros pueden ser asignables a otros parámetros de otros métodos, y remueve esos otros métodos del conjunto de métodos considerados. El método removido es considerado “menos específicos”

Si luego de este descarte queda un solo método, éste es invocado normalmente. Si queda más de un método, entonces la llamada es ambigua.

Vemos un ejemplo para ver como trabaja Java:

```
public class test2 {  
    public static void f1(int i, int j){}  
    public static void f1(int i, short j){}  
  
    public static void f2(long i, long j){}  
    public static void f2(int i, int j){}  
  
    public static void f3(short i, short j){}  
  
    public static void f4(double i, float j){}  
    public static void f4(float i, double j){}  
  
    public static void main (String args[]) {  
        int i = 0;  
        short s = 0;  
  
        f1(i, s);  
        f2(i, s);  
        f3(i, s);  
        f4(i, s);  
    }  
}
```

La llamada `f1()` invoca a `f1(int, short)` y esto es un matcheo exacto.

La llamada `f2()` invoca `f2(int, int)` porque `f2(int, int)` tiene parámetros que son invocables a `f2(long, long)` y por lo tanto éste es removido del conjunto.



La llamada a `f3()` causa un error porque el `int` del primer parámetro no puede ser asignado a un `short`.

Por último la llamada a `f4()` es ambigua por que ambos métodos pueden ser llamados y ninguno de los dos puede ser removido del conjunto de posibles métodos porque ninguno tiene parámetros asignables en el otro.

Para C++ es justo decir que la regla de `matcheo`s de los argumentos en este lenguaje es mucho más complicada.

Destructor y Concluir (`finalize`)

En C++ hay una noción de un destructor, una función llamada cuando una instancia de clase de objeto sale de escena. Por ejemplo:

```
class A {  
public:  
    A();  
    ~A();  
};  
void f() {  
    A a;  
};
```

Cuando la función `f()` comienza, el constructor `A::A()` es llamado para el objeto local “a”, y cuando `f()` finaliza, el destructor `A::~A()` es llamado.

Los objetos de la clase Java no tienen un comportamiento similar. Ellos son dinámicamente alocados, en este punto un constructor es ejecutado sobre la instancia del objeto, más tarde el “garbage collection” será el encargado de eliminarlo definitivamente.

Java no tiene de un método destructor que corre cuando la instancia del objeto deja de ser válida, pero en su lugar tiene un método `finalize()`:

```
protected void finalize();
```

que puede ser agregado a cualquier clase. El método `finalize()`, es llamado una instancia de un objeto un instante después que ha sido identificado por el garbage collection como un objeto que dejó de ser válido, lo que ocurre cuando no hay más referencias a éste.



Tipos Enumerados

En C++ existe una facilidad conocida como tipo de enumerados que puede ser usado para representar un conjunto de valores integrales:

```
enum Color {CO_R = 1, CO_G = 2, CO_B = 3};
```

Después de esta declaración; Color puede ser usado como un tipo y valores como CO_R puede ser usado como una constante. El chequeo de tipo es hecho, entonces para el ejemplo:

```
Color c = CO_R;  
c = 37;
```

es inválido.

En Java no existe tipo enumerado. Una manera de simular esto es simplemente definiendo constantes en una clase

```
public class Color {  
    public static final byte CO_R = 1;  
    public static final byte CO_G = 2;  
    public static final byte CO_B = 3;  
};
```

public significa que estos valores pueden ser accedidos por todos, static significa que ellos son compartidos a través de todas las instancias del objeto, final significa que los valores permanecen inmutables.

Typedef

En C/C++, hay una facilidad conocida con “typedef”, por lo cual se puede definir un nombre para representar un tipo. Por ejemplo:

```
typedef unsigned char* T;
```

dice que T es un tipo de puntero a un carácter sin signo. Una vez que el typedef haya sido especificado, éste se puede usar donde quiera que un tipo es requerido, por ejemplo para declarar una variable:

```
void f(){  
    T x;  
}
```

Java no tiene typedef, y es interesante considerar este punto. Hay al menos tres razones por las que los typedef son útiles en C/C++.



La primera razón es definir types como una ayuda a la portabilidad. Por ejemplo, un typedef puede ser definido para representar un entero de 32-bits.

```
typedef int INT32; // sobre una workstation
typedef long INT32; // sobre una PC
```

Los enteros son variables de tamaño en C y C++. Esto no ocurre en Java, puesto que los tipos son de tamaño uniforme.

La segunda razón para los typedef en C es que se introduce un “alias” para el nombre de la estructura:

```
Struct A {
    Int x;
}
typedef struct A A;
A x;
```

En C++ y en Java esto es implícito en el lenguaje.

Por último, los typedef son útiles para representar combinaciones de tipo complicadas, como en:

```
Typedef unsigned long** T;
```

El cual dice que T es un puntero de un puntero de un tipo long sin signo. Java no usa punteros.

Los typedef son útiles simplemente para propósito de abstracción, esto es, una manera abstracta de nombrar un tipo, como por ejemplo “edad” en lugar de “int”. Pero muchos de esos usos no se aplican en Java, y esto es una razonable decisión por la que dejaron afuera los typedef del lenguaje.

Operadores

Si uno conoce C/C++ muchos operadores le resultaran familiares en Java, pero hay algunas diferencias que son dignas de ser destacadas.

1. Java no tiene tipos sin signos, y también los operadores >>> han sido agregados para soportar el desplazamiento derecho sin signo. Esto es, el desplazamiento derecho >> propagará el signo, considerando que usando el desplazamiento >>> no llenará con cero el bits más significativo. >>>= también a sido agregado.

El operador “instanceof” puede ser usado para determinar si un objeto es una instancia de una clase dada, como en:

```
If (s instanceof String)
```



- ...
2. Java no tiene los siguientes operadores de C/C++:

```
,  
->  
.*  
->*  
::
```

3. Java no tiene operadores de sobrecarga, como por ejemplo [] para proveer un tipo de vector “inteligente” con algún tipo de ordenamiento.
4. En Java “+” puede ser usado para la concatenación de String.

Cláusula Finally y Método de Salida

Java no tiene la sentencia “goto”, aunque este identificador está reservado en el lenguaje. Hay varias maneras en las que “goto” es usado en C/C++, y es interesante considerar una alternativa para tal costumbre. Veamos una posible alternativa.

Una manera en que el “goto” es usado, es para saltar al final del cuerpo de una función. Supongamos que estamos manejando fechas del calendario, y tenemos una función que queremos que los años estén dentro de un rango de 1900-1999 y eventualmente divisible por cuatro. En C podríamos tener:

```
void f(int d){  
    if (d < 1900)  
        goto err;  
    if (d > 1999)  
        goto err;  
    if (d % 4)  
        goto err;  
    ...  
    return;  
err:  
    fprintf(stderr, “invalid date %d\n”, d);  
}
```

En Java, se puede conseguir algo similar, pero sin goto usando una sentencia de la forma “try-catch-finally” que es usada para el manejo de excepciones.

```
public class test {  
    public static void f(int d) {  
        boolean err = true;  
        try {  
            if (d < 1900)  
                return;  
        }  
    }  
}
```



```
        if (d > 1900)
            return;
        if (d % 4 != 0)
            return;
        err = false;
    }
    finally {
        if (err)
            System.out.println("Invalid date " + d);
    }
}

public static void main (String args[]) {
    f(1852);
    f(1976);
    f(1989);
}
}
```

El código que está dentro del bloque “try” es tratado, esto es, ejecutado. Después de la ejecución de este código, el bloque “finally” es ejecutado, sin importar lo que suceda dentro del bloque “try”. En el ejemplo de arriba, se sale del método (return) por varias condiciones de error, pero cuando el método es finalizado, se ejecuta el bloque “finally”. En este sentido, se pueden ejecutar una serie de sentencias en Java, y garantizar que no habrá problemas en esas sentencias, y el proceso seguirá.

Si una excepción es disparada en el bloque “try”, y existe el “catch” local para el “try”, este bloque es ejecutado, y luego el bloque “finally”. Si no hay un bloque de catch local al “try”, el bloque “finally” es ejecutado, y luego se propagará la excepción al catch del bloque “try” más cercano que pueda manejar la excepción.

Break con Etiquetas y la Instrucción Continue

En C/C++, el break y la sentencia continue son usados para quebrar y continuar con la próxima iteración del ciclo (loop), como por ejemplo un while o un for. El “break” se usa también en la sentencia switch.

Estas sentencias que se usan encerradas en loop, sería importante que fueran más flexibles, como por ejemplo usar break con label. En C se puede decir:

```
for (i = 1; i <= 10; i++) {
    for (j = 1; j <= 10; j++) {
        if (some condition)
            goto done;
    }
}
done:
```



```
/* other processing */
```

Este efecto puede ser alcanzado por quebrar en el loop de adentro, y luego testear alguna variable de estado del programa, y luego quebrar el loop de afuera.

En Java, hay una manera más simple de resolver el problema.

```
outer_block:
for (i = 1; i <= 10; i++) {
    for (j = 1; j <= 10; j++) {
        if (some condition)
            break outer_block;
    }
}
// Aquí vuelve el control cuando el break es ejecutado
```

La sentencia break tiene un label. Cuando la sentencia es ejecutada, el control se transfiere a la sentencia siguiente del bloque etiquetado encerrado entre llaves. El control NO se transfiere a la etiqueta. Las sentencias etiquetadas para el continue trabajan de manera similar; el control es transferido al final del loop etiquetado.

Arreglos

Java tiene arreglos que en su uso son como los arreglos convencionales de C/C++. Si por ejemplo uno quisiera sumar los primeros 10 elementos de un arreglo, se puede decir:

```
int sum = 0;
for (int i = 0; i < 10; i++)
    sum += a[i];
```

Pero hay una mayor diferencia en Java cuando maneja los arreglos. Una gran diferencia es que Java chequea los subíndices, si en el ejemplo anterior en vez de llegar hasta 10 lo hacemos llegar hasta 11, se disparará una excepción si el arreglo está definido de 10 elementos.

Otra diferencia es que en Java el arreglo puede ser de longitud cero, como por ejemplo:

```
int x[] = new int [0];
```

donde C++ lo soporta pero no C.

El tamaño o largo del arreglo se obtiene en cualquier momento diciendo:

```
int len = x.length;
```




Los nombres de los arreglos son referenciados, y de tal manera es posible asignar un arreglo como:

```
int x[] = new int[10];  
int y[] = null;  
y = x;
```

En C/C++, el arreglo actual no puede ser asignado

```
int x[10]  
int y[10];  
y = x;           // Invalido
```

aunque los arreglos están representados ciertamente por punteros.

En C++, el código

```
class A {  
public:  
    A();  
};  
A* p = new A[10];
```

Resultará 10 llamadas al constructor, una por cada elemento del arreglo, mientras que en Java el código:

```
public class A{  
    A();  
    A p[] = new A[10];  
};
```

no ocurre. Mas bien, el arreglo que tiene 10 referencias a la instancia de la clase A será creado, y cada elemento será inicializado en nulo. En otras palabras, la nueva llamada en C++ aloca el espacio y luego el constructor será llamado para cada elemento, mientras que en Java, se debe inicializar el arreglo diciendo

```
for (int i = 0; i < 10; i++)  
    p[i] = new A();
```

Declaraciones de Variables Locales

En C, cuando se declaran variables locales, éstas son agrupadas necesariamente todas juntas al comienzo de la función.

```
void f() {  
    int i;  
    double d;
```



```
    char* p;  
}
```

En C++ y Java esto no es necesario dado que permite que se declaren las variables en cualquier parte dentro de la función o método.

```
void f() {  
    int i = 0;  
    g(i);  
    String s = "xxx";  
    h(s);  
}
```

También es posible realizar las declaraciones de las variables al final de la función / método.

Performance

Una de las cosas que uno puede notar es que Java es más lento que otros lenguajes. Por ejemplo, en Java:

```
// sort numbers  
  
class bm1 {  
    static final int N = 6500;  
    public static void main(String args[])  
    {  
        int i;  
        int j;  
        short t;  
  
        short vec[] = new short[N];  
  
        // seed with descending order  
  
        for (i = 0; i < N; i++)  
            vec[i] = (short)(N - i);  
  
        // sort into ascending order  
  
        for (i = 0; i < N - 1; i++) {  
            for (j = i + 1; j < N; j++) {  
                if (vec[i] > vec[j]) {  
                    t = vec[i];  
                    vec[i] = vec[j];  
                    vec[j] = t;  
                }  
            }  
        }  
    }  
}
```



```
    }  
  }  
}
```

corre entre 35 a 50 veces más lento que su equivalente en C++.

```
// sort numbers  
  
#include <stdio.h>  
  
const int N = 6500;  
  
short vec[N];  
  
main()  
{  
    int i;  
    int j;  
    short t;  
  
    // seed with descending order  
  
    for (i = 0; i < N; i++)  
        vec[i] = N - i;  
  
    // sort into ascending order  
  
    for (i = 0; i < N - 1; i++) {  
        for (j = i + 1; j < N; j++) {  
            if (vec[i] > vec[j]) {  
                t = vec[i];  
                vec[i] = vec[j];  
                vec[j] = t;  
            }  
        }  
    }  
    return 0;  
}
```

Esto es debido a la naturaleza del interpretado de Java y también debido al chequeo extra en runtime que éste hace (tal como chequear si no se está afuera del rango de los vectores). En particular este algoritmo de ordenamiento no es muy eficiente dado que tiene orden N^2 , pero es ilustrativo para este punto.

Consideremos otro programa Java:

```
// number crunching
```



```
class bm2 {  
    public static void main(String args[])  
    {  
        long i = 1200000L;  
        double d;  
        while (i > 0) {  
            d = Math.sqrt((double)i) + Math.log((double)i);  
            i--;  
        }  
    }  
}
```

Este programa es tres veces más lento que su equivalente en C++.

```
// number crunching
```

```
#include <math.h>
```

```
main()  
{  
    long i = 1200000L;  
    double d;  
  
    while (i > 0) {  
        d = sqrt(double(i)) + log(double(i));  
        i--;  
    }  
  
    return 0;  
}
```

¿Por qué la diferencia?. Bien, el segundo programa, la mayor parte del tiempo lo gasta en la ejecución de las funciones matemáticas `sqrt()` y `log()`. Estas funciones son típicamente implementadas en hardware, vía rutinas de software de bajo nivel (métodos nativos). Si la mayor parte del tiempo se gasta en estas funciones (funciones matemáticas que son típicamente poco costosas), luego la interpretación de overhead requerida para manejar las interacciones del loop son menos importantes.

En muchos casos las diferencias de performance no son muy importantes. Por ejemplo, si realizamos un applet que es altamente interactivo u opera a través de Internet, el tiempo incurrido por la espera de respuesta de una persona y la red (Internet), será más importante que el tiempo puro del lenguaje.

Veamos como podría mejorar la performance. Hay varias posibilidades para mejorar esto. Una es la de generar código nativo, pero esto va en contra de cómo trabaja la portabilidad de los applets y aplicaciones de Java. Otra técnica similar es traducir Java a C (“java2c”), pero esto también va en contra de la portabilidad.



Hay una técnica llamada compilación “just in time” que está siendo considerada. Esta técnica consiste en hacer la compilación dinámica dentro del intérprete bytecode, la próxima vez que el bytecode es ejecutado, el actual código de máquina para la plataforma dada es sustituido en su lugar.

Performance – Operaciones de String

Supongamos que se quiere crear e imprimir una lista de números, como:

```
public class test1 {  
    public static void main(String args[])  
    {  
        String s = "[";  
  
        for (int i = 1; i <= 2000; i++) {  
            if (i > 1)  
                s += ",";  
            s += Integer.toString(i, 10);  
        }  
  
        s += "]";  
        System.out.println(s);  
    }  
}
```

Esto es válido pero lento. Este toma 13 segundos usando JDK 1.0 sobre una Pentium. Cuando se diseñó este programa se encontró que el “garbage collector” fue llamado en 40 oportunidades.

Resulta que usando sobre un String “+=” es poco costoso, y la razón es que los String son inmutables, esto es, no cambian después de la creación. Para agregar a un String se debe copiar el String a un StringBuffer, agregar éste y luego asignar el resultado a la variable original. Los StringBuffers son usados para hacer operaciones sobre String, como “+”, “+=”.

Esta idea puede ser ilustrada en el siguiente ejemplo:

```
public class test2 {  
    public static void main(String args[])  
    {  
        String s = "aaa";    // secuencia #1  
        s += "bbb";  
        System.out.println(s);  
  
        String ss = "ccc";   // secuencia #2
```



```
StringBuffer sb = new StringBuffer();
sb.append(ss);
sb.append("ddd");
String ss_save = ss;
ss = sb.toString();

System.out.println(ss_save);
System.out.println(ss);
    }
}
```

Estas dos secuencias son equivalentes; usando “+=” provoca lo mostrado en la secuencia #2 (excepto la línea `String ss_save = ss;`).

Volviendo al ejemplo original, se puede reescribir:

```
public class test3 {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer();
        sb.append("[");

        for (int i = 1; i <= 2000; i++) {
            if (i > 1)
                sb.append(",");
            sb.append(Integer.toString(i, 10));
        }
        sb.append("]");
        String s = sb.toString();
        System.out.println(s);
    }
}
```

Resultando 6 veces más veloz que el programa original.

Performance – Overhead por Llamada a los Métodos

En C/C++ se puede tener la idea de que llamando a una función, ésta tiene overhead asociado con el comportamiento del proceso actual efectuado por la función. Por ejemplo:

```
int max(int a, int b)
{
    return (a > b ? a : b);
}

int main()
```



```
{
    int i = 0;
    int j = 0;
    int k = 0;
    long n = 500000000L;
    while (n-- > 0)
        //k = max(i, j);
        k = (i > j ? i : j);
    return 0;
}
```

donde el cálculo de determinar el máximo entre dos enteros es tres veces más rápido cuando se hace “in line” que cuando se hace por medio de la llamada a la función `max()`. En C++ hay un especificador “in line” que puede ser usado para especificar que una función debe ser expandida en línea si es posible.

El overhead de la función es causada por varios factores, incluyendo el tiempo que pierde en el seteo del stack, transferir el actual control a la función llamada, y así sucesivamente.

En Java también existe overhead por llamada a los métodos. Por ejemplo:

```
public class xx {
    public /*final*/ int max(int a, int b)
    {
        return (a > b ? a : b);
    }

    public static void main(String args[])
    {
        int i = 0;
        int j = 0;
        int k = 0;
        int n = 1000000;
        xx p = new xx();

        while (n-- > 0)
            //k = p.max(i, j);
            k = (i > j ? i : j);
    }
}
```

haciendo el cálculo del máximo en línea es dos veces más rápido que llamar al método `max()`.

En Java una de las razones de overhead es que los métodos son virtuales por default, esto es, que el actual método que es llamado es resuelto en tiempo de corrida por consideración del actual tipo de objeto. Por ejemplo, la clase `Object`, de la cual



derivan todos los otros tipos de clase, tiene un método llamado hashCode() definido en ésta. Si un programa tiene una referencia a un Object, y el cual desea llamar a hashCode():

```
public void f(Object p)
{
    int h = p.hashCode();
}
```

luego la actual versión de hashCode() que es llamada depende de que “p” es realmente un Object o una referencia a una instancia de una clase derivada de Object.

Performance – charAt

Supongamos que necesitamos encontrar un carácter en un string, esto es, retorne una posición mayor o igual a cero si encontró la posición y -1 si no la encontró. En C, con sus punteros y el aprovechamiento de programación a bajo nivel, este es hecho a menudo tan eficiente o más como encontrar el carácter directamente en un loop, en lugar de usar una función de librería como strchr().

Esto es válido en Java? Supongamos que tenemos el siguiente código:

```
public class index {
    public static void main(String args[])
    {
        String s = "aaaaaaaaaa";
        int i = 250000;
        int n = 0;
        // método #1

        if (args[0].compareTo("index") == 0) {
            while (i-- > 0)
                n = s.indexOf('x');
        }

        // método #2
        else {
            while (i-- > 0) {
                int len = s.length();
                n = -1;
                for (int j = 0; j < len; j++) {
                    if (s.charAt(j) == 'x') {
                        n = j;
                        break;
                    }
                }
            }
        }
    }
}
```



```
}  
}  
}
```

Cuando corremos este código con JDK 1.0, encontramos que el método # 1 es aproximadamente 3 veces más rápido que el método #2, el cual es equivalente a usar punteros en C. ¿Por qué esto? Hay dos razones. Una es que `charAt()` es un método llamado, y no meramente una rápida ojeada que trae un carácter a la posición dada.

La segunda razón para una más lenta performance es que `charAt()` controla que el índice no se salga fuera de los límites.

Porque la llamada al método tiene overhead y el chequeo de los índices también, `indexOf()` gana fácilmente.

Performance – Garbage Collection y Seteo a Nulo

Java tiene Garbage Collection automáticamente en tiempo de ejecución y el sistema reclama dinámicamente memoria que no es usada por un largo tiempo. Por ejemplo:

```
public class test {  
    public void f()  
    {  
        char buf[] = new char[1024];  
        // stuff that uses buf  
    }  
}
```

Cuando el método `f()` existe, el espacio que “buf” usa es un desperdicio, porque sólo la variable local en un stack inválido referencia a ésta. El sistema en tiempo de ejecución puede eventualmente reclamar la memoria.

El garbage colection es automático. Pero algunas veces hay manera de ayudarlo. Consideremos en caso que estamos manejando stack de referencias de Obejct:

```
public class Stack {  
    private static final int MAXLEN = 10;  
    private Object stk[] = new Object[MAXLEN];  
    private int stkp = -1;  
  
    public void push(Object p){  
        stk[++stkp] = p;  
    }  
  
    public Object pop(){
```



```
        return stk[stkp--];  
    }  
}
```

Consideremos el caso donde el stack tiene dos elementos, y que se saca uno de ellos. A este punto `stk[0]` tendrá un elemento válido en él, y `stk[1]` tendrá el elemento recién sacado. Esto es, `stk[1]` tendrá una referencia a un Object, el cual puede ser referenciado por alguien, incluyendo una larga estructura de datos de unos miles de bytes. En tal caso, esta estructura no puede ser garbage collected, aunque pueda no estar mucho tiempo en uso.

Para remediar este problema, podemos reescribir el método `pop()` como:

```
public Object pop(){  
    Object p = stk[stkp];  
    stk[stkp--] = null;  
    return p;  
}
```

No hemos invalidado el Object, solo no hay una referencia válida a éste. El Stack puede tener un largo tiempo de vida, y reescribiendo el método `pop()` de esta forma ayudamos a que el garbage collection pueda ser realizado de una manera más oportuna.



Anexo III

Manual de instalación

Servidor

Instalación de los Programas y Librerías

Para poder instalar los programas y librerías provistas en el disco de instalación, se debe ir al directorio [Drive]:\Tesis\ServerAgent y ejecutar el programa de instalación **ServerAgentInstall.bat**. El mismo crea algunos directorios en el disco “C” y luego instala los programas y librerías necesarias para que ejecute la aplicación del Agente de Servicio. Por último, el instalador registra información de Java necesaria para la ejecución de la aplicación.

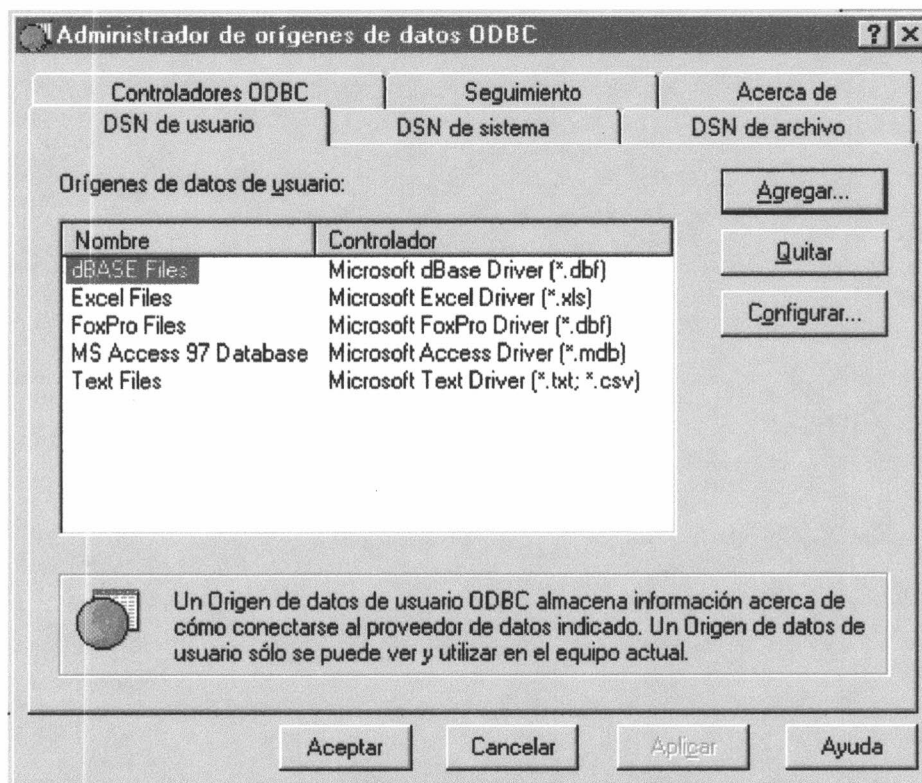
Base de Datos

Instalación del Motor de la Base de Datos

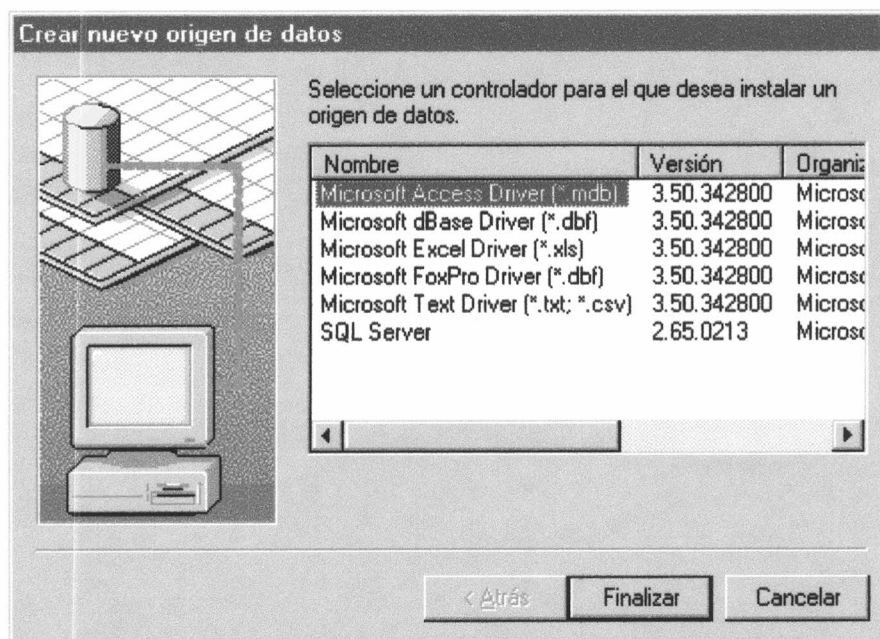
Para instalar la base de datos en un ambiente Windows 95 donde el motor de la base de datos es Access, se debe ir al panel de control



Luego, elegir el controlador de ODBC de 32 bits, y dando un doble click sobre dicho icono, aparecerá la ventana que accede al Administrador de orígenes de datos ODBC; posicionarse sobre la solapa DNS de usuario como muestra la siguiente figura:



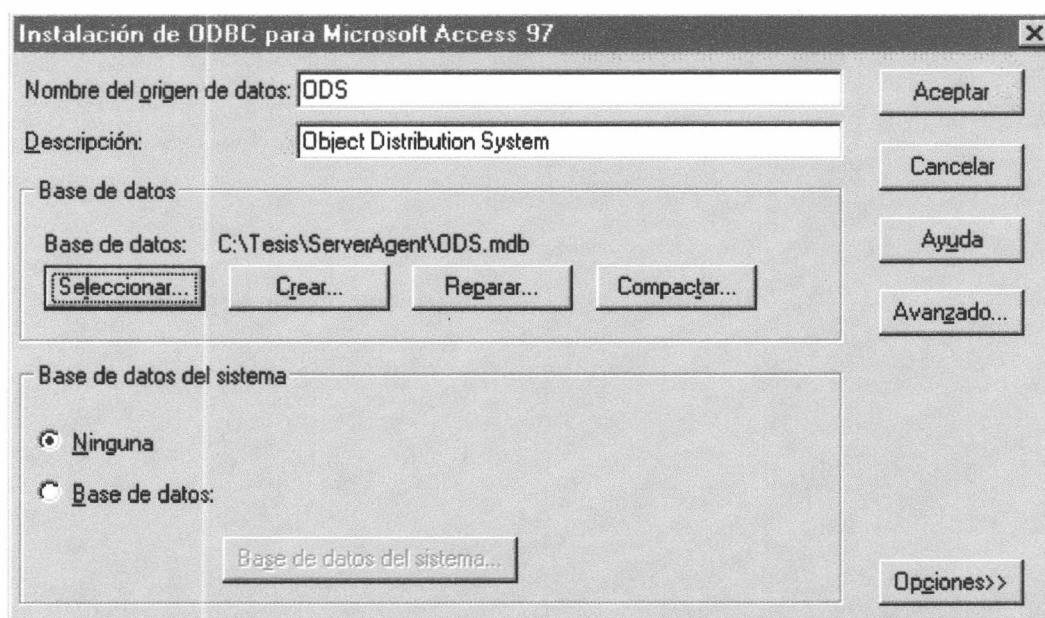
Una vez realizada esta operación, presionar el botón “Agregar”, permitiendo pasar a la ventana siguiente:



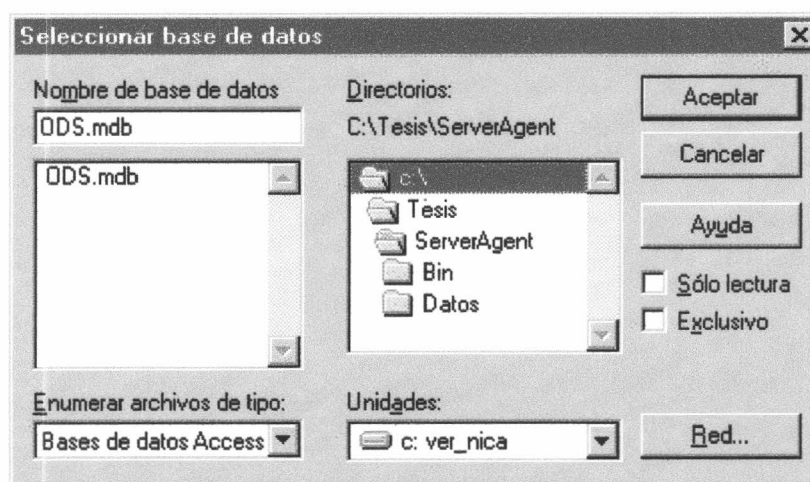
El cual permite agregar un nuevo origen de datos, para nuestro caso, se debe seleccionar “Microsoft Access Driver (*.mdb)” y luego presionar el botón “Finalizar”.



Una vez, realizado esto, aparece una ventana que permite indicar el nombre de la base de datos y donde se encuentra la misma. Por ejemplo:



En el casillero “Nombre del origen de datos:”, ingresar **ODS**, en el casillero “Descripción” ingresar **Object Distribution System**. Ahora se debe indicar donde se encuentra la base de datos, para ello, presionar el botón “Seleccionar”, luego aparece un menú de diálogo que por medio del cual se puede buscar donde está almacenada la base de datos. Por defecto, la misma se almacena en c:\Tesis\ServerAgent\ODS.mdb como muestra la siguiente figura:



Una vez concluida esta operación, ya quedó establecida la base de datos para el controlador de ODBC 32 bits.



Configuración de las Tablas de la Base de Datos

- **Usuario**

Esta tabla contiene los nombres de los usuarios que ingresarán al sistema. En la instalación de la base, esta tabla contiene 5 usuarios, y los mismos son:

Usuario	Password
Usuario1	Usuario1
Usuario2	Usuario2
Usuario3	Usuario3
Usuario4	Usuario4
Usuario5	Usuario5

Para poder cambiar o agregar usuarios, ingresar al Access y abrir la tabla usuarios, y luego efectuar los datos necesarios.

- **Grupo de interés (grupo_interes)**

Esta tabla contiene los grupos de interés que están habilitados en el presente Agentes de Servicio. Esta tabla está compuesta de dos campos, uno de ellos, id_gi, que indica el número de grupo de interés. Este número, debe representar al mismo grupo de interés en todos los Agentes de Servicio. En la base de datos provista en la instalación ya tiene establecidos algunos grupos, como por ejemplo:

Id_gi	Nombre
1	Grupo de Redes
2	Grupo de Java
3	Grupo de Imágenes
4	Grupo de Inteligencia Artificial
5	Grupo de Matemáticas Discreta

- **Usuario – Grupo de interés (user_ig)**

Esta tabla contiene la relación entre los usuarios y los grupos de interés. En la misma, se establece cual es el perfil del usuario dentro del grupo de interés, es decir, si es productor del grupo –P-, consumidor de información (objetos) –O- o consumidor de meta-información (no tiene acceso al documento físico) –M-. La siguiente tabla, muestra los datos que vienen provistos en el disco de instalación.

Id_user	Id_gi	Rótulo
1	1	P
1	2	P
1	3	P



2	1	P
2	2	P
2	3	P
2	4	M
2	5	O
3	1	P
3	2	M
3	3	P
3	4	M
3	5	P
4	3	O
4	4	M
4	5	O
5	4	P

El usuario debe estar por lo menos asociado a un grupo.

- **Cientes**

Esta tabla contiene los datos de los Agentes de Servicio clientes, es decir, cuales son los clientes a los cuales se les va a enviar información desde el presente Agente de Servicio. La tabla prevista en el disco de instalación está vacía para que el administrador del sistema lo configure de acuerdo a sus necesidades. Por ejemplo, se puede ingresar en esta tabla:

Id_cl	Dir_ip	Nombre
1	10.0.200.33	Cliente1
2	10.0.200.34	Cliente2

- **Cliente - Grupo de interés**

Se debe registrar en este archivo la condición de cada cliente con respecto a los grupos de interés. Se registra el identificador del cliente, identificador del grupo y que tipo de cliente es: productor (PRODUCTOR), consumidor de información (CONS_INFO) o consumidor de meta información (CONS_MINFO). Esto se utiliza para determinar en caso de haber nuevos documentos o actualizados, que tipo de información se le envía a cada cliente.

Hay que tener especial cuidado en la definición de los usuarios y clientes, ya que los usuarios de un Agente de Servicio deben coincidir con la definición de dicho server, porque no puede un Agente de Servicio ser consumidor de meta información en un grupo de interés y un usuario de dicho server ser consumidor de información. Esto implicaría una contradicción, ya que el servidor recibirá solo la meta información y el usuario estaría esperando también el documento.



El resto de las tablas se irán alimentando a medida que se utilice el sistema.

- **Resumen de datos iniciales del sistema**

La tabla siguiente muestra los datos de usuarios, grupos de interés y los perfiles de los usuarios que vienen en el disco de instalación. En base a los datos que figuran en esta tabla se comportará cada usuario en el sistema.

Nombre	Password	Grupo	Productor	Consumidor de información	Consumidor de meta información
Usuario1	Usuario1	Grupo de Imágenes	X		
		Grupo de Java	X		
		Grupo de Redes	X		
Usuario2	Usuario2	Grupo de Imágenes	X		
		Grupo de Java	X		
		Grupo de Redes	X		
		Grupo Inteligencia Artificial			X
		Grupo Matemática Discreta		X	
Usuario3	Usuario3	Grupo de Imágenes	X		
		Grupo de Java			X
		Grupo de Redes	X		
		Grupo Inteligencia Artificial			X
		Grupo Matemática Discreta	X		
Usuario4	Usuario4	Grupo de Imágenes		X	
		Grupo Inteligencia Artificial			X
		Grupo Matemática Discreta		X	
Usuario5	Usuario5	Grupo Inteligencia Artificial	X		

Registración del Sistema

En el disco de instalación viene un archivo que registra el producto de Java. Este archivo se instala automáticamente en la instalación. En caso de que por algún motivo se pierda la registración, el mismo se puede recuperar ejecutando de la línea de comando del DOS desde el disco de instalación

[Drive]:\Tesis\ServerAgent > regedit "RegEdit ServerAgent.reg"

Archivo de Configuración (OpenConfig.ini)

El archivo de configuración contiene información necesaria para que funcione el Agente de Servicio. El mismo contiene las siguientes variables:



Variable=Valor	Descripción	Puede cambiarse?
url=jdbc:odbc:ods	Indica el driver ODBC que maneja la base de datos Access	S
Driver=sun.jdbc.odbc.JdbcOdbcDriver	Indica el driver que usa Java para acceder a la base de datos.	N
SleepInicial=30000	Programa inicial que carga los demás módulos y controla que sigan funcionando. *	S
SleepADC=10000	Administrador de conexiones. *	S
SleepAOI=10000	Administrador de Objetos de Interés. *	S
SleepEOF=10000	Envío de Ofertas. *	S
my_ser_ip=	Indica la dirección IP de la máquina donde se ejecuta el sistema	S
PortADCe=2000	Administrador de conexiones, para enviar. **	N
PortADCr=2001	Administrador de conexiones, para recibir. **	N
PortEOI =2002	Envío de objetos de interés, para enviar y recibir. **	N
PortRDOr=2005	Recepción de Ofertas, para recibir. **	N
PortEDOe=2004	Envío de Ofertas, para enviar. **	N
Trama=50000	Indica el tamaño de la trama entre el Agente de Usuario y el Agente de Servicio	S
Trama_sa=50000	Indica el tamaño de la trama entre los Agente de Servicio. El valor debe ser igual entre los Agente de Servicio que se comunican	S
Remote_directory=c:\\Tesis\\Serveragent\\Datos	Indica donde se guardan los documentos que se generan y que se reciben desde otros Agente de Servicio, es decir, es el lugar físico donde están los archivos. Los archivos se registran con un nombre único que se forma con la dirección IP del server donde se genera el documento más el número de documento que es único en cada server.	S

* - Expresan tiempo de espera en milisegundos.

** - Estas variables indican los ports de comunicaciones que usan los diferentes módulos.

Puntos a Tener en Cuenta

- Se debe actualizar la variable **my_ser_ip** para indicar cual es la dirección del Agente de Servicio. Esta viene en blanco en el disco de instalación.
- No cambiar los port de comunicaciones.
- Se aconseja dejar los tiempos de espera que se especifican por default.
- Verificar las direcciones IP de la máquina propia y de los Agente de Servicio clientes.
- Verificar la relación entre los grupos de interés, los clientes y los usuarios.



- Ante dudas en el funcionamiento verificar el archivo de log. El mismo se encuentra por defecto en c:\Tesis\ServerAgent\Datos\filelog.txt

Ejecución de la Aplicación del Agente de Servicio

La aplicación del Agente de Servicio corre bajo DOS. Para poder ejecutar la aplicación, ir al directorio c:\Tesis\ServerAgent y ejecutar **ServerAgent.bat**.

Para cancelar la ejecución del mismo, desde la ventana de DOS presionar CTRL+C.



Cliente

Instalación de los Programas y Librerías

Para poder instalar los programas y librerías provistas en el disco de instalación, se debe ir al directorio [Drive]:\Tesis\UserAgent y ejecutar el programa de instalación **UserAgentInstall.bat**. El mismo crea algunos directorios en el disco “C” y luego instala los programas y librerías necesarias para que ejecute la aplicación del Agente de Usuario. Por último, el instalador registra información de Java necesaria para la ejecución de la aplicación.

Registración del Sistema

En el disco de instalación viene un archivo que registra el producto de Java. Este archivo se instala automáticamente en la instalación. En caso de que por algún motivo se pierda la registración, el mismo se puede recuperar ejecutando de la línea de comando del DOS desde el disco de instalación

[Drive]:\Tesis\UserAgent > **regedit “RegEdit UserAgent.reg”**

Archivo de Configuración (OpenConfig.ini)

El archivo de configuración contiene información necesaria para que funcione el Agente de Usuario. El mismo contiene las siguientes variables:

Variable=Valor	Descripción	Puede cambiarse?
dir_server=	Indica la dirección URL del servidor donde se quiere conectar el usuario.	S
dias_novedad=7	Indica la cantidad de días que se consideran para determinar si un documento es o no una novedad de acuerdo a los días que pasaron desde su llegada / creación.	S
local_directory=c:\Tesis\UserAgent\local_directoy	Indica donde se guardan los documentos que se reciben del servidor y de donde se eligen los documentos físicos a ingresar al servidor.	S

- Se debe actualizar la variable **dir_server** para indicar cual es la dirección del Agente de Servicio donde se quiere conectar el usuario. Esta viene en blanco en el disco de instalación.

Ejecución de la Aplicación del UserAgent

La aplicación del Agente de Usuario corre bajo DOS. Para poder ejecutar la aplicación, ir al directorio c:\Tesis\UserAgent y ejecutar **UserAgent.bat**.



Para cancelar la ejecución del mismo, en caso de que no se pueda salir por las vías normales del Agente de Usuario, desde la ventana de DOS presionar CTRL+C.



Anexo IV

Manual del Usuario

El agente de usuario es la interfase que le permite al usuario realizar operaciones sobre documentos. Este se encarga de comunicarse con el Agente de Servicio en forma transparente, es decir, el mismo no tiene porque conocer donde reside el Agente de Servicio, logrando así una transparencia de localización.

El Agente de Usuario funciona como un cliente en un sistema Cliente-Servidor. El mismo envía requerimientos al Servidor (Agente de Servicio), éste recibe el mensaje, procesa el mismo (en su mayoría sentencias SQL – Structured Query Language-) y devuelve el resultado de la operación al Cliente (Agente de Usuario). Entre el cliente y el servidor se establece una comunicación punto a punto.

La conexión está activa hasta que el Agente de Usuario cierre la aplicación. En algunos casos, puede llegar a ocurrir que se corte la comunicación mientras el usuario tiene la aplicación abierta; en este caso, cuando el usuario intente realizar un requerimiento al servidor, el cliente intenta restablecer la comunicación y si esto es factible todo sigue normalmente sin que el usuario se haya enterado de la situación. Si la misma no se logró restablecer, se le avisa al usuario de tal situación y éste deberá intentar conectarse más tarde.

El agente de usuario está desarrollado para correr sobre un ambiente Windows 95 / NT.

Cuando el usuario abre la aplicación, debe ingresar su nombre y password, y éstas son validadas por el servidor.

The image shows a Windows-style dialog box titled "Ingreso al sistema". It has a standard title bar with minimize, maximize, and close buttons. Inside the dialog, there are two labels: "Usuario :" and "Password :". Each label is followed by a text input field. At the bottom of the dialog, there are two buttons: "Ok" and "Cancel".

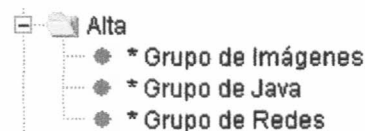
En caso de que el nombre del usuario y la password no fueran válidas, se le avisará al usuario que se ha denegado el acceso y el mismo puede volver a intentarlo nuevamente, con un máximo de tres intentos, llegado a esto, se sale de la aplicación.

Luego, aparecerá un TreeView en el lado derecho de la pantalla,

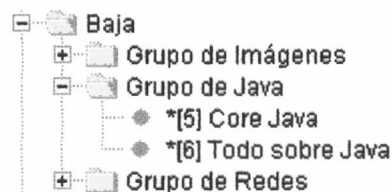


que se utiliza para mostrar las operaciones que puede realizar el usuario sobre los documentos, que esta dado según su perfil. El perfil lo asigna el administrador del sistema.

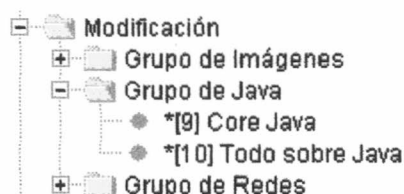
El TreeView tiene una carpeta de *Alta* donde se cargan los nombres de los grupos al cuál el usuario es productor, pudiendo el usuario no tener carpeta de *Alta* si es que el mismo no es productor de ningún grupo. Por ejemplo:



En la carpeta correspondiente a *Baja* se hace referencia a todos los documentos creados por el usuario y los cuales puede eliminar. Para el ejemplo siguiente, el usuario es productor de los grupos, “Grupos de Imágenes”, “Grupos de Java” y “Grupo de Redes”, y dentro del grupo “Grupo de Java”, el usuario creó los documentos “Core Java” y “Todo sobre Java” con anterioridad. Al dar de baja el documento, es mismo deja de estar disponible tanto para el propio dueño como para los demás usuarios que son consumidores.



El usuario puede modificar los documentos creados oportunamente por él. Al modificar el documento, es mismo aparecerá como novedad para los usuarios que son consumidores. Siguiendo el ejemplo, aparecen las siguientes carpetas:

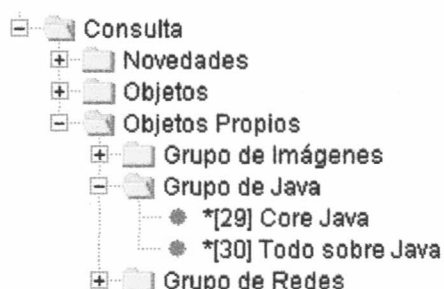


Las consultas que puede realizar el usuario son de documentos pertenecientes a determinados grupos, a los cuales está habilitado como consumidor de información, consumidor de meta información o productor. Los objetos que están en la carpeta de “*Novedades*” son aquellos que fueron creados o modificados dentro de un determinado



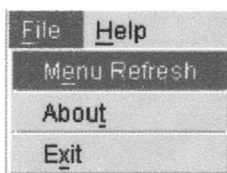
tiempo. Este tiempo se toma de un archivo de parámetros que se lee cuando se abre la aplicación, e indica la cantidad de días que puede tener el documento desde su creación o modificación. Si la fecha de creación o modificación con respecto a la actual (fecha del servidor) está dentro de los días indicados en el archivo de configuración, el documento será mostrado en la carpeta “*Novedades*”, dentro del grupo al cual pertenece el documento y al que el usuario tiene acceso. Estos documentos pueden ser creados o modificados por otros usuarios que pertenecen al mismo servidor donde se conecta el usuario o a otro servidor.

En la carpeta “*Objetos*” están todos los documentos donde el usuario tiene acceso para consulta y que no son propios. Aquí también aparecen los que figuran en la carpeta de “*Novedades*” y que no son propios. En la carpeta “*Objetos Propios*”, figuran todos los documentos creados por el propio usuario. En la opción de consulta sólo puede extraer el documento físico, excepto si es el usuario es consumidor de metainformación, y ver todos los datos existentes en los respectivos “*tabs*”.

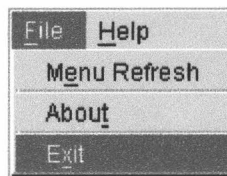


Cabe recordar que de acuerdo al perfil del usuario y a los documentos disponibles en ese momento en el servidor, es como se arman las carpetas, pudiendo en algún momento no tener todas las carpetas posibles.

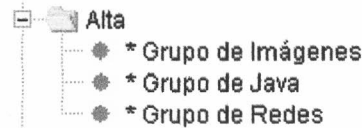
Mientras el usuario realiza operaciones, otros usuarios pueden actualizar sus propias documentos, y estos cambios no se ven reflejados en el “*TreeView*”, para evitar esto, hay una opción dentro del menú (File – Menú Refresh) que permite reflejar las novedades existentes en el servidor. Cuando un usuario da de alta, baja o modifica el título de un documento, no se refleja en el “*TreeView*” y para ver estos cambios debe refrescar el “*TreeView*”.



Para salir del sistema se lo puede hacer desde el menú (File – Exit) o dando un click en el extremo superior izquierdo (X).



Para proceder a dar de alta un documento, se debe dar doble click en la carpeta “Alta” o un click en el “+” o un “Enter” en dicha carpeta, logrando la apertura de la misma, donde aparecen los grupos a los cuales el usuario tiene perfil de productor.



Para elegir el grupo al cual pertenece el documento a dar de alta, el usuario debe dar doble click sobre el nombre del grupo. En ese momento se cargarán si es que existen, los nombres de los otros objetos que pertenecen al mismo grupo en la tabla de *documentos disponibles* y que luego el usuario podrá usar si así lo desea para vincular al documento creado por él con estos, como se describe más adelante.

Para la opción de alta, el campo grupo se carga automáticamente, y es necesario ingresar un título, un autor, la versión, el abstract y el documento físico en sí. Los datos de los co-autores y comentarios (comments) son opcionales. Por ejemplo:

Grupo
Grupo de Java
Título
Core Java
Autor
Gary Cornell
Co-Autores
Cay S. Horstmann
Fecha creación
16/04/2000 22:58:00
Versión
1.0

El documento a ingresar puede tener otros documentos que estén vinculados a él. Para ello se debe elegir el “tab” “Documents Compounds”, en el cual aparecen dos tablas, una que corresponde a los documentos disponibles y otra a los documentos ingresados al documento a dar de alta.



En el primero figuran todos aquellos documentos que pertenecen al grupo al cual se quiere dar de alta el documento. Puede ocurrir que dicha ventana esté vacía, y esto es porque el servidor no posee documentos que pertenezcan a ese grupo. Para el caso que ya existan documentos, y se quiere seleccionar uno de ellos, de debe dar doble click sobre el documento elegido o posicionarse en el mismo, el cual se ilumina, y luego presionar al botón de inserción, haciendo para ambos casos que el nombre del documento desaparezca de dicha tabla y aparezca en la tabla de documentos ingresados, quedando de esta manera el documento vinculado.

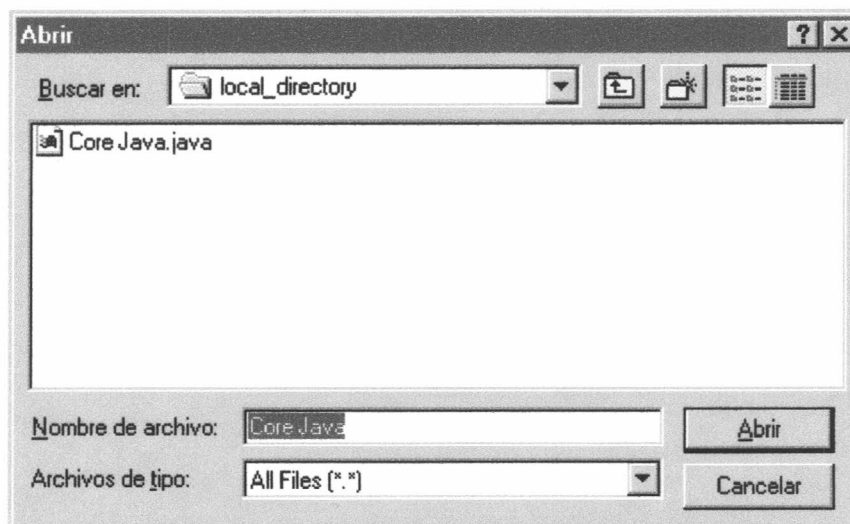
Para el caso de querer desasociar un documento, hacer doble click sobre el documento a desvincular o seleccionar el mismo y presionar el botón de baja, logrando que desaparezca el nombre del documento de la tabla de documentos ingresados y que aparezca en la tabla de documentos disponibles.

Se debe aclarar que un documento puede estar vinculado a varios documentos (siempre perteneciente al mismo grupo), realizando el procedimiento descripto arriba para cada documento al cual se lo desea vincular. Para ingresar físicamente el documento, se debe posicionarse en el “tab” “Property”, pararse en el botón consulta por medio del tabulador (binoculares)





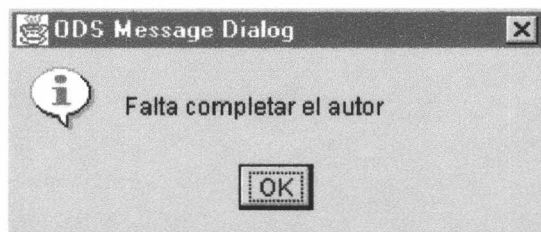
y presionar “Enter” o dar un click sobre dicho botón, y aparecerá un menú de diálogo que le permite al usuario buscar o localizar el documento a dar de alta.



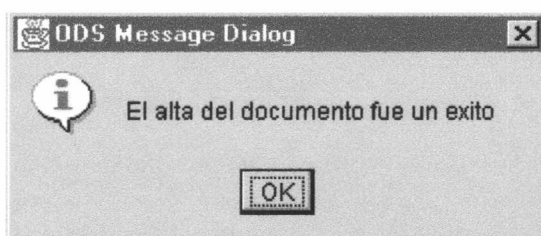
Para ejecutar la operación de alta se debe presionar al botón para este fin,



el cual validará que estén todos datos requeridos como obligatorios y en caso de faltar alguno, se le avisará al usuario los datos faltantes, posicionándose luego el cursor en el dato faltante, como por ejemplo:



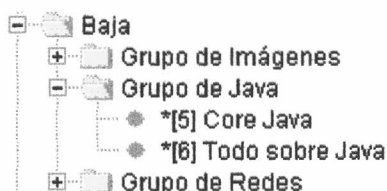
Luego, aparecerá en pantalla el resultado de la operación. Si esta finalizó correctamente aparece el siguiente mensaje:





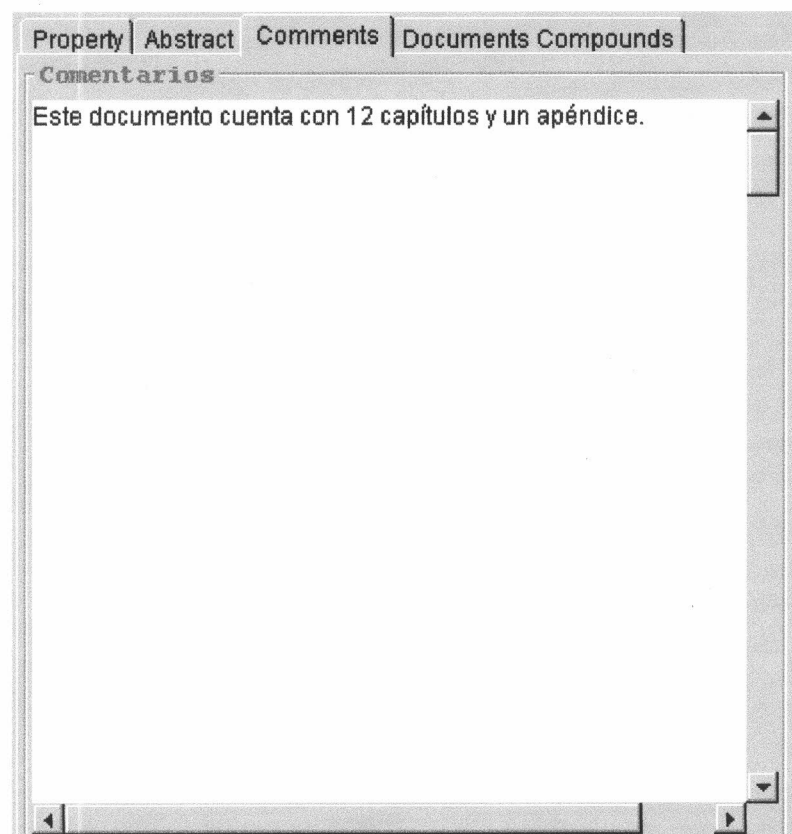
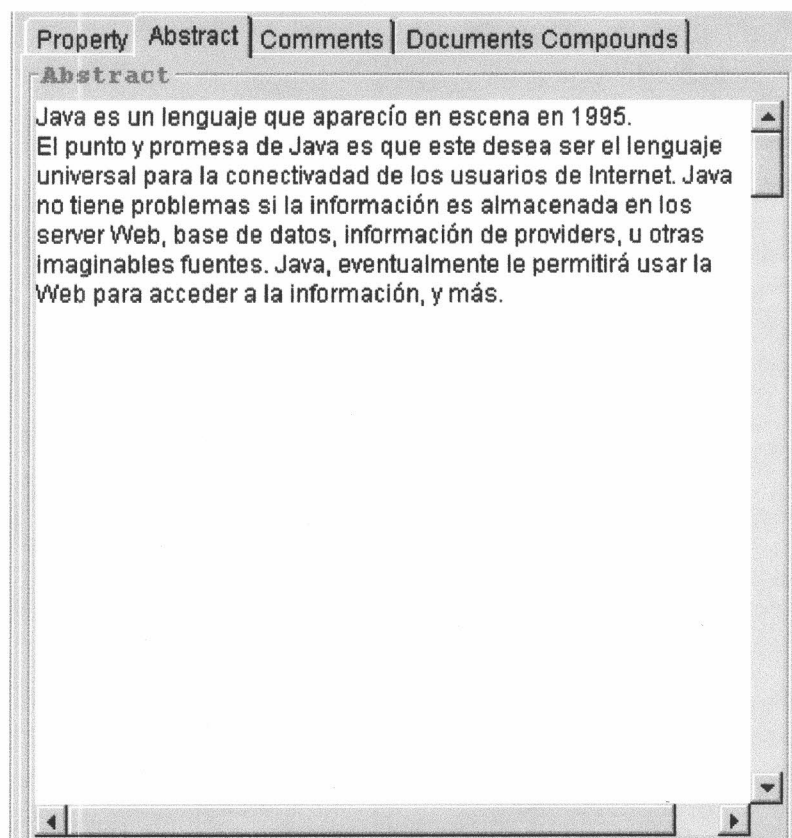
Hay que tener en cuenta que el documento dado de alta no aparecerá en el TreeView hasta que el usuario no refresque el mismo desde la opción del menú.

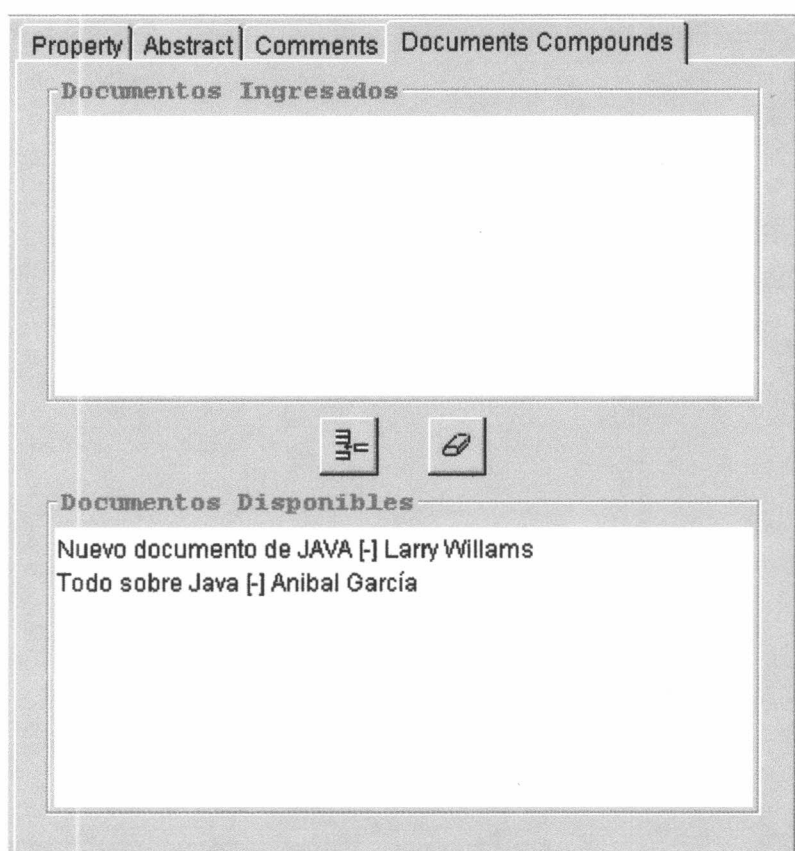
Para proceder a dar de baja un documento, se debe ir a la carpeta de *Baja*, entrar en ella como se describió para el caso del alta, entrar en la carpeta del grupo donde se encuentra el documento a dar de baja, elegir el documento dando doble click sobre el nombre del documento



y si por ejemplo el usuario eligió el documento “Core Java” aparecerá en pantalla los datos del documento que son de tipo obligatorio, como así también los datos opcionales, si es que los mismos fueron ingresados oportunamente, como ser los coautores, el comentario y los documentos vinculados.

Property	Abstract	Comments	Documents	Compounds
Grupo Grupo de Java				
Título Core Java				
Autor Gary Cornell				
Co-Autores Cay S. Horstmann				
Fecha creación 17/04/2000 00:25:21		Fecha recepción 		
Versión 1.0	Extensión java	   		

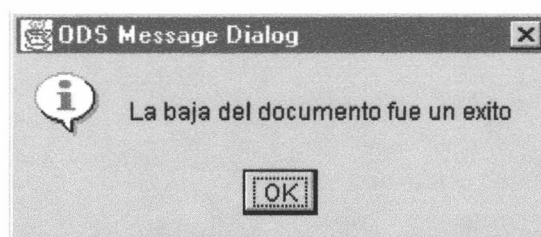




En la opción de baja, toda modificación que se haga no será tomada en cuenta. Se puede extraer el documento si es que el usuario así lo desea. Para darlo de baja, se debe presionar botón para tal fin (goma de borrar),



el cual dará de baja realmente el documento para el usuario. Luego, aparecerá en pantalla el resultado de la operación. Si todo anduvo bien aparece el siguiente mensaje:

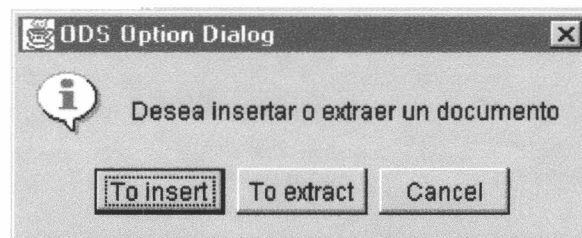


Hay que recordar que este documento no desaparece del TreeView hasta que se refresque el mismo desde el menú.

Para proceder a modificar un documento, se debe elegir uno de la misma manera que se describió para el caso de la baja, pero trabajando sobre la carpeta "Modificación". Se pueden modificar todos los datos del documento, pudiendo vincular otros documentos, abstract, etc.



Para el caso de la versión, la misma se debe modificar pues, esta información indica que hubo una actualización del documento. Se puede también modificar el documento físico, para ello se debe presionar el botón que permite extraer/insertar documentos (binoculares), donde el usuario debe elegir la opción de extraer o insertar,

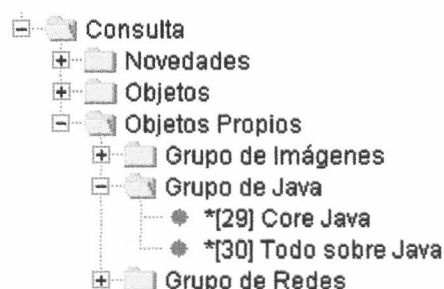


Y si el usuario acepta ingresar el documento aparecerá un menú de diálogo que le permitirá buscar el documento a insertar o buscar el lugar donde se desea dejar el documento físico a extraer. Para realizar realmente la operación de modificación se debe presionar el botón para este fin (plantilla con un lápiz).



En esta opción también se validará que estén completos los datos obligatorios. Luego, aparecerá en pantalla el resultado de la operación. Si en la modificación del documento se cambió el nombre del mismo, éste no aparecerá en el TreeView hasta que se refresque el mismo desde el menú.

Para realizar una opción de consulta, se debe ingresar en la carpeta de consulta, elegir cualquiera de las tres posible carpetas existentes (pueden no estar disponibles todas en ese momento),



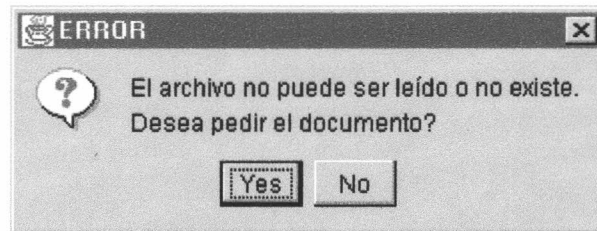
luego elegir el grupo que se desea consultar y después elegir el documento dando doble click sobre el nombre del mismo, cargándose todos sus datos en los tabs correspondientes.

Si el usuario es consumidor de mata información del grupo al cual pertenece el documento elegido, no podrá extraer el documento físico para consultarlo, pero si podrá hacerlo si es consumidor de información o productor del grupo.

En esta opción toda modificación que hace el usuario no tendrá validez. Para poder extraer el documento a consultar se debe presionar el botón de consulta

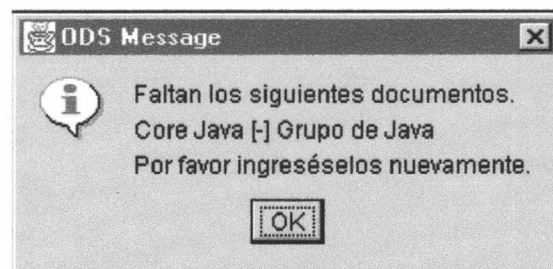


(binoculares), que hace aparecer un menú de diálogo donde el usuario puede buscar el directorio destino y almacenar el archivo en el lugar especificado por el usuario. En caso que al realizar el pedido del documento físico este no esté disponible por algún motivo o no puede ser leído, se le avisará al usuario de tal situación

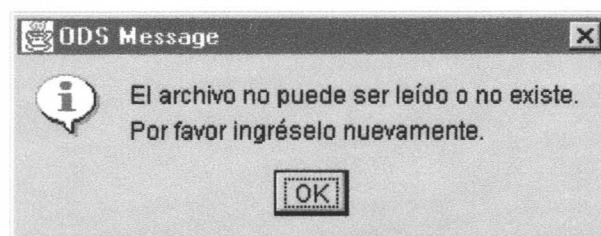


y se le dará la oportunidad de realizar el pedido *fuera de banda*, es decir, si el usuario lo desea genera un pedido especial del documento físico.

Si el documento faltante pertenece al servidor donde se conecta el usuario, entonces al realizarse un pedido *fuera de banda* hará que cuando ingrese al sistema el propio dueño del documento se le avisará que dicho documento está faltante y se le pedirá que lo actualice.



De esta forma el usuario que originó el pedido *fuera de banda* podrá consultar ese documento en otro momento. En caso que el propio dueño de un documento quiera extraer el documento físico y por algún motivo no lo puede hacer el servidor, se le avisará de tal situación y se le pedirá al usuario que actualice el mismo, no generando un pedido *fuera de banda*.

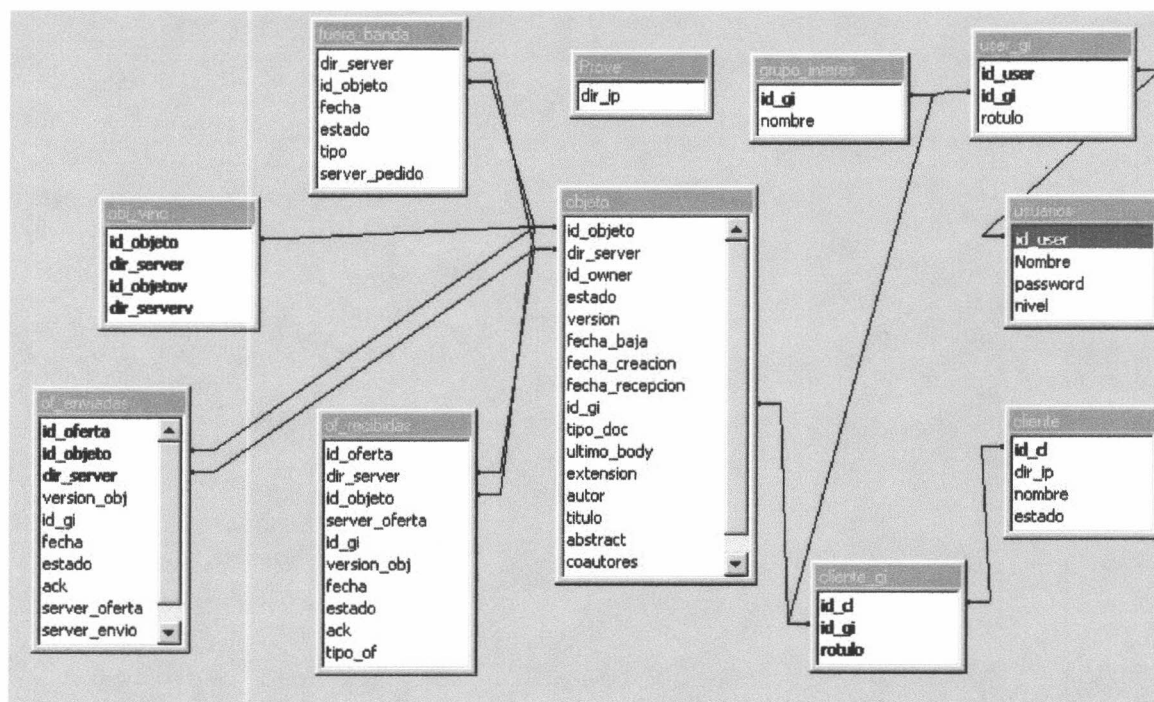




Anexo V

Base de Datos

Diagrama Entidad Relación



Descripción de las Tablas del Sistema

Cientes: registro de los Agentes de Servicio a los cuales se les deben enviar las ofertas y objetos. En esta tabla también se indica el estado de los clientes, para saber si están activos o inactivos.

Cliente_gi: Cliente-Rotulo-Grupo: relación entre los Agentes de Servicio, el grupo al que pertenece y el tipo de información que se deben intercambiar entre cada uno, pudiendo ser información o solamente meta información.

Fuera_banda: Pedidos fuera de banda: pedidos que se realizan en forma no habitual por algún error en la comunicación o por pérdida del documento físico en algún Agente de Servicio.

Grupo_interes: Grupos de Interés: diferentes grupos de interés que se encuentran en el Agente de Servicio. Esta tabla es la referencia que utilizan tanto los Servidores para enviar documentos como los usuarios para acceder e introducir los mismos.



Objeto: contiene los datos de los objetos. En esta tabla se almacena toda la información correspondiente a la meta información del objeto y los campos de control, como así también la referencia al documento físico.

Obj_vinc: Objetos Vinculados: relaciona los documentos compuestos con sus componentes. Se registra para cada documento compuesto la referencia a todos los otros documentos que lo componen.

Of_enviadas: Ofertas Enviadas: registra las ofertas enviadas y por enviar, en este archivo se guardan los datos básicos para identificar al objeto, su grupo de interés y la versión para poder realizar el intercambio de información con otros Agentes de Servicio.

Of_recibidas: Ofertas Recibidas: registra las ofertas recibidas con sus datos básicos para control de los objetos que pueden ser solicitados.

Prove: Proveedores: tabla temporal que registra los proveedores para enviarles la respuesta de los keep alive enviados por estos, esta tabla se utiliza para mantener activa la conexión con los proveedores del Agente de Servicio.

User_gi: Usuario-Grupo de Interés: relación entre los usuarios y los grupos de interés a los que pertenece, se incluye también que tipo de operación que los usuarios pueden hacer con los documentos de cada grupo.

Usuarios: registro de los usuarios habilitados y sus permisos, se incluye grupos de interés a los cuales pueden acceder y en cuales pueden introducir documentos.