Universidade de Aveiro
Departamento de Electrónica e Telecomunicações

Arquitecturas Distribuídas Cliente/Servidor: CORBA, DCOM e JavaRMI.

Vitor Manuel Gomes Roque

Dissertação de Mestrado apresentada à Universidade de Aveiro
como requisito parcial para a obtenção do grau académico
de Mestre em Engenharia Electrónica e Telecomunicações

AVEIRO
Junho de 1999

Trabalho realizado sob orientação de:

Professor Doutor José Luis Oliveira

Professor Auxiliar do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro

RESUMO

Na era das comunicações e da multimédia, as Arquitecturas Distribuídas Cliente/Servidor têm vindo a ganhar cada vez mais representatividade no panorama actual do desenvolvimento de aplicações. Verbas avultadas têm sido investidas pelas empresas desenvolventes deste tipo de tecnologia de forma a melhorar o mais possível e no menor espaço de tempo as suas plataformas com o intuito de ganhar a maior representatividade possível na área da computação distribuída.

A presente dissertação centra-se precisamente na análise de três das mais importantes arquitecturas distribuídas cliente/servidor disponíveis actualmente no mercado, nomeadamente as arquitecturas CORBA, DCOM e JavaRMI. A escolha destas arquitecturas não foi aleatória e teve como base a sua importância no mercado actual. A primeira - a CORBA - por ser fomentada pelo OMG, organização internacional formada por mais de 800 membros, a segunda - o DCOM - pela importância da empresa que a desenvolve, a Microsoft, devido ao peso desta no mundo da informática e a terceira - a JavaRMI - pela crescente popularidade da linguagem Java. Nesta análise são focados os pontos mais importantes de cada uma das arquitecturas e como é que estes pontos podem ser factores decisivos na escolha das plataformas por parte das organizações.

Neste sentido, o segundo capítulo desta dissertação faz uma retrospectiva de tecnologias até se atingir o estado actual, as arquitecturas distribuídas cliente/servidor.

Nos terceiro, quarto e quinto capítulos são abordados de forma sucinta as três arquitecturas CORBA, DCOM e JavaRMI respectivamente.

No sexto capítulo é apresentada uma comparação das funcionalidades das três arquitecturas no que respeita a:

- Interoperabilidade.
- Fiabilidade.
- Maturidade da Plataforma.

Finalmente no sétimo capítulo são apresentadas algumas das conclusões retiradas ao longo desta dissertação.

ABSTRACT

In the communication and multimedia era, Distributed Client/Server Architectures has come to have more and more representation in the current panorama of application development. Large amounts have been invested by companies that develop this kind of technology in order to improve their platforms as quickly and as much as possible with the objective of gaining as much representation as possible in the area of distributed computation.

This dissertation is focussed precisely on the analysis of three of the largest distributed client/server architectures currently available on the market, namely CORBA, DCOM, and JavaRMI. The choice of these architectures was not random but, rather, based on their significance on the current market: the first, CORBA, for being supported by the OMG, an international organization of more than 800 members; the second, DCOM, for the significance of the company that is developing it, Microsoft, due to its weight in the computer world; and the third, JavaRMI, for the increasing popularity of Java language. In this analysis, the most important points of each of the architectures are focussed on, exploring how each of these points can be decisive factors in the choice of platforms on the part of organizations.

Accordingly, the second chapter of this dissertation gives a retrospective view of technology up to the current state, distributed client/server architectures.

In the third, fourth, and fifth chapters, the three architectures, CORBA, DCOM and JavaRMI, respectively, are dealt with succinctly.

In the sixth chapter, a comparison of the functionality of the three architectures is presented with respect to:

- Interoperability.
- Reliability.
- Platform Maturity.

Finally, in the seventh chapter, some of the conclusions drawn throughout the dissertation are presented.

AGRADECIMENTOS

Os primeiros agradecimentos vão para a minha mulher e para a minha filha, como recompensa pelos longos períodos em que foram privadas da minha companhia. Nos momentos em que consegui partilhar a sua companhia, encontrei sempre a mesma coragem, determinação e um invejável sorriso, tendo recebido os indispensáveis incentivos e a cura para os meus desalentos. À Fáty e à Maria dedico esta minha dissertação...

Não posso também deixar de agradecer as preciosas sugestões, comentários e correcções, apresentadas pelo meu orientador, Professor José Luis Oliveira, bem como os conhecimentos que dele recebi.

Vai ainda um agradecimento para os meus colegas de Mestrado, por todas as partilhas de experiências e conhecimentos, e pela permanente boa disposição.

Gostava também de agradecer aos meus Pais, Irmãos e Sogros, que sempre me incentivaram e motivaram para a realização deste trabalho.

Por último, agradeço a muitos outros amigos, que deixo no anonimato, por não querer correr o risco de esquecer algum.

A todos, um sincero obrigado.

LISTA DE ACRÓNIMOS

- ADS - Active Directory Service.
- AI - Application Interface.
- ANSI - American National Standards Institute.
- API - Application Programming Interface.
- BOA - Basic Object Adapter.
- CCS - Concurrency Control Service.
- CF - Common Facilities.
- CICS - Customer Information Control System.
- CLSID - Class Identifier.
- COBOL - Common Business Oriented Language.
- COM - Component Object Model.
- CORBA - Common Object Request Broker Architecture.
- COS - Common Object Services.
- COSS - Common Object Services Specification.
- CPU - Central Processing Unit.
- CS - Collections Service.
- DCE - Distributed Communication Environment.
- DCE-CIOP- Distributed Communication Environment - Common Interoperability Protocol.
- DCOM - Distributed Component Object Model.
- DDE - Dynamic Data Exchange.
- DEC - Digital Equipment Corporation.
- DI - Domain Interfaces.
- DII - Dynamic Invocation Interface.
- DLL - Dynamic Link Library.
- DNS - Domain Name System.
- DSI - Dynamic Skeleton Interface.
- EJB - Enterprise Java Beans.
- ES - Event Service.
- ESIOP - Environment-Specific Inter-ORB Protocol.

- EXS - Externalization Service.
- FORTRAN- Formula Translator.
- GIOP - General Inter-ORB Protocol.
- GUID - Globally Unique Identifier.
- HP - Hewlett-Packard.
- ID - Identifier.
- IDL - Interface Definition Language.
- IEEE - Institute of Electrical and Electronics Engineers.
- IID - Interface Identifier.
- IIOP - Internet Inter-ORB Protocol.
- IMDB - In-Memory DataBase.
- INS - Interoperable Naming Service.
- IOR - Interoperable Object Reference.
- IP - Internet Protocol.
- IR - Interface Repository.
- ISO - International Organization for Standardization.
- ITU - International Telecommunications Union.
- JavaRMI - Java Remote Method Invocation.
- JDK - Java Development Kit.
- JMS - Java Messaging Service.
- JNDI - Java Naming and Directory Interface.
- JNI - Java Native Interface.
- JTS - Java Transaction Service.
- JVM - Java Virtual Machine.
- LAPD - Lightweight Directory Access Protocol.
- LCS - Life Cycle Service.
- LOA - Library Object Adapter.
- LS - Licensing Service.
- MSCS - Microsoft Clustering Technology.
- MSMQ - Microsoft Message Queue Server.
- MTS - Microsoft Transaction Server.
- NDS - Novell Directory Services.
- NIS - Network Information Service.
- NS - Naming Service.
- NSCS - National Computer Security Center.
- NTDS - Windows NT Directory Service.
- OA - Object Adapter.
- ODBMS - Object Database Management System.
- OLE - Object Linking and Embedding.

- OMA - Object Management Architecture.
- OMG - Object Management Group.
- OODA - Object Oriented Database Adapter.
- OQL - Object Query Language.
- ORB - Object Request Broker.
- OS - Object Services.
- OSF - Open Software Foundation.
- PC - Personal Computer.
- PDS - Persistent Data Service.
- PID - Persistent Identifier.
- PO - Persistent Object.
- POA - Portable Object Adapter.
- POM - Persistent Object Manager.
- POO - Programação Orientada por Objectos.
- POS - Persistent Object Service.
- PS - Property Service.
- QoS - Quality of Service.
- QS - Query Service.
- RDBMS - Relational Database Management System.
- RFP - Request For Proposals.
- RMP - Remoted Method Protocol.
- RPC - Remote Procedure Call (DCE).
- RRL - Remote Reference Layer.
- RS - Relationship Service.
- SCM - Service Control Manager.
- SENS - System Event Notification.
- SQL - Structured Query Language.
- SS - Security Service.
- SSL - Secure Sockets Layer.
- Sun - Sun Microsystems.
- TCP - Transport Control Protocol.
- TCP/IP - Transport Control Protocol/Internet Protocol.
- TI - Tecnologias de Informação.
- TRS - Transaction Service.
- TS - Time Service.
- TS - Trader Service.
- UDP - User Datagram Protocol.
- UDT - Uniform Data Transfer.
- URL - Uniform Resource Locator.

- UUID - Universal Unique Identifier.
- X/Open - The Open Group.

ÍNDICE

LISTA DE FIGURAS

CAPÍTULO 1

1. INTRODUÇÃO

1.1. INTRODUÇÃO

Assiste-se actualmente a uma revolução tecnológica impulsionada fundamentalmente pelas Tecnologias de Informação e Telecomunicações. Associada a esta revolução e como resultado, assiste-se também a uma alteração dos hábitos de trabalho, lazer e mesmo de pensar.

Os recursos humanos passaram a ser considerados o recurso mais importante das organizações, sendo o tempo que estes gastam na realização das suas tarefas um factor preponderante na competitividade das suas organizações. Como tal, a evolução a que se tem assistido tem sido sempre no sentido de se arranjarem formas de se despender cada vez menos tempo na realização das tarefas.

No contexto de desenvolvimento de software uma dessas formas foi a necessidade de associar numa única arquitectura diversas abstracções tais como: objectos, modelo cliente/servidor, plataformas distribuídas, reutilização, portabilidade entre outros.

Desta conjugação resultaram várias arquitecturas de onde se destacam as arquitecturas CORBA (*Common Object Request Broker Architecture*), DCOM (*Distributed Component Object Model*) e JavaRMI (*Java Remote Method Invocation*), que vieram motivar uma alteração radical no conceito de desenvolvimento de aplicações em todas as áreas de aplicação.

Nesta dissertação vão ser discutidos os pontos mais importantes de cada uma destas arquitecturas, tendo havido a preocupação de na sua descrição utilizar uma linguagem o mais simples possível, para que esta dissertação possa ser utilizada por todos aqueles que sem grandes conhecimentos sobre este assunto e que pretendam, nas suas organizações, fazer a implementação sobre este tipo de solução encontrem neste documento uma boa base de trabalho.

1.2. ESTRUTURA DA DISSERTAÇÃO

Tendo como objectivo enquadrar os leitores nesta área temática, esta dissertação dedica o segundo capítulo à apresentação de uma breve

retrospectiva de tecnologias até se atingir o estado actual das arquitecturas distribuídas cliente/servidor.

Os terceiro, quarto e quinto capítulos à apresentação do estado actual e futuro de desenvolvimento das arquitecturas cliente/servidor CORBA, DCOM e JavaRMI. Objectivos, vantagens, características e cenário de evolução, são alguns dos aspectos abordados em cada um destes capítulos que pretendem fornecer uma visão tão completa quanto possível destas arquitecturas.

No sexto capítulo é feita uma comparação entre as três arquitecturas referidas onde se referenciam alguns dos aspectos que devem ser tidos em consideração quando se pretender fazer a implementação de arquitecturas cliente/servidor nas organizações (em especial as arquitecturas CORBA, DCOM ou JavaRMI).

Por fim, no sétimo capítulo são apresentadas as conclusões resultantes deste trabalho.

1.3. NOTAÇÃO UTILIZADA

Ao longo de todo o texto desta dissertação irão aparecer termos em inglês cuja tradução para português não existe, não reflecte o seu real significado ou não é universalmente aceite. Esta situação deve-se fundamentalmente ao facto de a literatura existente nesta área ser publicada em língua inglesa. Nestes casos, os termos são apresentados em caracteres itálicos. Sempre que possível são também utilizadas algumas traduções apropriadas ou que já se encontram enraizadas no Português.

Para evitar a repetição de longas expressões técnicas, que tornariam fastidiosa a leitura desta dissertação, são também utilizados acrónimos ao longo de todo o texto. Para além de serem apresentados no início deste documento, a correspondência entre os termos técnicos e os respectivos acrónimos é sempre feita na primeira ocorrência do acrónimo no texto.

Todas as referências bibliográficas utilizadas ao longo deste texto são evocadas entre parêntesis rectos (da forma [nomeautor - ano] ou [nomeautor]) e apresentadas no final deste documento.

CAPÍTULO 2

2. PERSPECTIVA HISTÓRICA

As TI (Tecnologias de Informação) têm sofrido transformações muito rápidas nas últimas décadas (Figura 1). Sistemas manuais deram origem a sistemas computacionais automatizados tendo o processamento de informação passado a ser feito em sistemas *batch*. Quando os CPUs (*Central Processing Unit*) se tornaram mais potentes, o processamento da informação passou a ser realizado por sistemas *on-line*. Quando os CPUs se tornaram mais baratos, os sistemas pessoais tornaram-se mais representativos realizando cada vez mais processamento. Esta última evolução motivou o aparecimento de várias ilhas de informação isoladas dentro das empresas, com deficiências quer ao nível da interactividade quer ao da acessibilidade. A tendência seguinte foi para a integração surgindo os sistemas distribuídos.

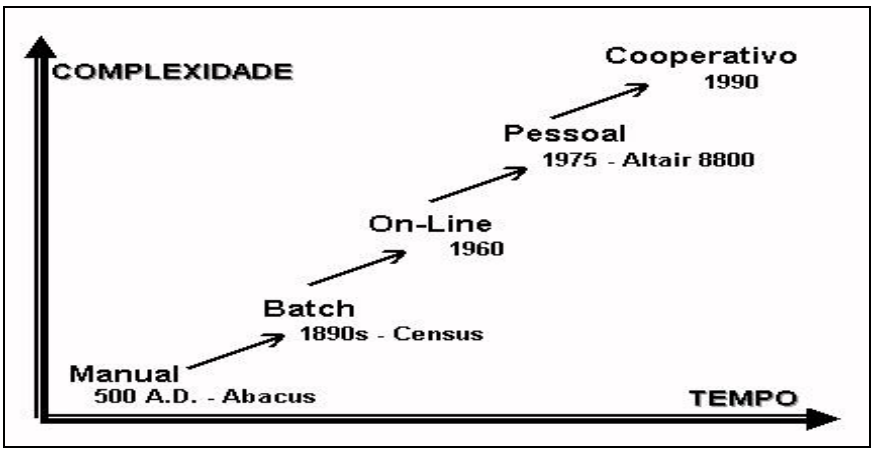

Figura 1 - Tipos de computação [Pope - 1997].

Muitas das alterações nos métodos de processamento da informação ocorreram como resultado dos progressos verificados no campo do hardware.

Quanto ao software este tem tido uma evolução constante, muitas vezes motivada pela capacidade de utilizar de modo mais eficiente o hardware disponível que cada vez apresenta melhor desempenho e é mais sofisticado. Existe uma máxima bem conhecida que diz que "quando a capacidade do hardware aumenta, as aplicações são desenvolvidas de forma

a consumir essa capacidade" [Pope - 1997]. Esta evolução pode ser caracterizada por quatro fases bem diferenciadas; uma primeira fase dos interruptores, uma segunda fase do código máquina, uma terceira fase do *assembler* e uma quarta fase das linguagens de programação (Figura 2).

Devido à sua complexidade os interruptores e a linguagem máquina eram de difícil compreensão e utilização para os operadores/programadores. A dificuldade era de tal forma evidente que era extremamente complicado fazer um simples programa que calculasse, por exemplo, a soma de dois números.

A representação menmónica das instruções máquina pelo *assembly* era bastante mais fácil de apreender pela mente humana, o que tornou a escrita de programas um pouco mais simples do que no caso apresentado anteriormente.

O aumento crescente da complexidade das aplicações suscitou sucessivamente novas necessidades junto dos sistemas de programação tendo motivado o aparecimento das linguagens de procedimentos no início dos anos 60, como é o caso do FORTRAN (*Formula Translator*). O paradigma da programação passa desta forma a centrar-se em procedimentos/funções.

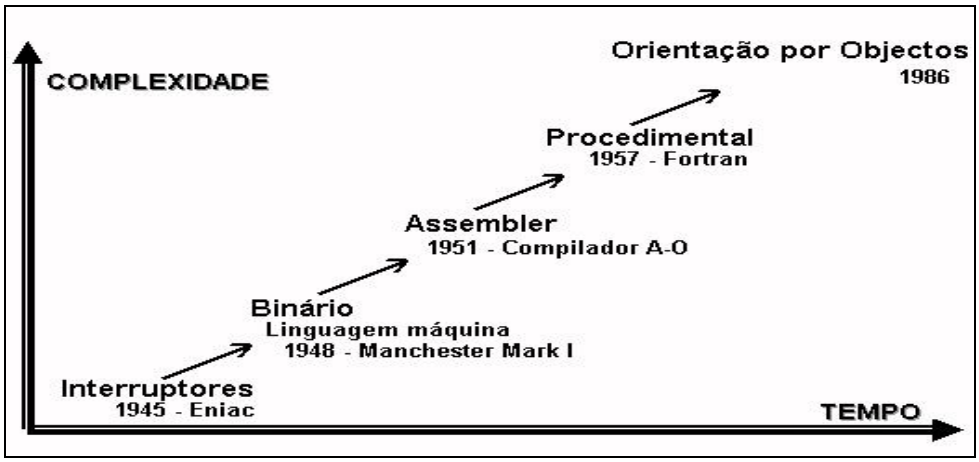

Figura 2 - Complexidade crescente.

Todas as grandes alterações nas tecnologias de processamento de informação motivaram igualmente alterações na gestão dos negócios. Por exemplo sistemas manuais e as suas transacções foram alterados para operações *batch*. Mais tarde estes sistemas tornaram-se mais interactivos, com transacções *on-line* e ambientes *time-sharing*. Desta forma os processos de gestão tornaram-se obviamente também mais interactivos, embora ainda sob um controlo centralizado (Figura 3).

Os PCs (*Personal Computer*) vieram alterar radicalmente o panorama anterior. Com a introdução destes sistemas inicia-se um processo de descentralização, passando grande parte das tarefas que eram realizadas por *mainframes*, a ser realizadas por estes pequenos sistemas. A estruturação, controle e automação do processo de descentralização dá origem ao processamento distribuído[1].

[1] Há 30 anos atrás, M.E. Conway afirmou: "Organizações que façam o desenho de sistemas, estão forçadas a produzir sistemas que são cópias das estruturas de comunicação dessas organizações" [Brooks -1975]

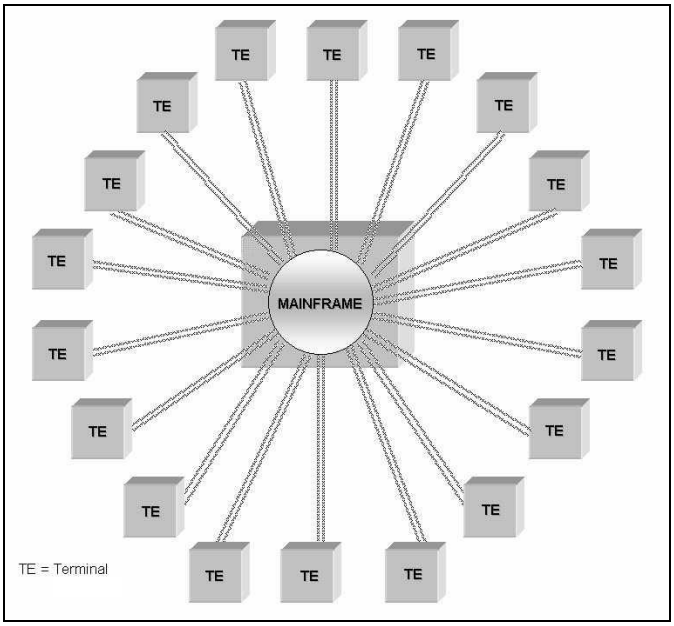

Figura 3 - Arquitectura centralizada.

Surge desta forma o Modelo Cliente/Servidor que tem como impulsionadores por um lado um mercado cada vez mais dominante de computadores pessoais, como referido, e por outro, a necessidade de especialização do software para determinados contextos. Desta forma, termos como *downsizing* e *rightsizing* passam a fazer parte do nosso vocabulário.

O paradigma da POO (Programação Orientada por Objectos) aparece como a etapa seguinte do processo evolutivo. Os objectos chegam aos sistemas de informação como um meio de reduzir a complexidade criada durante os últimos anos da década de 80, no entanto, não ganham representatividade no seio das TI até meados dos anos 90.

A grande vantagem resultante da utilização de objectos tem a ver com a capacidade destes em modelarem o mundo real tal como este é, e agirem como abstracções de entidades e de acções. No paradigma anterior havia sempre a necessidade de uma transformação conceptual prévia entre o espaço-problema e o espaço-computacional. Outras motivações para a utilização de uma abordagem orientada a objectos prendem-se com as características destes [Berard - 1999] [Mens - 1997]:

- Recorrem à mais recente tecnologia de desenvolvimento de software.
- Promovem e facilitam a reutilização de software.
- Facilitam a cooperação entre as diferentes entidades.
- Quando sujeita a uma análise e projecto, a solução é muito parecida com o problema original.
- O software é facilmente modificado, alargado e mantido.

Actualmente é grande a expectativa e também a dependência em torno deste paradigma. Linguagens de programação foram revistas, sistemas operativos redesenhados e bibliotecas refeitas, no processo de transição para objectos. Apesar de inúmeros programadores continuarem a utilizar programação procedimental, dependem quase exclusivamente de linguagens e compiladores orientados por objectos.

Um dos grandes beneficiários da orientação por objectos foi a tecnologia cliente/servidor (Figura 4). Esta tecnologia pode ser definida de uma forma rudimentar como uma transacção económica. Alguém, o consumidor, deseja qualquer coisa que alguém pode fornecer, o produtor. O produtor (servidor) tem a capacidade de produzir qualquer coisa (serviço) que o consumidor (cliente) deseja consumir (utilizar o serviço). O pedido feito pelo consumidor determina a natureza da relação entre produtor e consumidor. Os produtores têm acesso automático aos recursos que necessitam utilizar para satisfazer os pedidos dos clientes. Os consumidores por sua vez, podem estar em qualquer parte, precisando contudo de uma forma de acesso ao produtor.

Geralmente a relação cliente/servidor é uma relação n : 1 (vários clientes para um servidor), no entanto em certas situações os clientes podem ter necessidade de recorrer a mais do que um servidor, servidores estes que podem estar em diferentes localizações. Este tipo de tecnologia possibilita uma maior fiabilidade, especialização e segurança (entenda-se segurança neste caso como acesso a serviços) que a tecnologia baseada numa arquitectura centralizada, permitindo desta forma melhorar o desempenho global do sistema.

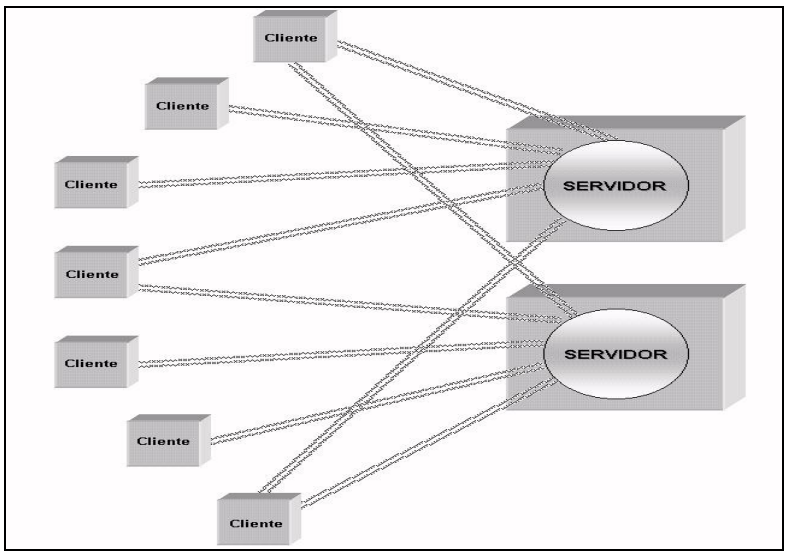

Figura 4 - Arquitectura cliente/servidor.

Contudo as novas soluções costumam introduzir sempre novos problemas. A decisão de conversão para este tipo de arquitectura deve ser fundamentada com uma análise cuidada custo/benefício. Só depois desta análise feita e se os resultados forem positivos, é que se deve passar à fase seguinte. O planeamento cuidado requer que sejam compreendidos todos os riscos e ramificações das alterações de maneira a evitar e desviar os problemas.

Por exemplo, os custos relativamente ao hardware (*mainframe*) numa arquitectura centralizada podem descer na conversão para uma arquitectura cliente/servidor. No entanto os custos associados às comunicações poderão sofrer uma acentuada subida. Os custos relacionados com o desenvolvimento da solução poderão também aumentar e a distribuição introduzirá numa fase inicial na empresa alguns problemas adicionais.

Outro tipo de problemas são os que poderão surgir em consequência da rede que não sendo visíveis exteriormente só são possíveis de ser calculados depois de todas as alterações terem sido realizadas.

A replicação ou a distribuição de informação desactualizada (dados que perderam a sua integridade ou não apropriados, etc.) é outro custo importante que não deve ser esquecido "se ter uma cópia de informação corrompida é mau, multiplique-se agora essa informação por dezenas ou centenas e pode eventualmente ser uma catástrofe" [Pope - 1997].

Existem contudo disponíveis várias tácticas e directrizes (a maioria dos problemas que podem eventualmente surgir já se encontram catalogados com a respectiva solução) que podem ser utilizadas para evitar os problemas descritos anteriormente, no que respeita à distribuição de dados e aplicações [Pope - 1997].

Desta forma com um planeamento e preparação adequados a transição para uma arquitectura cliente/servidor pode ser pacífica, isto é, todos os problemas serem ultrapassados em tempo útil.

Resultante da necessidade de conjugar numa única arquitectura diversos conceitos como objectos, modelo cliente/servidor, distribuíção e outros, surgem os objectos distribuídos. É neste contexto que os próximos capítulos vão ser desenvolvidos, introduzindo os conceitos subjacentes a três tecnologias concorrentes nomeadamente, a CORBA (*Common Object Request Broker Architecture*), o DCOM (*Distributed Component Object Model*) e a JavaRMI (*Java Remote Method Invocation*), e como a introdução deste tipo de tecnologia nos actuais sistemas de informação os pode melhorar.

Quando os computadores se tornarem mais rápidos e com maiores capacidades, novos métodos na definição de problemas e respectivas resoluções aparecerão. No entanto actualmente a metodologia de orientação por objectos continua a ser a melhor aproximação para a resolução dos problemas do mundo real (problemas actuais).

CAPÍTULO 3

3. CORBA - COMMON OBJECT REQUEST BROKER ARCHITECTURE

3.1. INTRODUÇÃO

A arquitectura CORBA (*Common Object Request Broker Architecture*) é uma especificação para uma infra-estrutura de comunicações, tendo a primeira especificação sido realizada pelo OMG (*Object Management Group*) em Outubro de 1991. Surge como uma facilidade de computação *peer-to-peer* onde todas as aplicações são objectos podendo estes alternadamente serem clientes e servidores. Permite a integração de aplicações escritas em diferentes linguagens e a ser executadas em sistemas operativos diferentes. A filosofia de base desta arquitectura é um *bus* de objectos (Figura 5), onde estes podem interactuar uns com os outros e utilizar um conjunto de serviços que o *bus* coloca à sua disposição [Orfali - 1998].

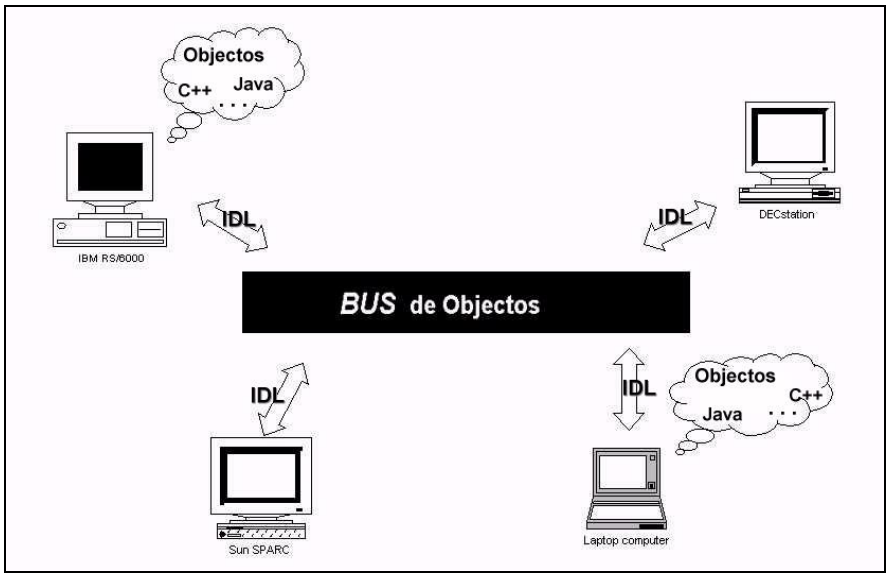

Figura 5 - *Bus* de objectos.

A localização dos objectos é transparente ao utilizador, pois é competência do *bus* encaminhar os pedidos aos objectos correspondentes.

3.2. O OBJECT MANAGEMENT GROUP

O OMG (*Object Management Group, Inc.*) é uma organização internacional formada por mais de 800 membros, incluindo empresas vendedoras de sistemas de informação, empresas de desenvolvimento de software, universidades e utilizadores [OMG - 1998]. É de realçar a participação neste consórcio da Microsoft, mesmo estando a desenvolver uma plataforma concorrente, o DCOM.

Fundado em Abril de 1989 pelas empresas 3Com Corporation, American Airlines, Canon Inc., Data General, Hewlett-Packard, Philips Telecommunications N.V., Sun Microsystems e Unisys Corporation, o OMG promove a teoria e prática da tecnologia de orientação por objectos no desenvolvimento de software. Nas funções desta organização inclui-se o estabelecimento das especificações de gestão de objectos para promoção de uma norma comum para o desenvolvimento de aplicações. Os objectivos primordiais são: reutilização, portabilidade e interoperabilidade do software baseado em objectos em ambientes distribuídos e heterogéneos. De acordo com estas especificações será possível desenvolver um ambiente de aplicações heterogéneo utilizando diferentes plataformas de hardware e sistemas operativos.

O OMG trabalha tendo em conta outros organismos de normalização, onde se incluem: a ISO (*International Organization for Standardization*), o X/Open (*The Open Group*), o consórcio *World Wide Web*, o ANSI (*American National Standards Institute*), o IEEE (*Institute of Electrical and Electronics Engineers*) e o ITU (*International Telecommunications Union*).

3.3. ARQUITECTURA DE GESTÃO DE OBJECTOS

A OMA (*Object Management Architecture*) é a base sobre a qual toda a tecnologia do OMG assenta. Disponibiliza dois modelos fundamentais nos quais a CORBA e outras interfaces normalizadas se baseiam: o Modelo de Objectos e o Modelo de Referência [Vogel - 1997].

O componente principal da OMA é o ORB (*Object Request Broker*). O ORB fornece uma infra-estrutura de comunicações que permite aos objectos comunicar entre si independentemente das plataformas e técnicas utilizadas para a sua implementação. Desta forma, o componente ORB garantirá a portabilidade e a interoperabilidade dos objectos através de uma rede de sistemas heterogéneos.

3.3.1. MODELO DE OBJECTOS

O Modelo de Objectos define quer os conceitos que permitem facilitar o desenvolvimento de aplicações distribuídas recorrendo à utilização do ORB, quer os conceitos de orientação por objectos sobre os quais a norma CORBA é desenvolvida. Os conceitos definidos pelo Modelo de Objectos são [OMA]:

- Objectos - Um objecto pode modelar qualquer tipo de entidade, como por exemplo uma pessoa, um barco, um documento ou outros.
- Operações - As operações são aplicadas aos objectos e permitem concluir coisas específicas acerca do objecto, como por exemplo a determinação da data de nascimento de um objecto pessoa. As operações associadas a um objecto determinam o comportamento desse objecto.
- Tipos - Os objectos são criados como instâncias de Tipos. Os Tipos podem ser encarados como um *template* para a criação de objectos. Uma instância do tipo barco pode ser, barco vermelho com 10m de comprimento e capacidade para 6 pessoas. Um Tipo caracteriza o comportamento das suas instâncias através da descrição das operações que podem ser aplicadas a esses objectos.
- Subtipos ou Supertipos - Ao relacionamento entre Tipos é dada a designação de Subtipos ou Supertipos.

A norma CORBA, segundo [Vogel - 1997], é uma refinação do Modelo de Objectos.

3.3.2. MODELO DE REFERÊNCIA

O Modelo de Referência identifica e caracteriza os componentes, interfaces e protocolos que compõem a OMA. Estão incluídos o componente ORB que possibilita que clientes e objectos comuniquem num ambiente distribuído e heterogéneo e quatro categorias de interfaces para objectos [OMG - 1997]:

- Serviços de Objectos (OS - *Object Services*): São interfaces para serviços gerais que podem ser utilizados em qualquer aplicação de objectos distribuídos (ver ponto 3.11).
- Facilidades Comuns (CF - *Common Facilities*): São interfaces para facilidades horizontais aplicáveis à maioria dos domínios aplicacionais (ver ponto 3.12).
- Interfaces Domínios (DI - *Domain Interfaces*): São interfaces para domínios específicos de aplicações.
- Interfaces Aplicação (AI - *Application Interfaces*): São interfaces não normalizadas específicas para determinadas aplicações.

Estas categorias de interfaces são mostradas na Figura 6.

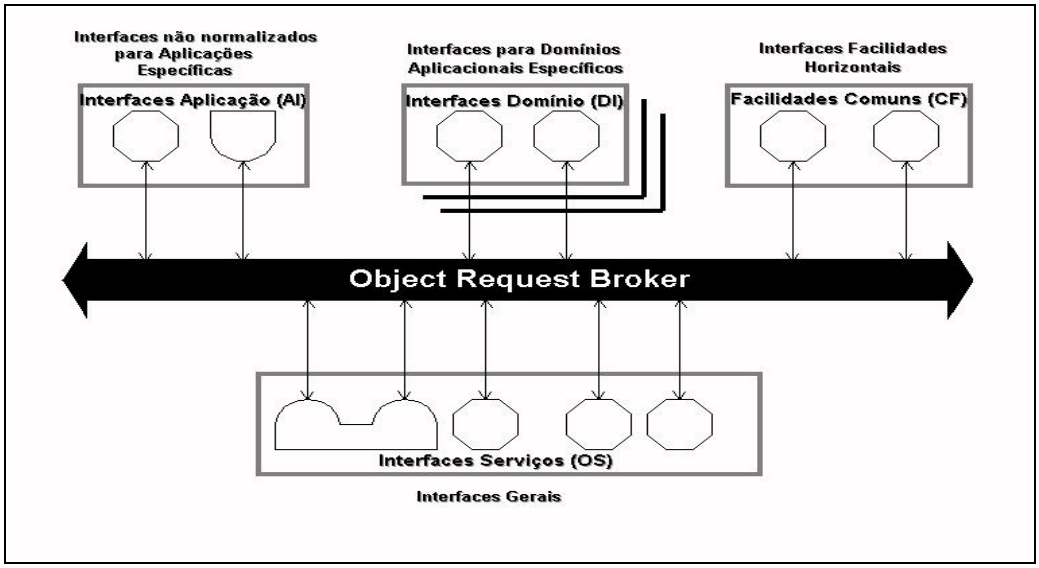

Figura 6 - Modelo de Referência OMA: categorias das interfaces [OMG - 1997].

Note-se que as aplicações necessitam unicamente suportar ou utilizar interfaces compatíveis OMG para participarem na OMA, não necessitando portanto ser construídas recorrendo ao paradigma da orientação por objectos.

SERVIÇOS DE OBJECTOS

Os Serviços de Objectos são serviços de âmbito geral fundamentais quer para o desenvolvimento de aplicações CORBA, quer fornecendo uma plataforma universal para a interoperabilidade entre aplicações.

Estes serviços são os blocos de construção básicos para o desenvolvimento de aplicações de objectos distribuídos. Podem ser utilizados para a construção de facilidades de alto nível e esqueletos de objectos capazes de interoperar através de múltiplas plataformas.

Os serviços adoptados pelo OMG são colectivamente designados por Serviços CORBA (*CORBAservices*) onde estão incluídos os seguintes: Nomes, Eventos, Gestão do Ciclo de Vida, Armazenamento Persistente, Transacções, Controlo da Concorrência, Relações, Exteriorização, Licenças, Associação de Propriedades, Segurança, Tempo, Colecções e Negociação.

FACILIDADES COMUNS

As Facilidades Comuns são interfaces para facilidades horizontais aplicáveis à maioria dos domínios aplicacionais. As Facilidades CORBA adoptadas pelo OMG são colectivamente chamadas Facilidades CORBA (*CORBAfacilities*) onde se incluem as seguintes: Interface Utilizador, Serviço de Gestão da Informação, Serviço de Gestão de Sistemas e Serviço de Gestão de Tarefas.

INTERFACES PARA DOMÍNIOS

Interfaces para Domínios são interfaces específicas para domínios aplicacionais tais como: Finanças, Saúde, Manufactura, Telecomunicações, Comércio Electrónico, Transportes e outros.

INTERFACES APLICAÇÃO

São interfaces não normalizadas específicas para determinadas aplicações.

3.3.3. ESQUELETOS DE OBJECTOS

Ao contrário das interfaces para partes individualizadas da infra-estrutura OMA, os esqueletos de objectos são componentes de alto nível que fornecem funcionalidades de interesse directo aos utilizadores em aplicações particulares ou domínios tecnológicos (Figura 7).

Os esqueletos de objectos são colecções de objectos cooperantes categorizados em: Aplicação, Domínio, Facilidade e Serviço. Cada objecto no esqueleto suporta ou faz uso de alguma combinação de interfaces aplicação, domínio, facilidade comum e serviço.

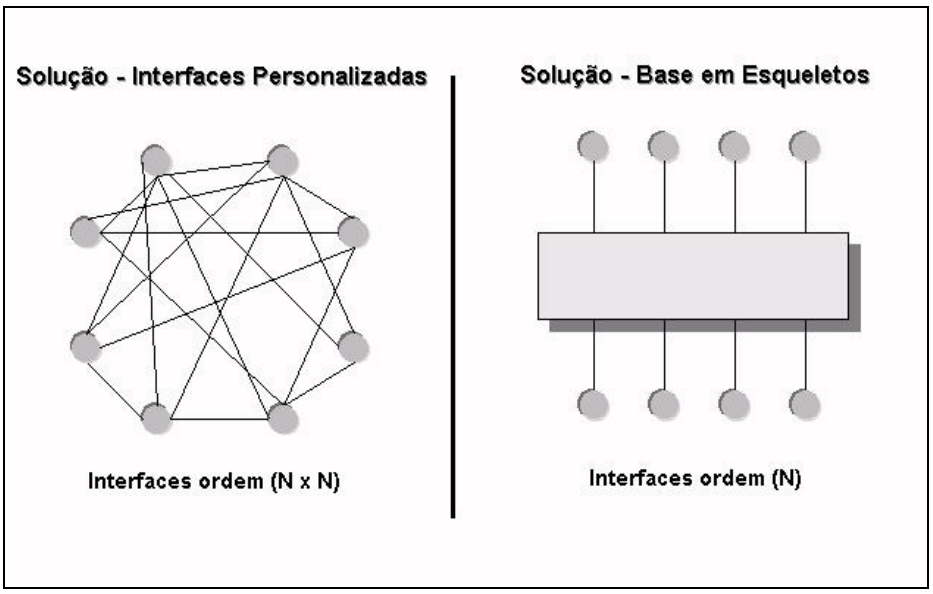

Figura 7 - Interfaces personalizadas vs interfaces baseadas em esqueletos.

Desta forma, um esqueleto de objectos pode conter zero ou mais objectos correspondentes a aplicação, domínio, facilidade e serviço. Assim os serviços suportam as interfaces correspondentes (OS); as facilidades suportam as interfaces correspondentes (possíveis combinações de interfaces Facilidade e herdados de OS) (CF), os domínios suportam as interfaces correspondentes (possíveis combinações de interfaces domínio e herdados de CF e OS) (DI) e para as aplicações a mesma coisa. Deste modo, componentes de alto nível e interfaces são construídos reutilizando componentes de baixo nível.

Na Figura 8 é ilustrado o conceito de esqueleto de objectos. Os objectos são representados pelos círculos interiores (núcleo) sendo limitados por um círculo exterior onde estão representadas as interfaces que esses objectos suportam.

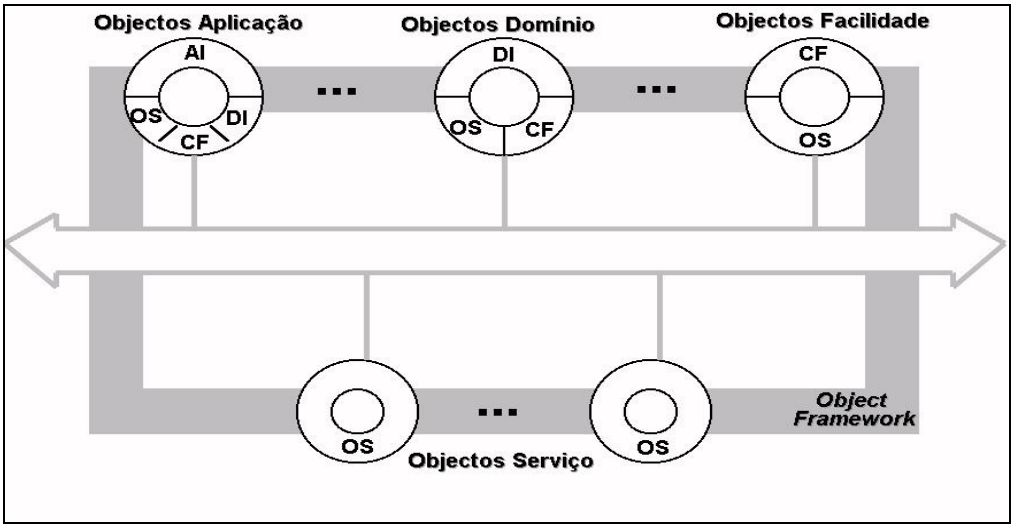

Figura 8 - Modelo de Referência OMA: utilização das interfaces [OMG - 1997].

Neste caso genérico, os objectos suportam todas as interfaces possíveis relacionadas com a sua categoria. Outras situações podem contudo ocorrer, como por exemplo os objectos domínio suportarem unicamente interfaces OS.

A Figura 9 mostra como os objectos num esqueleto de objectos fazem pedidos a outros objectos no mesmo esqueleto de modo a proporcionar uma maior funcionalidade. Na Figura 9 são representados três pedidos: um de um objecto aplicação para um serviço, outro de um objecto facilidade para um serviço e por último de um objecto domínio para uma aplicação. Este último caso pode ser por exemplo uma *callback* para uma interface domínio suportada pela aplicação.

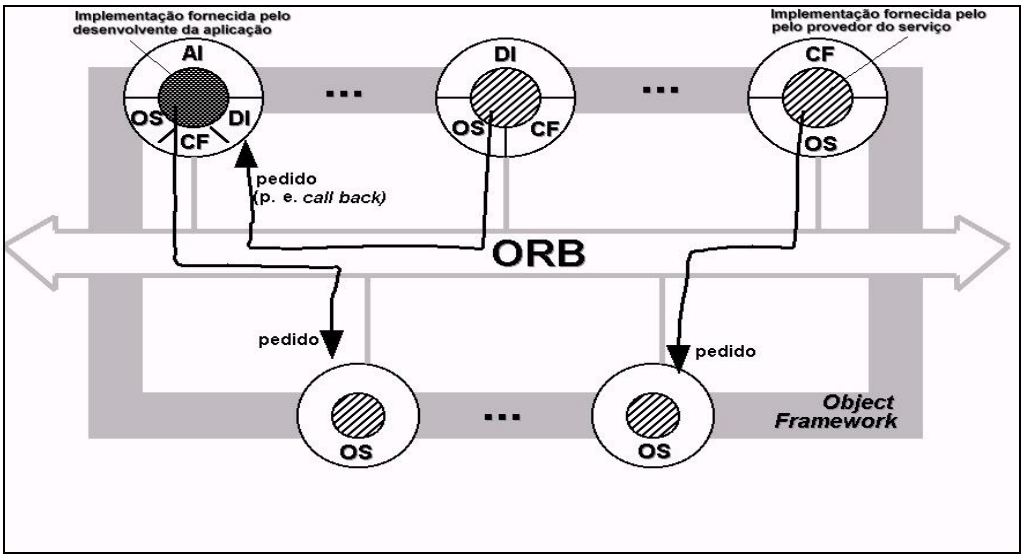

Figura 9 - Exemplo do fluxo de um pedido [OMG - 1997].

A especificação de um esqueleto de objectos define aspectos como: estrutura, interfaces, tipos, sequência de operações e qualidade de serviço dos objectos que constituem o esqueleto. Isto inclui requisitos nas

implementações de forma a garantir a portabilidade e interoperabilidade das aplicações através de plataformas diferentes. As especificações podem inclusive incluir novas interfaces domínio para domínios aplicacionais particulares.

A parte aplicação de uma interface por definição não é incluída na especificação do esqueleto de objectos, uma vez que é específica e totalmente definida pelo programador.

3.4. OBJECT REQUEST BROKER

O ORB (*Object Request Broker*) é um meio de comunicação capaz de estabelecer relações cliente/servidor entre objectos. Comporta-se como um *bus* de objectos (Figura 5) fornecendo os meios através dos quais os objectos conseguem comunicar uns com os outros, de um modo transparente e sem preocupação com a forma como a comunicação é realizada, isto é, o ORB é o responsável pela intercepção da chamada e por encontrar um objecto que possa implementar o pedido, passar-lhe os parâmetros, invocar os seus métodos e devolver os resultados (Figura 10). O cliente não tem que se preocupar com a localização do objecto (na mesma máquina ou numa máquina remota), qual a linguagem de programação em que foi desenvolvido, o seu sistema operativo, ou qualquer outro aspecto de sistema que não seja parte da interface (Figura 11). Note-se que as funções cliente/servidor são unicamente utilizadas para coordenar as interacções entre dois objectos [Orfali - 1998]. Os objectos no ORB podem agir quer como cliente, quer como servidor dependendo da situação.

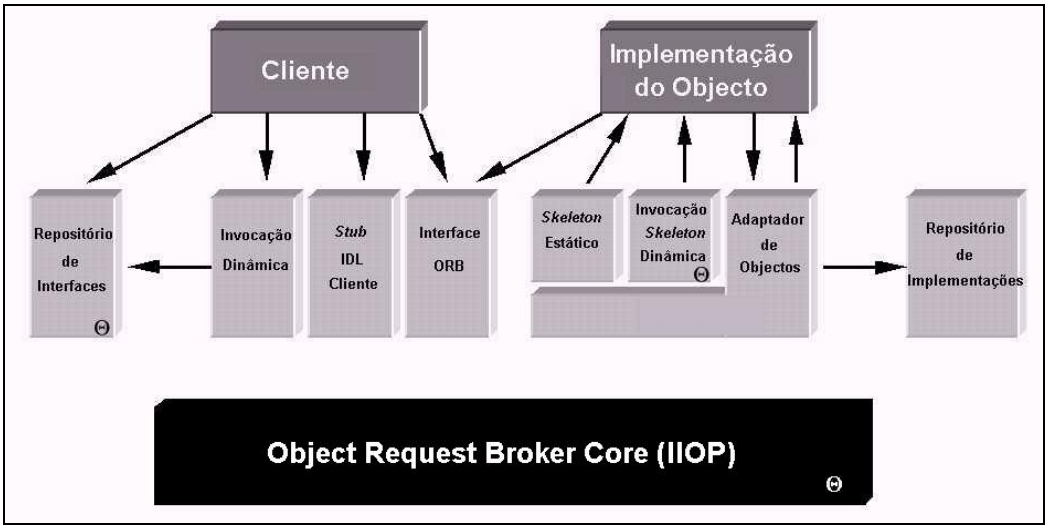

Figura 10 - Arquitectura do ORB.

O ORB disponibiliza um conjunto de serviços distribuídos de comunicação de nível mais elevado aos tradicionalmente utilizados em ambientes cliente/servidor [Orfali - 1998]. Inclui também os meios comuns de comunicação, tais como: RPCs (*Remote Procedure Call*), mensagens, procedimentos para bases de dados e serviços ponto-a-ponto.

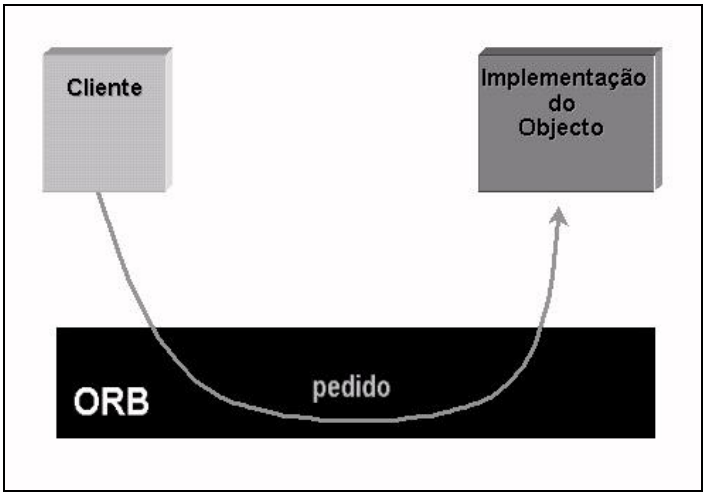

Figura 11 - Envio de um pedido via ORB.

A CORBA pela sua natureza, consegue agrupar as vantagens inerentes à metodologia e programação orientada por objectos, às vantagens dos sistemas distribuídos e abertos. De uma forma resumida as vantagens associadas a esta arquitectura são as seguintes [Orfali - 1998]:

- Invocação estática e dinâmica de métodos. Por intermédio do ORB, a CORBA permite na compilação definir estaticamente a invocação a métodos ou permite que estes sejam encontrados dinamicamente durante a execução. Isto deveu-se ao facto de o OMG ter recebido duas propostas importantes de alteração ao RFP (*Request For Proposals*) do ORB originalmente proposto: uma das propostas pertencia à HyperDesk e DEC (*Digital Equipment Corporation*) e estava fundamentada em APIs (*Application Programming Interface*) dinâmicas e a segunda proposta pertencia à Sun (*Sun Microsystems*) e HP (*Hewlett-Packard*) e estava fundamentada em APIs estáticas. As outras plataformas intermediárias disponíveis apenas suportam ligações estáticas.
- Transcodificação da linguagem de alto-nível. A invocação de métodos de objectos do servidor é realizada na linguagem de programação em que o cliente foi desenvolvido. O ORB permite que os clientes sejam independentes da linguagem de programação do servidor e do seu sistema operativo. Isto é conseguido através da separação da interface da implementação, o que não acontece com as APIs tradicionais.
- Sistema auto-descritivo. A arquitectura CORBA disponibiliza metadados na execução para descrever cada interface do servidor. O ORB deverá assim suportar um repositório de interfaces que contenha informação da descrição das funções que o servidor disponibiliza e os seus parâmetros. Os clientes utilizam os metadados para determinar como invocar serviços durante a execução. Os metadados são gerados automaticamente, ou por um pré-compilador de IDL (*Interface Definition Language*), ou por compiladores que sabem como gerar a IDL directamente a partir de uma linguagem orientada por objectos.
- Transparência local/remota. Um ORB pode ser executado em modo auto-suficiente (*standalone*), em *laptop*, ou interligado a outros ORBs. É da responsabilidade do ORB gerir e executar as chamadas entre objectos num mesmo processo, entre vários processos que estejam na mesma máquina ou em máquinas diferentes. Os

programadores não têm que se preocupar com os problemas de transporte, localização, sistema operativo e plataforma, nem com a activação dos objectos. É da competência do ORB resolver todas estas questões.

- Transacções e Segurança embebidas. O ORB inclui informação adicional nas suas mensagens que lhe permite lidar com as questões de segurança e de transacções.
- Polimorfismo nas mensagens. Contrariamente a outras plataformas intermediárias, o ORB não invoca unicamente uma função remota, invoca sim uma função num objecto alvo. Isto significa que a mesma chamada a uma função, poderá ter efeitos diferentes dependendo do objecto que a recebe. Por exemplo, a invocação a um método *configure_yourself* comporta-se de forma diferente quando aplicado a um objecto base de dados versus um objecto impressora.
- Coexistência com sistemas existentes. A separação que a arquitectura CORBA faz entre a definição de objecto e a sua implementação é útil para o encapsulamento de aplicações já existentes. Utilizando a IDL pode fazer-se com que o código existente se pareça com um objecto no ORB, mesmo que este esteja implementado recorrendo a código procedimental, CICS ou COBOL. Isto torna a arquitectura CORBA uma solução evolutiva. As novas aplicações podem ser escritas recorrendo a objectos, enquanto que as existentes podem ser encapsuladas recorrendo à IDL.

3.4.1. ORB PARTE CLIENTE

Como se verifica na Figura 10, do lado cliente do ORB fazem parte os seguintes componentes:

- *Stubs IDL*. Disponibilizam interfaces estáticas para serviços. Estes *stubs* pré-compilados definem a forma como os clientes vão invocar os serviços pretendidos nos servidores. Na perspectiva do cliente o *stub* funciona como uma chamada local, isto é, uma *proxy* local para um objecto servidor remoto. Os serviços são definidos utilizando a IDL e ambos os *stubs*, cliente e servidor, são gerados pelo compilador IDL, devendo cada cliente ter um *stub* IDL para cada interface que utiliza no servidor. Note-se que o *stub* inclui já o código para a realização do *marshaling*.
- Interface de Invocação Dinâmica. A DII (*Dynamic Invocation Interface*) permite determinar os métodos a invocar na execução. A arquitectura CORBA define APIs normalizadas para localização dos metadados que definem a interface do servidor, gerando os parâmetros, emitindo a chamada remota e trazendo de volta os resultados.
- Repositório de Interfaces. O IR (*Interface Repository*) é uma base de dados distribuída que contém *machine-readable versions* das interfaces IDL, nomeadamente a descrição das interfaces, métodos que suportam e parâmetros que necessitam. Funciona como um repositório de metadados dinâmico para ORBs, isto é, as APIs permitem aos componentes aceder, armazenar e actualizar a informação dos metadados de forma dinâmica. Esta forma de utilização dos metadados permite que cada componente do ORB tenha interfaces auto-descritivas, inclusivamente o próprio ORB é um bus auto-descritivo.

- Interface ORB. A interface ORB consiste em APIs para serviços locais do interesse das aplicações. Por exemplo, a arquitectura CORBA disponibiliza APIs para conversão de referências de objectos em *strings* e vice-versa. Estas chamadas podem ser bastantes úteis se for necessário guardar e comunicar referências de objectos.

3.4.2. ORB PARTE SERVIDOR

Do lado do servidor fazem parte os seguintes componentes (Figura 10):

- *Stubs/Skeletons IDL*. Aos *stubs* do lado servidor o OMG atribui-lhes o nome de *skeletons* cuja função é disponibilizar interfaces estáticas para cada serviço exportado pelo servidor. Os *skeletons*, tal como os *stubs* do lado cliente são criados através do compilador IDL.
- *Dynamic Skeleton Interface*. A DSI (*Dynamic Skeleton Interface*) visa disponibilizar um mecanismo de ligação em tempo de execução a servidores que necessitem manipular chamadas para métodos em componentes que não têm *skeletons* IDL. A DSI verifica os parâmetros da chamada para perceber a quem esta se destina, isto é, qual é o objecto alvo e respectivo método. Contrastando com este facto, os *skeletons* compilados IDL são especificamente definidos para uma determinada classe de objectos e esperam uma implementação para cada método IDL definido. Os *skeletons* dinâmicos são bastante úteis para a implementação de *bridges* genéricas entre ORBs. A DSI é a parte equivalente no servidor ao DII e pode receber ambos os tipos de invocação dos clientes, nomeadamente as invocações estáticas e dinâmicas.
- Adaptador de Objectos. O OA (*Object Adapter*) disponibiliza o ambiente para a instanciação de objectos do servidor, passagem de pedidos para os objectos em causa e atribuir-lhes IDs (também designados por Referências de Objectos - *Object References*). O OA tem também como função registar as classes que suporta e as suas instâncias, isto é os objectos, no repositório de implementações. A arquitectura CORBA na sua versão 2.0 especifica que cada ORB deve suportar um adaptador normalizado chamado BOA (*Basic Object Adapter*). Os servidores não estão limitados a suportar um único adaptador de objectos, podendo suportar vários. A arquitectura CORBA na sua versão 3.0 introduz uma versão portátil do BOA chamada POA (*Portable Object Adapter*).
- Repositório de Implementações. O Repositório de Implementações disponibiliza um repositório de informação acerca das classes que o servidor suporta, dos objectos que foram instanciados e respectivos IDs (*Identifier*). Também é utilizado para guardar informação adicional associada à implementação dos ORBs nomeadamente, *trace information*, *audit trails*, segurança e outros dados de caracter administrativo.
- Interface ORB. A interface ORB consiste num número de APIs para serviços locais que são idênticos aos disponibilizados pelo lado cliente.

Não é possível ao lado servidor a distinção entre uma invocação estática e dinâmica, uma vez que ambas têm a mesma semântica no que respeita às mensagens. Em ambos os casos o ORB localiza o adaptador de objectos no servidor, transmite os parâmetros e transfere o controle para a implementação do objecto através do *skeleton*.

Os aspectos anteriores constituem os componentes principais do ORB que vão ser detalhados nos pontos que se seguem.

3.5. LINGUAGEM DE DEFINIÇÃO DE INTERFACES

Um dos problemas associados à programação distribuída prende-se com a necessidade de operar com diferentes plataformas, sistemas operativos e mesmo diferentes linguagens de programação. Por exemplo o conceito de inteiro difere de plataforma para plataforma e de linguagem para linguagem. Na arquitectura CORBA este problema é ultrapassado recorrendo à IDL (*Interface Definition Language*) (Norma ISO 14750) (Figura 12). A IDL é a linguagem utilizada para descrever as interfaces que os clientes utilizam (chamam) e os servidores (implementações de objectos) fornecem. Uma definição de uma interface escrita em IDL consiste num conjunto de operações e parâmetros relativos a essas mesmas operações. Suporta o mecanismo de herança entre interfaces para permitir a reutilização. Uma interface IDL fornece a informação necessária para desenvolver os clientes que utilizam as operações da interface em causa. Os clientes (interfaces cliente) não são escritos em IDL mas numa outra linguagem de programação, como por exemplo Java, C ou C++, se o mapeamento IDL para essa linguagem estiver definido.

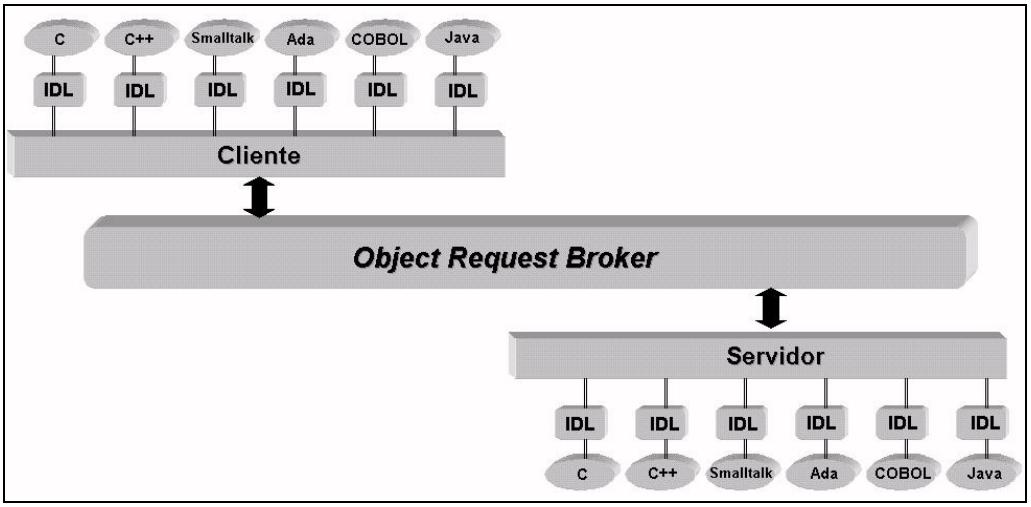

Figura 12 - Linguagem de Definição de Interfaces.

Como qualquer linguagem, a IDL tem a sua própria sintaxe sendo esta muito parecida com a sintaxe do C++ mas mais restrita pois a IDL é uma linguagem de especificação e não de implementação [Vinoski - 1997] (Figura 13).

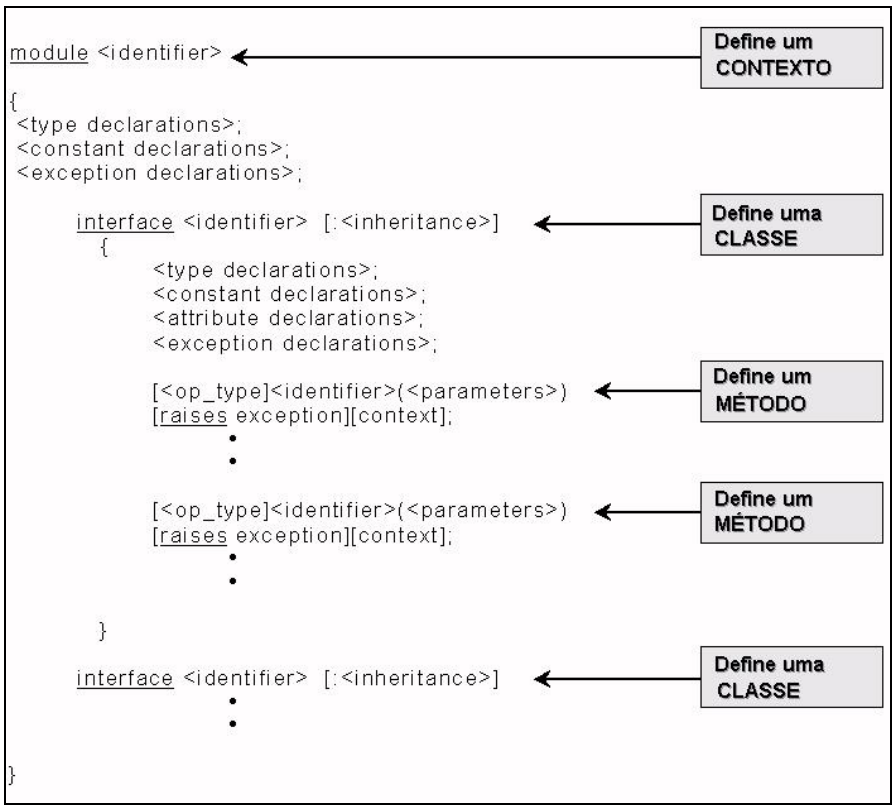

Figura 13 - Estrutura de um ficheiro IDL.

Um compilador de IDL será responsável por transformar o código IDL em interfaces e código adequado ao sistema que se pretende desenvolver. O resultado típico da compilação será um conjunto de ficheiros declarativos e programas de adaptação (programa *stub* no cliente e programa *skeleton* no servidor) para cada interface (Figura 14). O programa cliente liga-se directamente ao *stub*. Na perspectiva do cliente o *stub* age como uma chamada a uma função local e de forma transparente o *stub* fornece uma interface para o ORB que realiza a codificação e descodificação dos parâmetros associados às operações num formato independente utilizado na comunicação. O programa *skeleton* é a implementação correspondente do lado do servidor. Quando o ORB recebe um pedido, o *skeleton* fornece uma referência para uma implementação da função no servidor. Quando o servidor completa o processamento do pedido, o *skeleton* e o *stub* devolvem os resultados ao programa cliente. Se existirem erros no desenrolar deste processo, são geradas excepções por parte do servidor ou por parte do ORB, que nas alturas próprias são devolvidas ao cliente.

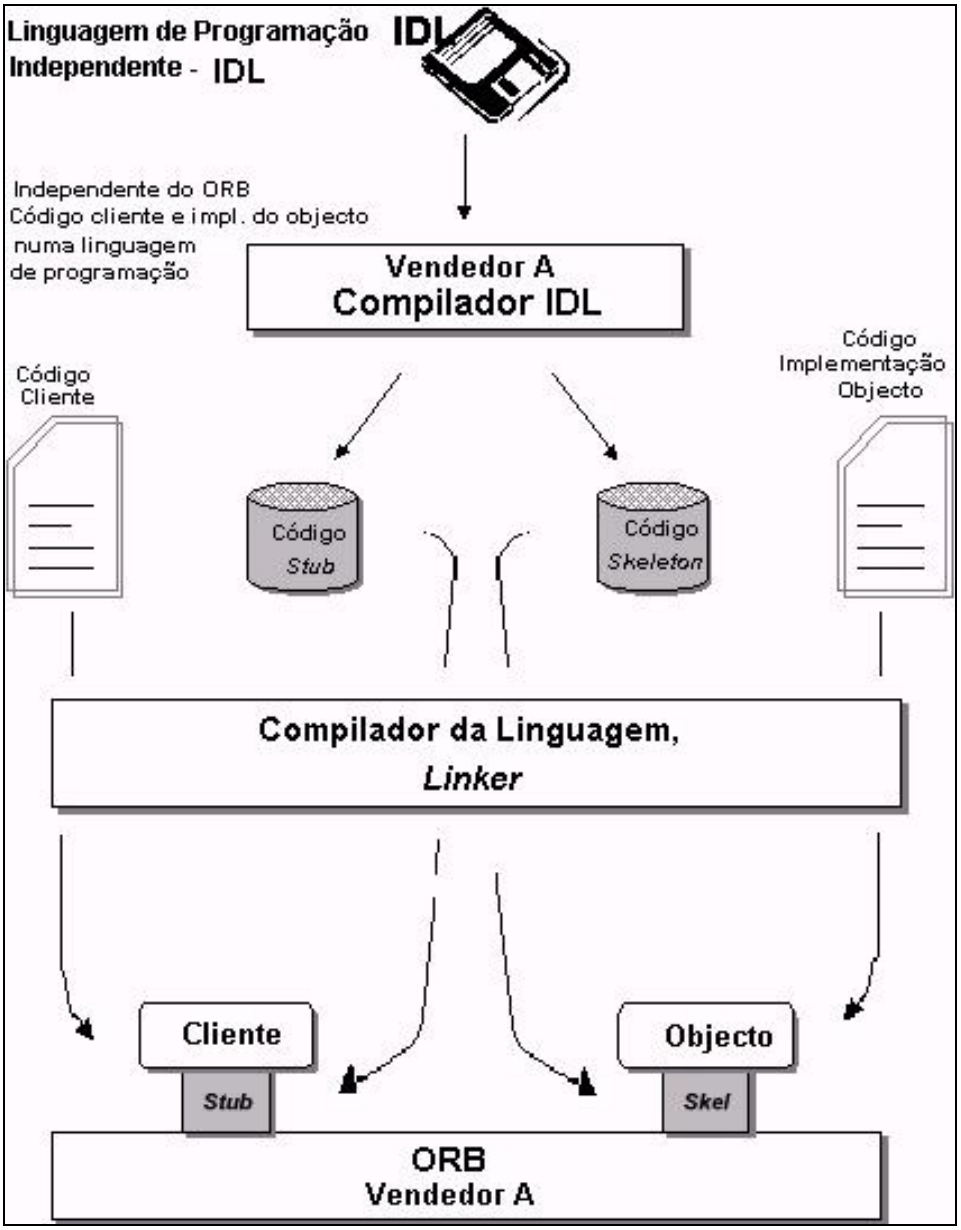

Figura 14 - Ficheiro IDL, cliente e implementação do objecto (o sombreado indica componentes gerados) [Siegel - 1996].

Na Figura 15 representa-se o modo de como se pode fazer a integração de um componente exterior num ambiente de software recorrendo à IDL.

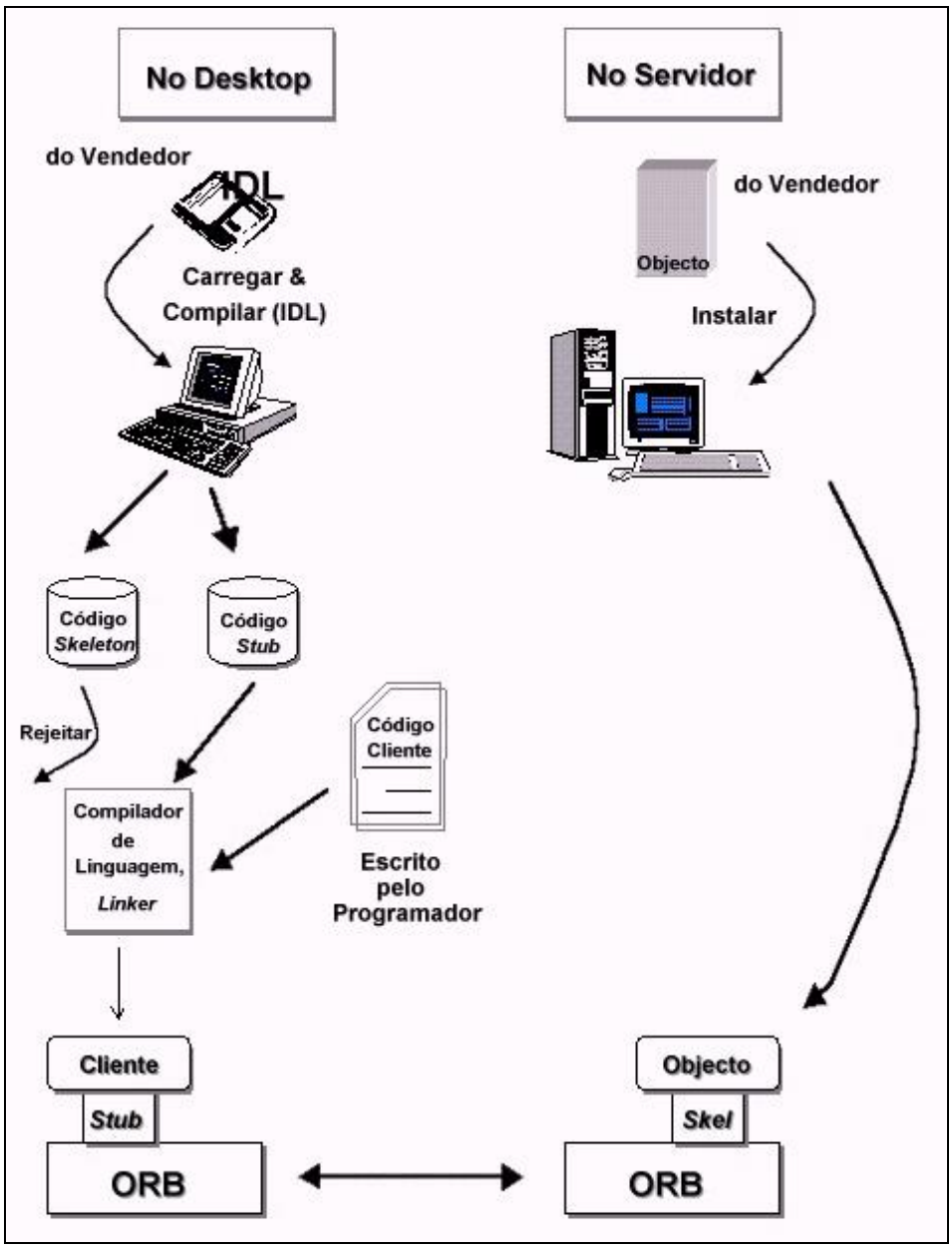

Figura 15 - Integração de componentes num ambiente de software [Siegel - 1996].

3.5.1. TIPOS DE DADOS

A IDL é uma linguagem muito próxima do C++. Suporta tipos de dados básicos e estruturados [OMG - 1995a].

Na Tabela I são apresentados os tipos de dados suportados pela IDL.

Tabela I. Tipos de Dados IDL.

	Tipo	Descrição
Básicos:	(unsigned) short	inteiro 16 bits - signed [unsigned].
	(unsigned) long	inteiro 32 bits - signed [unsigned].
	float	número vírgula flutuante - 16 bits.
	double	número vírgula flutuante - 32 bits.
	char	caracter ISO Latin-1 (8859.1).
	boolean	tipo lógico/boleano. Toma os valores verdade (true) ou falso (false).
	string	conjunto de caracteres.
	octet	tipo genérico de 8 bits.
	enum	identificadores com numeração implícita.
	any	pode representar um valor de qualquer um dos tipos IDL, básico ou estruturado.

Estruturados:	structure	registo:
		`struct struct_type_name {`
		`  type1 member_name1;`
		`  type2 member_name2;`
		`};`

	union	é o cruzamento das declarações union e switch do C:
		`union union_type_name switch(discriminator_type) {`
		`  case value1 : type1 member_name1;`
		`  case value2 : type2 member_name2;`
		`  default : type3 member name3;`
		`}`

	array	lista indexada de comprimento fixo.
		`typedef array type name1 member type1(10);`
		`typedef array type name2 member_type2(10)(60);`

	sequence	lista indexada de comprimento variável que pode ter um limite superior.
		`typedef bounded seq type name sequence <member type1, 30>;`
		`typedef unbounded_seq_type_name sequence <member_type2>;`

	exception	estrutura que pode ser retornada de uma operação em caso de terminação alternativa e é geralmente utilizada para indicar uma condição de erro.
		`exception exception_name {`
		`  type1 member_name1;`
		`  type2 member_name2;`
		`}`

3.6. Interface de Invocação Dinâmica

Os primeiros produtos ORB na sua maioria tinham como base a DII (*Dynamic Invocation Interface*) que é uma alternativa aos interfaces estáticos

compilados IDL [Mowbray - 1995]. A DII permite que um programa cliente construa pedidos e faça invocações a objectos de forma dinâmica. O cliente especifica o objecto a ser invocado, o método a ser executado e o conjunto de parâmetros a serem enviados na chamada ou sequência de chamadas. Esta informação é geralmente obtida pelo cliente a partir do IR ou de outra fonte. As invocações dinâmicas fornecem uma flexibilidade total no que respeita à introdução em sistemas distribuídos de novos objectos em tempo de execução, isto é, a DII possibilita aos clientes a invocação de operações em objectos sem que para tal seja necessário a geração do código do *stub* por parte do compilador IDL.

De um modo geral, a programação no que respeita a interfaces estáticos é mais simples e resulta em código mais robusto para o programador do que a DII. No entanto esta fornece um nível de flexibilidade que é por vezes necessário para algumas aplicações como é o caso por exemplo dos sistemas operativos [Mowbray - 1995].

A DII apresenta as seguintes vantagens [Pope - 1997]:

- O desenvolvimento em tempo de execução é muitas vezes ideal para uma rápida prototipagem.
- Menor *overhead* no desenvolvimento que no ciclo tradicional *code/load/run/debug*.
- Numa aproximação dinâmica, o código produzido pode ser até aproximadamente 80% menos que o equivalente código compilado.
- No respeitante a novos objectos, estes podem eventualmente não ter que ser recompilados.

Para a invocação dinâmica de um método num objecto, o cliente deverá realizar os quatro passos descritos na Figura 16.

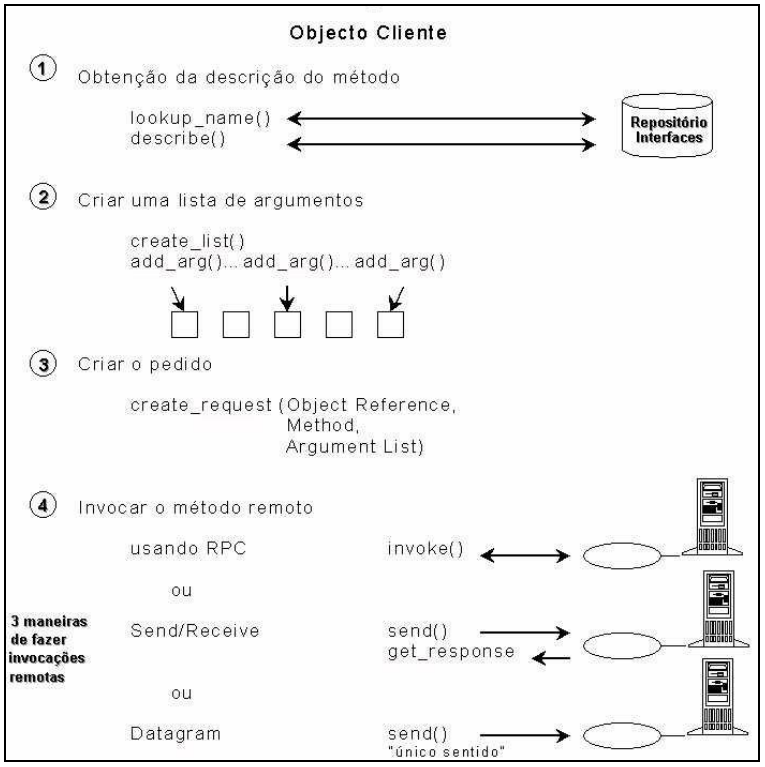

Figura 16 - Interface de Invocação Dinâmica.

3.7. INTERFACE DE SERVIÇOS DINÂMICA

A DII fornece um mecanismo que possibilita aos clientes a invocação de operações em objectos sem que para tal seja necessário o código para a geração do *stub* por parte do compilador IDL.

A DSI (*Dynamic Skeleton Interface*) fornece um mecanismo semelhante sobre o ORB (Figura 17). Através dela o ORB pode invocar uma implementação de um objecto em tempo de execução para o qual não existe *skeleton*.

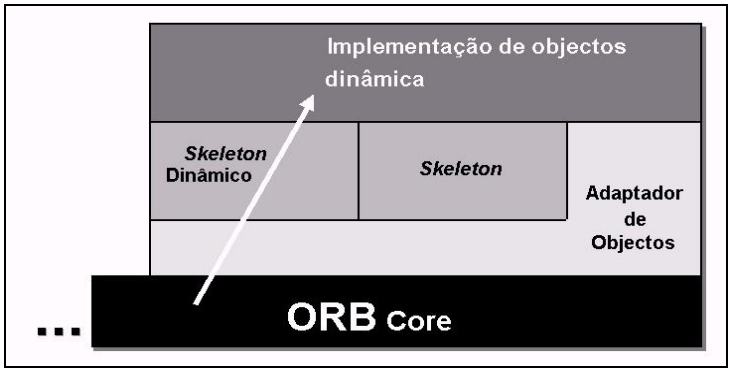

Figura 17 - Interface de Serviços Dinâmica.

A implementação de um objecto não distingue uma chamada via *skeleton* ou via DSI. A ideia por detrás da DSI é a invocação de todas as implementações de objectos via a mesma operação geral.

Como características importantes da DSI podem ser enumeradas as seguintes [Pope - 1997] [Cortés - 1998]:

- Um servidor que utilize este método de acesso, pode ver reduzido de forma substancial o número de *skeletons* estáticos a suportar.
- Capacidade de comunicação com outros ORBs, onde a DSI do servidor actuaria como cliente do ORB onde está localizado o objecto destino.
- Pode receber invocações dinâmicas e estáticas.
- Permite a ligação em tempo de execução com um objecto para o qual não existe *skeleton*.

3.8. REPOSITÓRIO DE INTERFACES

O IR (*Interface Repository*) é uma base de dados *on-line* de definições de objectos. É portanto o componente do ORB que fornece armazenamento persistente das definições das interfaces (gere e fornece acesso a uma colecção de definições de objectos especificados em IDL) (Figura 10).

Para que um ORB processe correctamente os pedidos deverá ter acesso às definições dos objectos que por ele são utilizados. As definições de objectos podem ser disponibilizadas ao ORB de duas maneiras distintas:

- Através da incorporação das definições em rotinas *stub* (por exemplo o código que mapeia subrotinas numa determinada linguagem em protocolos de comunicação).
- Como objectos acedidos através do IR.

Em particular, o ORB pode utilizar as definições de objectos mantidas no IR para interpretar e manipular os valores fornecidos no pedido com o fim de [Orfali - 1998]:

- Comunicar com outras implementações ORB distintas.
- Verificar os parâmetros do pedido.
- Disponibilizar objectos auto-descritivos.
- Disponibilizar metadados para clientes e ferramentas.

Os clientes por sua vez solicitam os serviços do IR para:

- Navegação na lista de interfaces.
- Facilitar a instalação e distribuição dos objectos.

Uma vez que a interface para a definição do objecto mantida no repositório de interfaces é pública, a informação mantida no repositório pode também ser utilizada pelos clientes e serviços.

O IR utiliza módulos como forma de agrupar as interfaces e permite navegar através de tal agrupamento utilizando nomes. Os módulos podem conter constantes, *typedefs*, excepções, definições de interfaces e outros módulos (Figura 18).

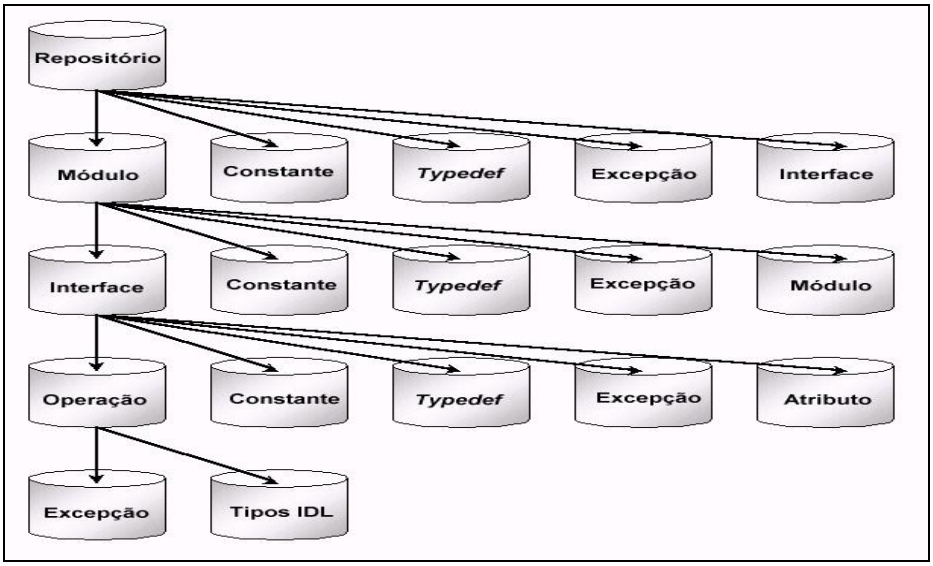

Figura 18 - Repositório de Interfaces.

- Um Módulo define um agrupamento lógico de interfaces.
- Uma Interface define a interface do objecto.
- Uma Operação define um método numa interface de um objecto.
- Uma Constante define uma constante.
- Uma Excepção define as excepções que podem ser devolvidas por uma operação.
- Uma *Typedef* define os tipos de nomes que fazem parte de uma definição IDL.

Um ORB pode ter acesso a múltiplos IRs. Como se verifica na Figura 19, a mesma interface `Doc` está instalada em dois repositórios diferentes, uma na `SoftCo, Inc.`, que vende `Doc objects`, e uma na `Customer, Inc.`, que compra `Doc Objects da SoftCo.`. A `SoftCo` dá à interface em questão o identificador do seu repositório (*RepositoryID*). O cliente também tem a sua interface `Doc` com o mesmo *RepositoryID* do IR da `SoftCo.`, de tal forma que o ORB do cliente sabe que estas interfaces são a mesma podendo estabelecer a comunicação entre ORBs.

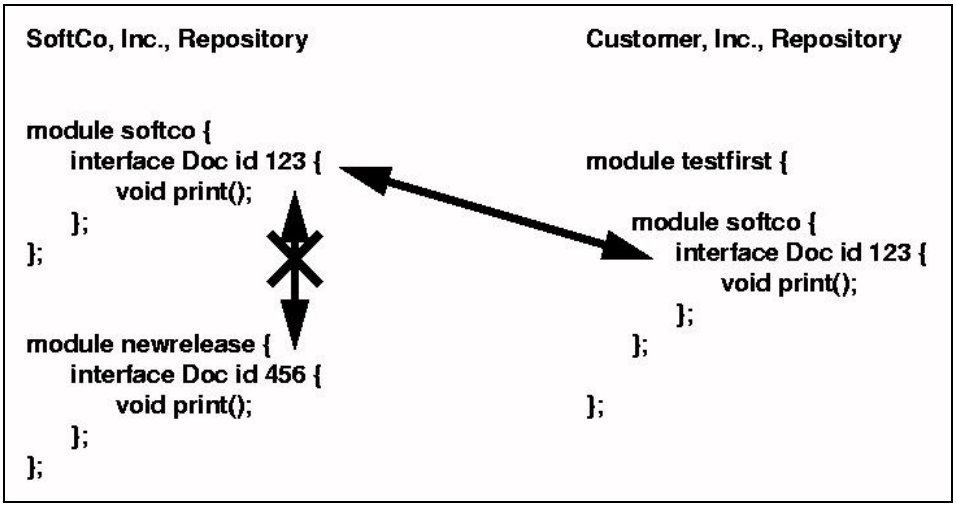

Figura 19 - Utilização de *RepositoryIDs* para estabelecer a correspondência entre IRs.

3.8.1. IDENTIFICADORES DE REPOSITÓRIO

Os *RepositoryIDs* são identificadores globais que identificam univocamente componentes e respectivas interfaces através de diferentes ORBs e repositórios. Os *RepositoryIDs* são *strings* geradas automaticamente pelo sistema e são utilizadas para manter a consistência das convenções de nomes utilizadas entre repositórios, não permitindo colisões (de nomes). Estes IDs podem também ser utilizados para fazer a replicação de cópias dos metadados através de múltiplos repositórios, possibilitando desta forma a manutenção da informação num estado de integridade entre repositórios.

O formato do ID é um nome seguido por dois pontos (:) e os caracteres de acordo com o formato em causa; IDL, DCE UUID e LOCAL.

FORMATO IDL

O *RepositoryID* consiste em três componentes separados por dois pontos (":")

- O 1º componente é o nome do formato, "IDL".
- O 2º componente é uma lista de identificadores separados pelo caracter "/".
- O 3º componente é construído a partir dos números *major* e *minor version*, no formato decimal separados por ponto (".").

Exemplo: Um exemplo válido para o *RepositoryID* para a interface `Cat` no módulo `MyAnimals` é - IDL:DogCatInc/MyAnimals/Cat/:1.0. Neste caso a designação `DogCatInc` é um prefixo único que identifica uma organização. Podem também ser utilizados como prefixos IDs Internet ou outro qualquer nome desde que seja único.

FORMATO DCE UUID

O formato DCE - UUID (*Distributed Communication Environment - Universal Unique Identifier*) do *RepositoryID* inicia-se com os caracteres "DCE:" seguidos pelo UUID, um ponto e o número correspondente à *minor version* em décimal.

Exemplo: DCE:700dc518-0110-11ce-ac8f-0800090b5d3e:1

FORMATO LOCAL

O formato LOCAL do *RepositoryID* inicia-se com os caracteres "LOCAL:" seguidos por uma *string* arbitrária. Os IDs no formato LOCAL não podem ser utilizados fora de um determinado repositório para o qual foram definidos, e não necessitam de estar de acordo com nenhuma convenção em particular.

3.9. ADAPTADORES DE OBJECTOS

Um OA (*Object Adapter*) compreende a interface entre o ORB e a implementação do objecto. É a interface principal que as implementações de objectos utilizam para aceder a funções do ORB. Os OAs suportam as seguintes funções [Orfali - 1996]:

- Registo das implementações de objectos no Repositório de Implementações. O Repositório de Implementações funciona como o local onde são guardadas de forma persistente as implementações dos objectos. Este repositório é gerido pelo OA.
- Instanciação de novos objectos em tempo de execução. O OA é responsável pela criação de instâncias de objectos cujas implementações se encontrem no Repositório de Implementações.
- Geração e gestão de referências de objectos. O OA atribui referências (IDs únicos) aos novos objectos que cria.
- Divulgação de objectos servidores. O OA pode divulgar os serviços que disponibiliza no ORB.

- Manipulação de chamadas de clientes. O OA é responsável por dirigir as chamadas dos clientes para o *skeleton* apropriado.
- Enviar as chamadas para o método apropriado. O OA está implicitamente envolvido na invocação do método em causa. Por exemplo, o OA pode estar envolvido na activação da implementação do objecto.

Existem muitos potenciais tipos de adaptadores de objectos, nomeadamente, adaptadores de âmbito geral, para a integração de bases de dados (OODA - *Object Oriented Database Adapter*), para a integração de bibliotecas (LOA - *Library Object Adapter*) e outros. A arquitectura CORBA define apenas o BOA, mas reconhece a necessidade para a existência destes outros tipos de adaptadores [Mowbray - 1995].

3.9.1. ADAPTADOR DE OBJECTOS BÁSICOS

O BOA (*Basic Object Adapter*) é um objecto criado directamente pelo ORB que pode ser invocado da mesma forma que outro qualquer objecto, disponibilizando operações às quais os servidores podem aceder, podendo em determinadas circunstâncias chamar também implementações de objectos. Funciona também como interface com o núcleo do ORB e *skeletons* recorrendo a interfaces particulares, o que significa que o BOA é especifico a cada ORB [Orfali - 1998]. É portanto uma interface planeada para estar amplamente disponível e suportar uma ampla variedade de implementações comuns de objectos.

Quando o pedido de um cliente especifica um servidor inactivo, o BOA activa automaticamente o processo servidor. A primeira responsabilidade do servidor é registar a sua implementação no BOA. O BOA guarda este registo para o utilizar num futuro pedido do objecto. Após um objecto ser activado este pode receber pedidos dos clientes utilizando o método *callback* do *skeleton*.

O BOA disponibiliza os seguintes mecanismos [Orfali - 1996]:

- Geração e interpretação de referências de objectos.
- Activação e desactivação de implementações.
- Autenticação do cliente que faz a chamada.
- Invocação de métodos através de *skeletons*.
- Registo de implementações no IR.

A Figura 20 mostra a estrutura básica do BOA e algumas das iterações entre o BOA e uma implementação de um objecto. O BOA inicia um programa para fornecer a implementação do objecto (1.). A implementação do objecto notifica o BOA que a sua inicialização foi terminada (a implementação do objecto está pronta) e que está preparada para receber pedidos (2.). Quando a implementação do objecto recebe o primeiro pedido o objecto é activado (3.). Em pedidos subsequentes, o BOA chama o método apropriado utilizando o *skeleton* (4.). Em qualquer altura, a implementação pode aceder a serviços BOA tais como criação de objectos, desactivacao, etc. (5.).

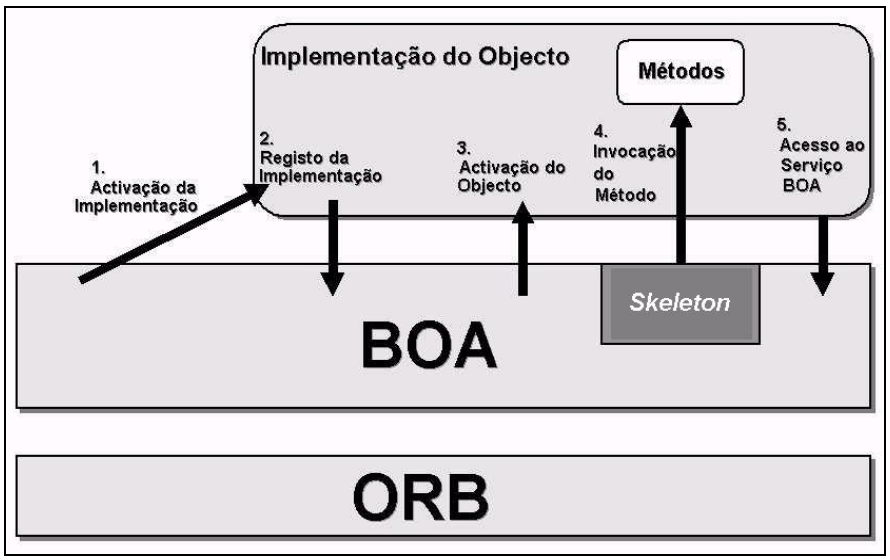

Figura 20 - Estrutura e operação do BOA.

ACTIVAÇÃO E DESACTIVAÇÃO DE IMPLEMENTAÇÕES

Existem dois tipos de activações que o BOA necessita realizar como parte da invocação de operações. A primeira é a activação da implementação, a qual ocorre quando nenhuma implementação do objecto está disponível para manipular o pedido. A segunda é a activação do objecto, a qual ocorre quando nenhuma instância do objecto está disponível para manipular o pedido.

A activação de uma dada implementação requer a coordenação entre o BOA e os programas que contêm a implementação. O termo servidor é utilizado como sendo uma entidade executável (processo ou *thread*) que o BOA pode iniciar num sistema específico. O BOA inicia a actividade pela implementação, iniciando o servidor apropriado, provavelmente por um mecanismo de operação dependente do sistema. A implementação inicializa-se a si própria, notificando o BOA que está preparada para manipular pedidos através da chamada às operações *impl_is_ready* ou *obj_is_ready*. Entre o tempo de arranque do programa, até à indicação que está pronto, o BOA não permite que outros pedidos sejam enviados para o servidor. Depois disso, através dos *skeletons*, o BOA já poderá fazer as chamadas aos métodos das implementações.

A política de activação descreve as regras que uma dada implementação segue quando há múltiplos objectos ou implementações activas. Existem quatro políticas que todas as implementações do BOA suportam para a activação de implementações:

- Servidor partilhado, onde múltiplos objectos activos, de uma dada implementação, partilham o mesmo servidor.
- Servidor não partilhado, onde somente um dado objecto, de uma implementação, a cada momento pode estar activo num servidor.
- Servidor-por-método, onde cada invocação de um método é implementada por um servidor. Quando o método termina, o servidor é também terminado .
- Servidor persistente, onde o servidor é activado por algo exterior ao BOA. Assume-se que um servidor persistente é partilhado por múltiplos objectos activos.

Os tipos de activação de implementações descritos são ilustrados na Figura 21.

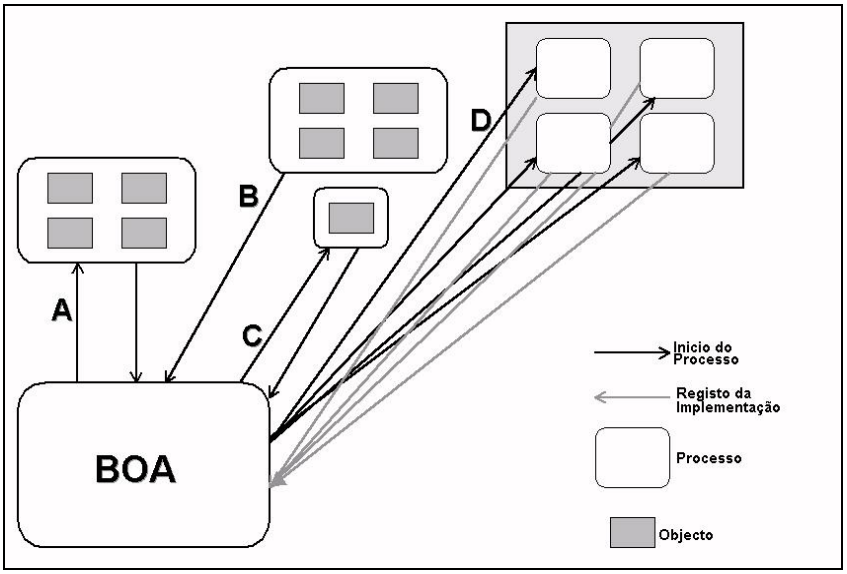

Figura 21 - Políticas de activação de implementações.

A: Servidor partilhado, o BOA inicia um processo e então esse processo regista-se no BOA.

B: Servidor persistente, é muito semelhante ao anterior, regista-se no BOA, sem o BOA ter que iniciar um processo.

C: Servidor não partilhado, o processo iniciado pelo BOA unicamente suporta um único objecto.

D: Neste caso, todas as invocações a métodos implicam a criação de um processo.

SERVIDOR PARTILHADO

Num servidor partilhado, múltiplos objectos podem ser implementados pelo mesmo programa. Este é o tipo mais comum de servidor. O servidor é activado pela primeira vez quando é feito um pedido a um qualquer objecto implementado por esse servidor. Quando a inicialização do servidor termina, este notifica o BOA desse facto chamando a rotina *impl_is_ready*. Subsequentemente, o BOA irá entregar pedidos ou solicitações de activação de objectos a qualquer objecto implementado pelo servidor em causa. O servidor permanecerá activo e receberá pedidos até chamar *deactivate_impl*. O BOA não activará qualquer outro servidor para esta implementação se já houver algum activo.

Um objecto permanecerá activo enquanto o seu servidor estiver activo, a menos que o servidor chame *deactivate_obj* para o objecto em causa.

SERVIDOR NÃO PARTILHADO

Num servidor não partilhado, cada objecto é implementado num servidor diferente. Este tipo de servidor é conveniente se for propósito de um objecto encapsular uma aplicação ou se o servidor necessitar acesso exclusivo a um recurso como por exemplo uma impressora. Um novo servidor é activado a partir do momento que é feito um primeiro pedido ao objecto. Quando a inicialização do servidor termina, este notifica o BOA que está disponível chamando a rotina *obj_is_ready*. Subsequentemente, o BOA entregará pedidos para esse objecto. O servidor permanecerá activo e receberá pedidos até chamar a rotina *deactive_obj*.

Um novo servidor é iniciado sempre que um pedido é feito a um objecto que ainda não se encontra activo, mesmo se um servidor para outro objecto com a mesma implementação se encontrar activo.

SERVIDOR-POR-MÉTODO

Neste caso, é iniciado um novo servidor sempre que é feito um pedido. O servidor está activo somente durante a execução do método em questão. Vários servidores para o mesmo objecto ou para o mesmo método de um mesmo objecto podem estar activos simultaneamente. Pelo facto de ser iniciado um novo servidor cada vez que é feito um pedido, não é necessário à implementação notificar o BOA quando um objecto é activado ou desactivado.

SERVIDOR PERSISTENTE

Servidores persistentes são aqueles que são activados de forma externa ao BOA. Estas implementações notificam o BOA que estão disponíveis utilizando a operação *impl_is_ready*. Uma vez o BOA ter conhecimento do servidor persistente, este trata-o como sendo um servidor partilhado. Se nenhuma implementação esta apta quando um pedido chega, é devolvido um erro para esse pedido.

3.10. INTEROPERABILIDADE

Com a interoperabilidade entre ORBs pretende-se que de uma forma flexível e polivalente os objectos distribuídos possam ser geridos por vários ORBs heterogéneos.

Os elementos da interoperabilidade são os seguintes:

- Arquitectura de interoperabilidade ORB.
- Suporte para *bridges* inter-ORB.
- Protocolos inter-ORB gerais e para a internet (GIOPs e IIOPs).

Como complemento a arquitectura CORBA suporta ESIOPs (*Environment-Specific Inter-ORB Protocols*) que são protocolos optimizados para determinados ambientes, como é o caso do DCE (*Distributed Communication Environment*).

3.10.1. ARQUITECTURA DE INTEROPERABILIDADE

A arquitectura de interoperabilidade fornece um modelo conceptual para definição dos elementos de interoperabilidade e para identificação de pontos de concordância. Caracteriza também novos mecanismos de interoperabilidade e especifica as convenções necessárias para se conseguir a interoperabilidade entre diferentes ORBs (produzidos por exemplo por empresas diferentes). Especificamente esta arquitectura introduz conceitos de *bridging* para domínios ORB. Através do uso de técnicas de *bridging*, os ORBs podem comunicar sem o conhecimento de quaisquer detalhes de implementação uns dos outros, como por exemplo, qual o protocolo utilizado para a implementação das especificações CORBA.

O IIOP (*Internet Inter-ORB Protocol*) pode ser utilizado para o *bridging* de dois ou mais ORBs através da implementação de *half-bridges*, que por sua vez comunicariam através do IIOP. Este tipo de aproximação funciona quer para ORBs auto-suficientes (*standalone*) quer para ORBs em rede que usem um ESIOP [OMG - 1995a] (Figura 22). Este protocolo pode também ser utilizado para a implementação de comunicações internas no ORB.

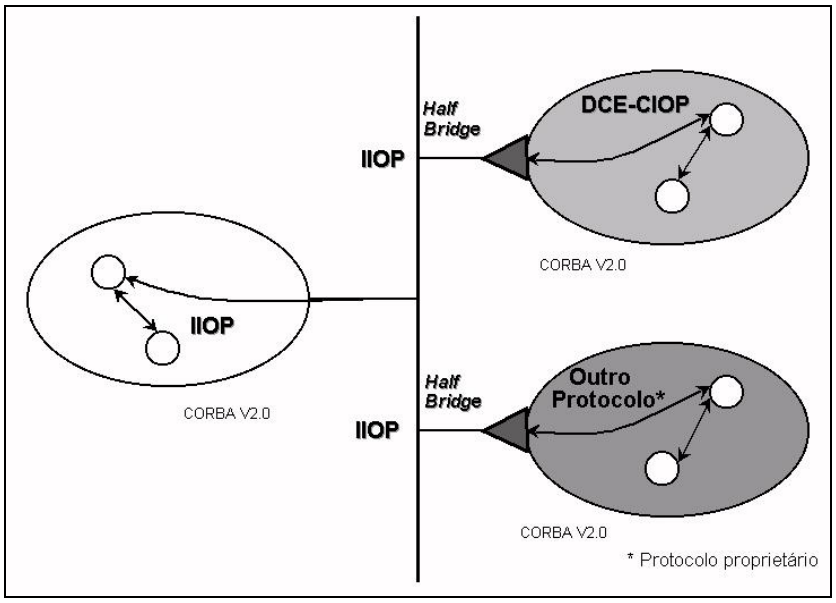

Figura 22 - Exemplos de interoperabilidade CORBA.

3.10.2. SUPORTE PARA BRIDGES INTER-ORB

A arquitectura de interoperabilidade identifica claramente o papel dos diferentes tipos de domínios (Transacção, Segurança e outros) relativamente à sua informação específica.

Quando dois ORBs estão no mesmo domínio podem comunicar directamente, no entanto quando uma invocação deixar o seu domínio, esta deverá atravessar uma *bridge* (Figura 22 e Figura 23). O papel da *bridge* é fazer com que o conteúdo e semântica sejam mapeados de forma apropriada de um ORB para o outro o que implica que os utilizadores de um determinado ORB só vêm os conteúdos e semântica próprios desse mesmo ORB.

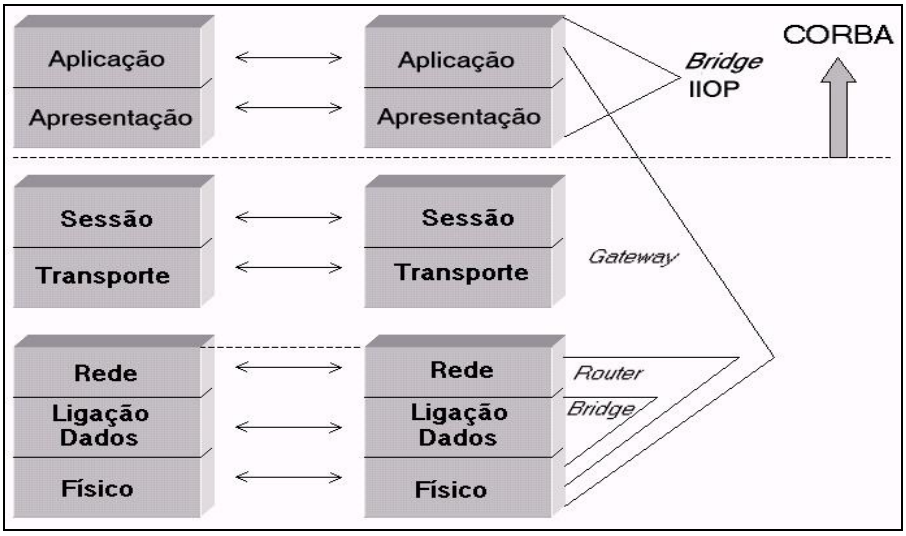

Figura 23 - IIOP e *bridges* de rede.

O elemento de suporte para *bridges* inter-ORB especifica APIs e convenções de forma a ser possível a fácil construção de *bridges* entre domínios ORB. Este suporte pode também ser utilizado para fornecer interoperabilidade com sistemas não-CORBA, como é o caso por exemplo do DCOM da Microsoft.

3.10.3. GENERAL INTER-ORB PROTOCOL

O GIOP (*General Inter-ORB Protocol*) especifica uma sintaxe de transferência normalizada (representação de dados) e um conjunto de formatos de mensagens para comunicação entre ORBs. O GIOP foi construído especificamente para interacções entre ORBs e foi desenhado para trabalhar directamente sobre qualquer protocolo de transporte orientado à conexão que reconheça um conjunto mínimo de suposições (Figura 24).

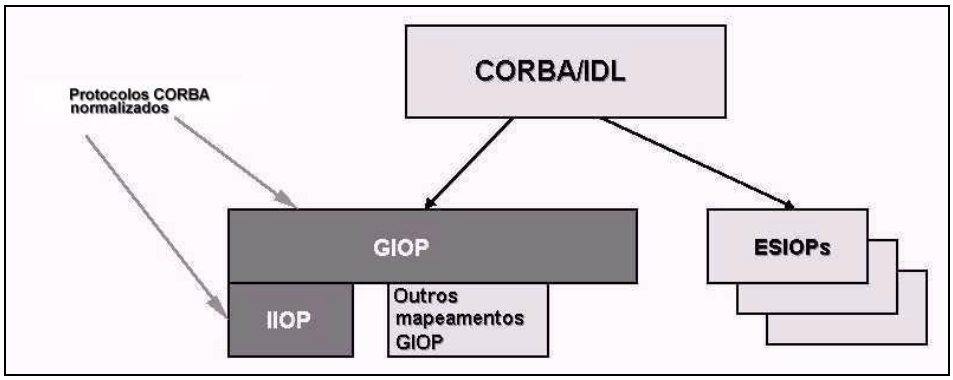

Figura 24 - Relação entre protocolos.

Para o seu desenho foram tidos em consideração aspectos como:

- Possibilidade de utilização o mais alargada possível - tanto o GIOP como o IIOP correm sobre o TCP/IP (*Transport Control Protocol/Internet Protocol*) (Figura 25).
- Simplicidade - ser o mais simples possível.

3.10.4. ENVIRONMENT-SPECIFIC INTER-ORB PROTOCOL

O primeiro ESIOP (*Environment-Specific Inter-ORB Protocol*) a ser especificado foi o para o DCE sendo actualmente ainda a única definição existente. É similar na forma ao GIOP, mas os mapeamentos são mais directos para o DCE.

O DCE-CIOP (*Distributed Communication Environment - Common Interoperability Protocol*) é uma derivação ou subconjunto do DCE. Neste caso o mapeamento para operações inter-ORB é mais efectivo do que directamente a partir do DCE (Figura 25).

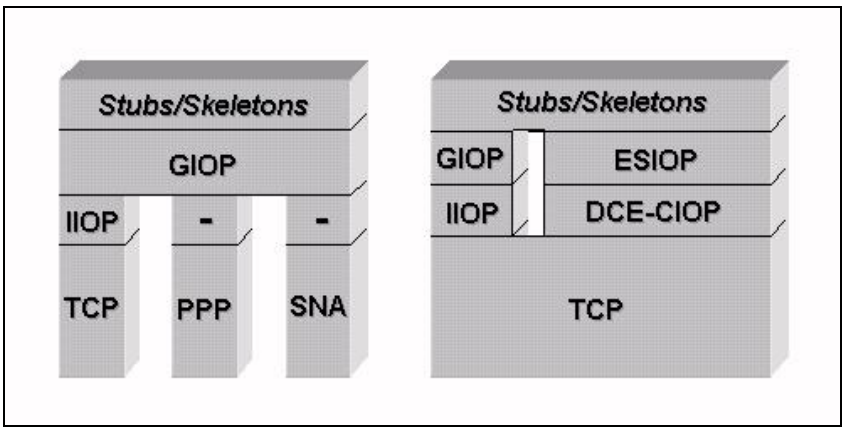

Figura 25 - Interoperabilidade CORBA.

3.11. SERVIÇOS CORBA

3.11.1. INTRODUÇÃO

O serviços CORBA são um conjunto de serviços de sistema com interfaces IDL que têm como finalidade aumentar e complementar a funcionalidade do ORB permitindo criar componentes, dar-lhes nomes e coloca-los disponíveis no sistema. Até à data actual o OMG publicou normas para 15 serviços (Figura 26).

A publicação destes serviços teve inicio com a recomendação COSS (*Common Object Services Specification*) [OMG - 1994] que detalha os princípios gerais que vão definir os COS (*Common Object Services*) hoje designados serviços CORBA (*CORBAservices*). Inicialmente esta recomendação descrevia unicamente as interfaces e operações de quatro serviços CORBA de âmbito geral nomeadamente: nomes, eventos, ciclo de vida e armazenamento persistente de objectos.

A recomendação dos serviços CORBA (COSS2 - RFP2) [OMG - 1995] incluí além das anteriores, as especificações para mais quatro serviços nomeadamente: controlo da concorrência, exteriorização, relações, transacções e controlo da concorrência.

O COSS3 (RFP3) e o COSS4 (RFP4) decorrem em paralelo e definem no primeiro caso os serviços tempo e segurança e no segundo caso os serviços licenças, associação de propriedades e interrogação.

Os últimos serviços a ser definidos foram os serviços pertencentes ao COSS5 (RFP5), nomeadamente colecções e negociação.

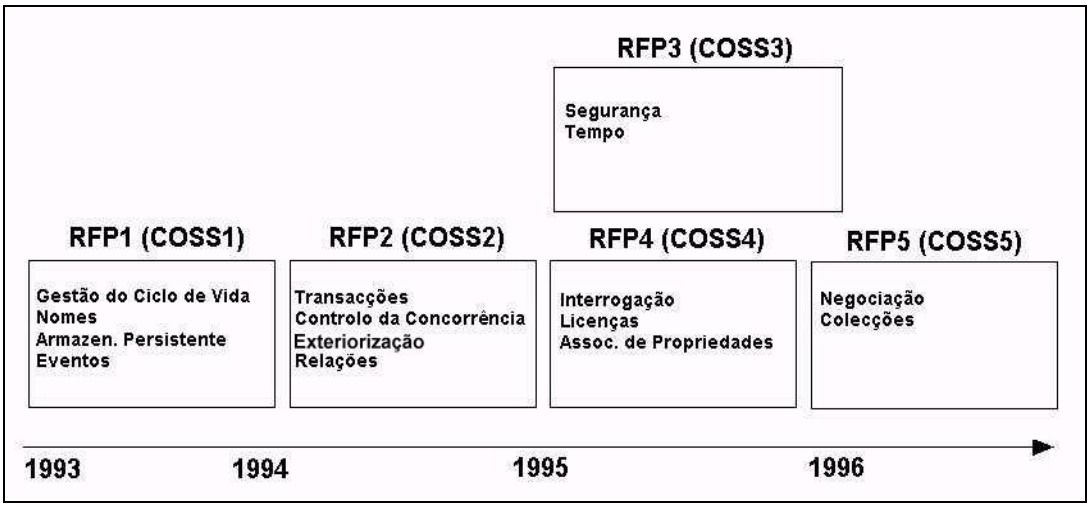

Figura 26 - Evolução dos serviços CORBA.

O OMG recomenda que no desenho/implementação de cada serviço se tenham em consideração os seguintes princípios: propriedades do serviço, consistência das interfaces, resultados, acomodação de serviços futuros, dependências de serviços, relação com a CORBA, relação com a OMA e semelhanças com as normas.

Segundo Baker [Baker - 1997] os serviços CORBA podem ser divididos e integrados nas seguintes grandes categorias:

- SERVIÇOS RELACIONADOS COM SISTEMAS DISTRIBUÍDOS:
  - Serviço de Nomes.
  - Serviço de Eventos.
  - Serviço de Segurança.
  - Serviço de Negociação.
  - Serviço de Colecção.
- SERVIÇOS RELACIONADOS COM BASES DE DADOS:
  - Serviço de Controlo da Concorrência.
  - Serviço de Associação de Propriedades.
  - Serviço de Transacções.
  - Serviço de Relações.
  - Serviço de Interrogações.
  - Serviço de Armazenamento Persistente.
  - Serviço de Exteriorização.

- SERVIÇOS DE ÂMBITO GERAL:
  - Serviço de Gestão do Ciclo de Vida.
  - Serviço de Licenças.
  - Serviço de Tempo.

3.11.2. DESCRIÇÃO DOS SERVIÇOS

SERVIÇO DE NOMES

O NS (*Naming Service*) permite aos componentes (objectos) localizar outros através do nome. Este serviço permite que um objecto seja encontrado numa subdirectoria local ou de outra rede. Respeita as normas: X.500 da ISO, DCE da OSF (*Open Software Foundation*), NIS (*Network Information Service*) da Sun, NDS (*Novell Directory Services*) da Novell e Internet LAPD (*Lightweight Directory Access Protocol*).

SERVIÇO DE EVENTOS

O ES (*Event Service*) fornece as capacidades básicas de comunicação entre objectos. O ES permite aos componentes enviar e receber eventos. Sempre que um componente esteja interessado em receber um determinado tipo de evento, deve registar-se no serviço, eliminando o registo quando deixar de ter interesse em recebê-lo (ao evento).

O canal de eventos é o objecto do serviço responsável por receber e reenviar os eventos. Os componentes que enviam e os que recebem eventos não se conhecem *a priori*. Só o canal de eventos está a par dos componentes que se registaram com o intuito de os receber.

SERVIÇO DE SEGURANÇA

O SS (*Security Service*) disponibiliza um modelo de segurança para objectos distribuídos e compreende: identificação e autenticação dos sujeitos, autorização e controlo de acessos, auditoria de segurança nas comunicações, prova contra repudiação das acções exercidas e administração da informação de segurança (Figura 27).

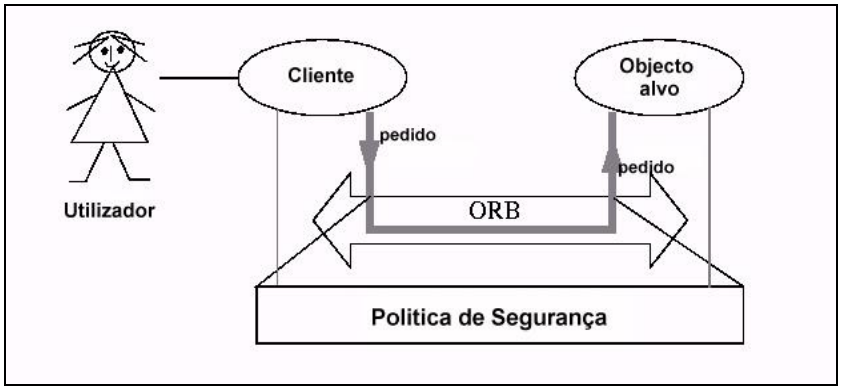

Figura 27 - Modelo de segurança para sistemas de objectos.

SERVIÇO DE COLECÇÕES

O CS (*Collections Service*) disponibiliza uma maneira uniforme para criar e manipular genericamente as colecções mais comuns. As colecções são grupos de objectos, que como grupo, suportam algumas operações e exibem comportamentos específicos que estão mais relacionados com a natureza da colecção do que com o tipo de objectos que contêm. Exemplos de colecções são: conjuntos, filas, pilhas, listas e árvores binárias.

Por exemplo, os conjuntos podem suportar entre outras as seguintes operações: inserção de um novo elemento, união, intersecção, cardinalidade, teste de igualdade e teste de vazio. Uma das semânticas de um conjunto é: se um objecto `O` é membro de um conjunto `S`, então inserindo `O` em `S` o conjunto resultante é o conjunto `S` inicial.

SERVIÇO DE NEGOCIAÇÃO

O TS (*Trader Service*) fornece um serviço de conetagem para objectos. O provedor do serviço regista a disponibilidade do serviço através da invocação de uma operação de exportação no negociador (*trader*), passando como parâmetros informação acerca do serviço oferecido, isto é, disponibiliza uma espécie de Páginas Amarelas para objectos onde estes publicitam os seus serviços. A operação de exportação transporta uma referência de um objecto que pode ser utilizada por um cliente para invocar operações nos serviços anunciados, uma descrição do tipo do serviço oferecido (isto é, os nomes das operações às quais ele responderá de acordo com os seus parâmetros e tipo de resultados) e informação sobre os atributos do serviço oferecido.

O espaço gerido pelos negociadores pode ser dividido de forma a facilitar a administração e a navegação. Esta informação é armazenada de forma persistente pelo negociador. Sempre que um potencial cliente deseja obter uma referência para um serviço que executa uma determinada tarefa, invoca uma operação de importação, passando como parâmetros uma descrição do serviço requerido. Uma vez solicitado este pedido de importação, o negociador averigua as ofertas apropriadas para aceitação. Para uma oferta ser aceitável, esta deve ter um tipo de acordo com o solicitado e ter propriedades compatíveis com as *constraints* especificadas pelo importador.

Num domínio comercial, o TS pode ser distribuído por vários objectos negociadores. Negociadores em domínios diferentes podem ser federados. As federações habilitam os sistemas em domínios diferentes a negociar a partilha dos serviços sem perder o controlo sobre as suas próprias políticas e serviços. Deste modo um determinado domínio pode partilhar informação com outros domínios com os quais está federado.

SERVIÇO DE CONTROLO DA CONCORRÊNCIA

O CCS (*Concurrency Control Service*) possibilita a coordenação do acesso de múltiplos clientes a recursos partilhados. Coordenar o acesso a um recurso significa que quando vários clientes tentam concorrentemente aceder ao mesmo recurso, qualquer conflito no acesso ao recurso entre os clientes é resolvido de forma a que o recurso permaneça num estado de consistência.

O utilização concorrente de um recurso é regulado através de fechos (*locks*). Cada fecho está associado a um único recurso e a um único cliente. A coordenação no acesso aos recursos é conseguida evitando que vários

clientes possuam ao mesmo tempo fechos sobre um mesmo recurso se eventualmente as suas actividades forem conflituosas. Então, antes de aceder ao recurso partilhado o cliente deve obter o fecho apropriado. O CCS define vários modos de fecho, que correspondem a diferentes categorias de acesso. Esta diversidade de fechos fornece uma maior flexibilidade na resolução de conflitos. Por exemplo, disponibilizando modos diferentes para leitura (*read*) e escrita (*write*) permite ao recurso suportar vários clientes concorrentes relativamente a transacções de leitura (*read-only*). O CCS define também *intention locks* que suportam fechos com vários níveis de granularidade.

Tabela II. Compatibilidade entre fechos.

	Modo Solicitado				
Modo Concedido	**IR**	**R**	**U**	**IW**	**W**
Intention Read (IR)					x
Read (R)				x	x
Upgrade (U)			x	x	x
Intention Write (IW)		x	x		x
Write (W)	x	x	x	x	x

De forma sucinta, o CCS define operações de sincronização (fechos) sobre objectos executados por transacções ou *threads*.

Serviço de Associação de Propriedades

O PS (*Property Service*) fornece a capacidade de dinamicamente associar informação textual a objectos/componentes como nomes, datas, referências e outras, sem recurso à IDL. Define operações para criar e manipular conjuntos de nome-valor ou de nome-valor-modo sendo os nomes *strings* IDL simples.

Serviço de Transacções

O TRS (*Transaction Service*) suporta vários modelos transaccionais, incluindo os modelos *flat* (obrigatório de acordo com a especificação) e *nested* (opcional). O TRS suporta a interoperabilidade entre os diferentes modelos de programação. Por exemplo, alguns utilizadores podem pretender incluir implementações de objectos em aplicações procedimentais já existentes e acrescentar implementações de objectos com código que recorre ao paradigma procedimental. Para que isto seja possível num ambiente transaccional, é necessário que o objecto e o código procedimental partilhem uma única transacção [OMG -1997].

SERVIÇO DE RELAÇÕES

O RS (*Relationship Service*) permite criar associações ou elos dinâmicos entre componentes que não se conhecem. Fornece também mecanismos para atravessar os elos que interligam grupos de componentes. Pode ser utilizado para verificar ou manter relações de integridade, relações de inclusão (*containment*), ou qualquer outro tipo de relações que se achar necessário entre componentes.

SERVIÇO DE INTERROGAÇÃO

O QS (*Query Service*) permite aos utilizadores e objectos a invocação de interrogações sobre outros objectos. As interrogações são declarações com predicados e incluem a capacidade de especificar valores de atributos, invocar operações arbitrárias e invocar outros serviços.

O QS é uma extensão do SQL (*Structured Query Language*) baseado na recente especificação do SQL3 e na linguagem proposta pelo OMG, a OQL (*Object Query Language*).

SERVIÇO DE ARMAZENAMENTO PERSISTENTE

O POS (*Persistent Object Service*) disponibiliza um conjunto de interfaces comuns aos mecanismos utilizados para manter e gerir o estado persistente dos objectos, nomeadamente uma interface para armazenar os objectos em ODBMS (*Object Database Management System*) ou em RDBMS (*Relational Database Management System*). O objecto tem a responsabilidade de gerir o seu próprio estado, contudo, pode utilizar ou delegar ao POS essa função. Pode haver diferentes clientes e diferentes implementações do POS a trabalhar conjuntamente. Isto é particularmente importante no caso do armazenamento pois por exemplo, mecanismos úteis para armazenamento de documentos podem já não o ser para bases de dados, ou mecanismos apropriados para computadores móveis podem não ser aplicáveis a *mainframes*.

A estrutura de implementação deste serviço é constituída pelos seguintes componentes (Figura 28):

- Objecto Persistente (PO - *Persistent Object*): Um objecto persistente é a definição de um objecto que utiliza as interfaces especificadas por este serviço para guardar o seu estado persistente.
- Identificador de Persistência (PID - *Persistent Identifier*): Identifica num *DataStore* onde estão armazenados os dados de um determinado objecto persistente.
- Gestor de Objecto Persistente (POM - *Persistent Object Manager*): Coordena os pedidos do objecto aos serviços de dados persistentes para guardar ou recuperar o seu estado de persistência.
- Serviço de Dados Persistente (PDS - *Persistent Data Service*): Fornece uma interface que permite aceder às bases de dados onde estão armazenados os objectos, de um modo que é transparente do tipo de base de dados utilizada.
- Protocolo: Definição da comunicação entre o PO e o PDS.
- Armazém de Dados (*DataStore*): Sistema de armazenamento dos dados do estado persistente do objecto.

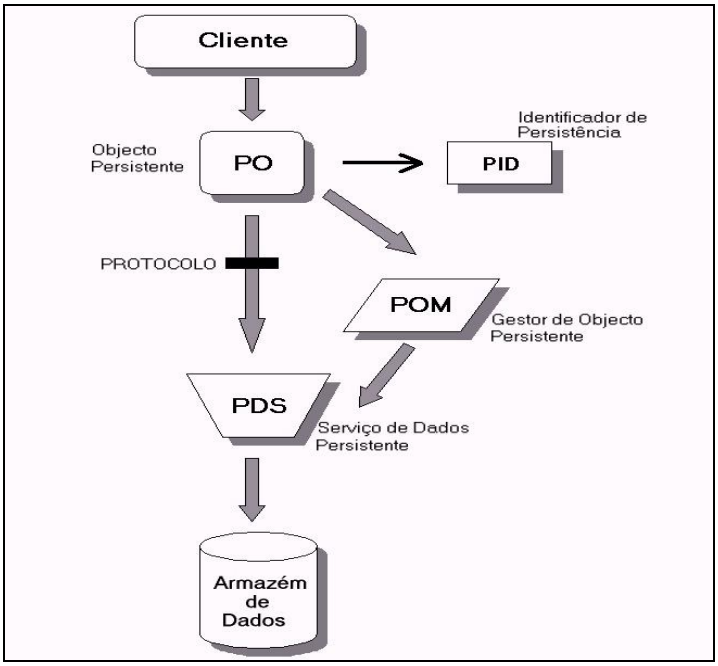

Figura 28 - Componentes principais do POS e suas iterações.

SERVIÇO DE EXTERIORIZAÇÃO

O EXS (*Externalization Service*) define protocolos e convenções para exteriorizar e interiorizar objectos. Exteriorizar um objecto é registar o estado do objecto numa *stream* de dados (na memória, num ficheiro, no disco, através da rede, etc.). O objecto exteriorizado pode existir durante espaços de tempo arbitrários, ser transportado por meios externos ao ORB e ser interiorizado num ORB diferente não conectado.

SERVIÇO DE GESTÃO DO CICLO DE VIDA

O LCS (*Life Cycle Service*) define operações para criar, copiar, mover e terminar componentes colocadas no *bus*, enquanto que o RS permite atravessar os elos que interligam grupos de objectos sem que estes sejam activados.

SERVIÇO DE LICENÇAS

O LS (*Licensing Service*) valida a licença de operação de um determinado componente por sessão, por nó, por realização e por local.

SERVIÇO DE TEMPO

O TS (*Time Service*) habilita o utilizador a obter o tempo actual conjuntamente com o erro associado estimado, isto é, disponibiliza interfaces para sincronização do tempo num ambiente de objectos distribuídos. Este serviço verifica a ordem pela qual os eventos ocorreram e calcula o intervalo entre dois eventos, isto é, permite a programação de acções e executa-as *à posteriori* e proporciona cálculos sobre o tempo decorrido entre acções.

3.12. FACILIDADES CORBA

Uma vez que, os serviços CORBA proporcionam utilitários para objectos individuais ou grupos de objectos, as facilidades CORBA proporcionam suporte de nível superior para aplicações. De facto existem dois tipos de facilidades CORBA, as facilidades horizontais que podem aplicar-se a qualquer domínio aplicacional e as facilidades verticais que são especializadas para segmentos de mercado individualizados (Figura 29). As facilidades CORBA são uma nova área da arquitectura CORBA cujas especificações ainda não estão completamente amadurecidas. Esta é no entanto uma situação que tende a alterar-se rapidamente.

As facilidades horizontais foram planeadas para abranger as seguintes áreas [Orfali - 1996]:

- Interface Utilizador: apresentação dos objectos e composição de documentos; ajuda, verificação de ortografia e sistemas de verificação da gramática; gestão da secretária; e outros. O sistema OpenDoc para composição de documentos foi aceite como uma facilidade CORBA para esta área.
- Serviço de Gestão da Informação: inclui a modelação da informação; armazenamento e recuperação de informação; codificação e tradução de objectos e documentos compostos; suporte para tempo e calendário.
- Serviço de Gestão de Sistema: inclui a gestão do ORB e a gestão de aplicações CORBA, isto é, define interfaces para gestão, instrumentação, configuração, instalação, operação e reparação de componentes distribuídos. O OMG escolheu um outro grupo de normalização, o X/Open, para fornecer esta especificação.
- Serviço de Gestão de Tarefas: inclui fluxo de trabalho; agentes; gestão de regras; elaboração de *scripts* e *e-mail*.

O OMG estabeleceu um conjunto de grupos de trabalho cada um para um dado domínio especifico, com a função de definirem as especificações das facilidades verticais CORBA. As áreas actuais de trabalho são: Saúde, Telecomunicações, Financeira, Produção, Negócios, Transportes e Comércio Electrónico. Cada grupo definirá interfaces e serviços que ajudarão os programadores a escrever e gerir aplicações em cada uma das áreas.

Note-se que não existem regras absolutas relativamente a estes aspectos, um utilitário por exemplo, poderá eventualmente ser considerado um serviço CORBA ou uma facilidade CORBA. Como exemplo, o Serviço de Licenças poderia muito bem ser conotado como uma facilidade CORBA.

Finalmente note-se que cada serviço CORBA ou facilidade CORBA são uma especificação mais do que uma implementação de referência devendo contudo as suas interfaces estar definidas em IDL.

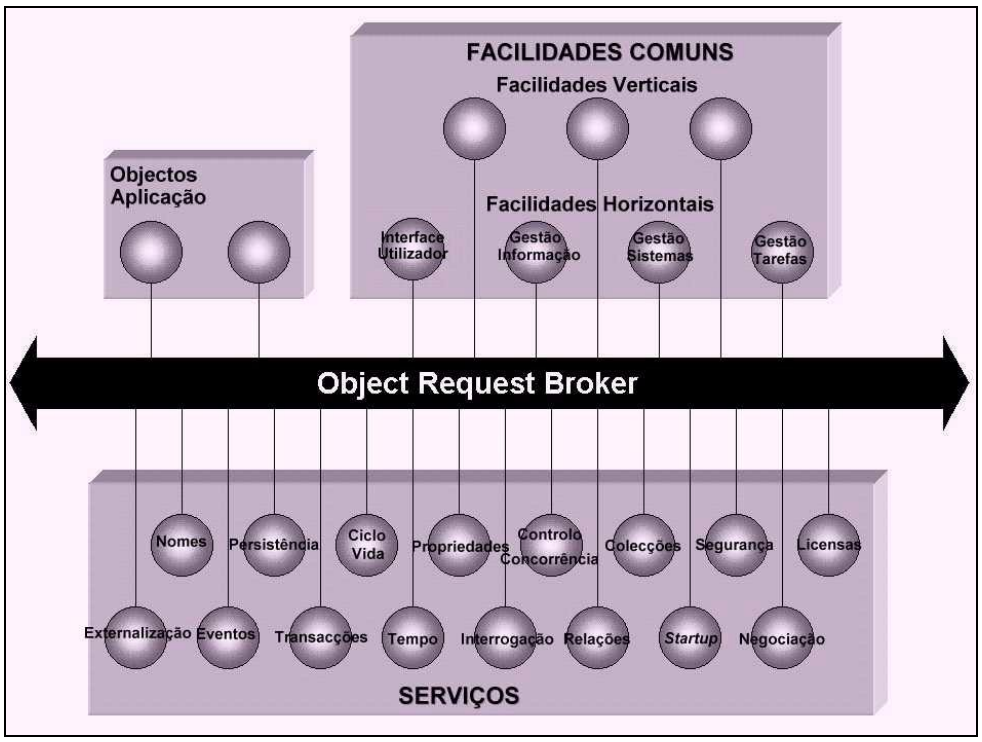

Figura 29 - Arquitectura de Gestão de Objectos.

Teoricamente a arquitectura CORBA é a melhor plataforma intermediária cliente/servidor jamais definida, contudo na prática, esta será unicamente tão boa quão bons forem os produtos que a implementem [Orfali - 1998].

3.13. ESTADO DE DESENVOLVIMENTO DE ALGUMAS PLATAFORMAS ORB

O estado de desenvolvimento dos ORBs ainda se encontra numa fase muito embrionária, prevendo-se contudo uma rápida evolução. Numa tentativa de conquista de mercado todas as empresas envolvidas neste projecto tentam fazer vingar as suas plataformas implementando o maior número possível de funcionalidades.

É então objectivo deste capítulo fornecer uma lista das implementações CORBA mais utilizadas, descrevendo para cada uma as principais características e funcionalidades.

A informação a seguir descrita foi cedida pelas respectivas empresas em Março de 1999.

Os aspectos considerados nesta avaliação foram:

- Características da parte nuclear do ORB.
- Plataformas suportadas.
- Serviços CORBA suportados.

Atendendo ao elevado número de plataformas disponíveis e uma vez que era difícil fazer o estudo para todas, foram considerados os seguintes ORBs pela sua importância no mercado actual. A sequência de apresentação dos ORBs é ocasional não tendo qualquer ordem específica.

- **CORBAPlus** da Expertsoft, Inc. - San Diego, USA [CORBAPlus - 1999].
- **DAIS** da PeerLogic, Inc. - San Francisco, USA [DAIS - 1999].
- **OAK** da Paragon Software, Inc. - Vienna, USA [OAK - 1999].
- **Orbix** da IONA Technologies PLC - Dublin, Ireland [Orbix - 1999].
- **omniORB** dos AT&T Laboratories - Cambridge, England [omniORB - 1999].
- **RCP-ORB** da Nortel Networks Corp. - Ontario, Canada [RCPORB - 1999].
- **ORBexpress** da Objective Interface Systems, Inc. - Virginia, USA [ORBex - 1999].
- **TIB/ObjectBus** da Tibco, Inc. - Palo Alto, USA [TIB/OB - 1999].

Nas tabelas seguintes, o caracter "x" significa característica suportada.

3.13.1. Características da Parte Nuclear do ORB

Tabela III. Linguagens suportadas.

	Linguagens							
Designação ORB	**IDL**	**C++**	**C**	**Smalltalk**	**Ada**	**Java**	**OLE[2]**	**COBOL**
CORBAPlus	x	x		x		x	x	
DAIS	x	x	x			x	x	
OAK	x	x				x		
Orbix	x	x		x	x	x	x	x
omniORB	x	x						
RCP-ORB	x	x	x	x				
ORBexpress	x	x			x			
TIB/ObjectBus	x	x	x			x		

[2] OLE - Visual Basic, PowerBuilder, Delphi.

Tabela IV. Protocolos suportados.

	PROTOCOLOS	
Designação ORB	**IIOP**	**DCE**
CORBAPlus	x	
DAIS	x	
OAK	x	
Orbix	x	
omniORB	x	
RCP-ORB	x	
ORBexpress	x	
TIB/ObjectBus	x	

Legenda:

IIOP - *Internet Inter-ORB Protocol*.

DCE - *Distributed Communication Environment*.

Tabela V. Núcleo.

	NÚCLEO			
Designação ORB	**DII**	**DSI**	**IR**	**BOA**
CORBAPlus	x	x	x	x
DAIS	x	x	x	x
OAK	x	x	x	x
Orbix	x	x	x	x
omniORB	x	x	x	x
RCP-ORB				x
ORBexpress	x	x	x	x
TIB/ObjectBus	x	x	x	x

Legenda:

DII - *Dynamic Invocation Interface*.

DSI - *Dynamic Skeleton Interface*.

IR - *Interface Repository*.

BOA - *Basic Object Adapter*.

3.13.2. SERVIÇOS SUPORTADOS

Tabela VI. Serviços suportados.

	SERVIÇOS												
Designação ORB	**NS**	**LCS**	**ES**	**TS**	**CCS**	**EXS**	**POS**	**TRS**	**QS**	**CS**	**TS**	**PS**	**SS**	**LS**
CORBAPlus	x		x											
DAIS	x	x	x	x				x					x	
OAK	x		x											
Orbix	x		x	x				x					x	
omniORB	x			x	x							x		
RCP-ORB	x	x	x											
ORBexpress	x													
TIB/ObjectBus	x		x											

Legenda:

NS - *Naming Service*/Serviço de Nomes.

LCS - *Life Cycle Service*/Serviço de Gestão do Ciclo de Vida.

ES - *Event Service*/Serviço de Eventos.

TS - *Trader Service*/Serviço de Negociação.

CCS - *Concurrency Control Service*/Serviço de Controlo da Concorrência.

EXS - *Externalization Service*/Serviço de Exteriorização.

POS - *Persistent Object Service*/Serviço de Armazenamento Persistente.

TRS - *Transaction Service*/Serviço de Transacções.

QS - *Query Service*/Serviço de Interrogação.

CS - *Collections Service*/Serviço de Colecções.

TS - *Time Service*/Serviço de Tempo.

PS - *Property Service*/Serviço de Associação de Propriedades.

SS - *Security Service*/Serviço de Segurança.

LS - *Licensing Service*/Serviço de Licenças.

3.13.3. PLATAFORMAS SUPORTADAS

Tabela VII. Plataformas suportadas.

	SISTEMAS OPERATIVOS												
Designação ORB	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
CORBAPlus	x	x	x	x			x	x					x[3]
DAIS	x	x	x	x			x	x			x		x[4]
OAK	x	x			x		x						
Orbix	x	x	x	x		x	x	x	x	x	x	x	x[5]
omniORB	x	x	x	x	x	x	x	x		x	x		x[6]
RCP-ORB	x	x			x		x	x					x[7]
ORBexpress	x	x	x	x		x	x				x		
TIB/ObjectBus	x	x	x				x						

Legenda:

A - Solaris

B - HPUX

C - AIX

D - DEC

E - Linux

F - SGI

G - NT

H - Windows 95/98

I - OS/2

J - Mac

K - VMS

L - MVS

M - Outros

3.14. EVOLUÇÃO DA ARQUITECTURA - CORBA 3.0

Antes de ser atingido o estado actual de desenvolvimento da arquitectura CORBA - CORBA 2.3 - esta sofreu várias revisões ao longo dos anos.

É objectivo deste ponto fazer uma breve resenha do processo evolutivo desta arquitectura assinalando as datas e factos ocorridos mais relevantes.

No final são focadas as principais características a ser introduzidas nesta arquitectura pela nova versão 3.

[3] SunOS.

[4] SunOS, Unixware e SCOUnix.

[5] SunOS, Sinix, Unixware e Ultrix.

[6] SunOS

[7] SunOS

As informações a seguir apresentadas têm como base dois documentos do OMG:

- History of CORBA [OMG - 1999].
- OMG in Motion [Siegel - 1999].

que podem ser encontrados respectivamente em:

- http://www.omg.org/corba/corbahistory.html.
- http://www.omg.org.

A história da arquitectura CORBA começa em Outubro de 1991 quando o OMG decide lançar a primeira versão (1.0) da arquitectura. Apresentava como principais características:

- Inclusão do Modelo de Objectos CORBA.
- Inclusão da IDL, apenas com o mapeamento para a linguagem C.
- Inclusão de APIs para suporte do DII e do IR.

Em Fevereiro de 1992 surge a versão 1.1. Embora tenha sido a segunda versão da especificação da arquitectura, foi a primeira que foi amplamente divulgada pelo OMG.

Corrige muitas das ambiguidades da especificação original e são-lhe acrescentadas as seguintes funcionalidades:

- Inclusão das interfaces para o BOA e Gestão de Memória.
- É clarificada a função do IR assim como corrigidas as ambiguidades no Modelo de Objectos.

A versão 1.2 é lançada em Dezembro de 1993 tendo como finalidade a correcção de algumas ambiguidades detectadas na versão anterior relativamente a aspectos como gestão de memória e comparação de referências de objectos.

Em Agosto de 1996, numa altura em que os objectos distribuídos começam a ganhar cada vez mais representatividade no seio das TIs surge a versão 2.0. Esta versão é o resultado da decisão do OMG em reexaminar o seu modelo de objectos e introduzir-lhe as seguintes características:

- DSI.
- Extensões ao IR.
- Arquitectura de interoperabilidade (GIOP, IIOP e DCE CIOP).
- Serviços de segurança e transacções.
- Interoperação com OLE2/COM.
- Mapeamento IDL para mais duas linguagens - C++ e Smalltalk.

A versão 2.1 surge em Agosto de 1997 e trouxe consigo melhoramentos ao nível da segurança mais especificamente ao nível do protocolo IIOP seguro

e IIOP sobre SSL (*Secure Sockets Layer*)). São adicionados mais dois mapeamentos IDL neste caso para as linguagens COBOL e Ada e revista a questão da interoperabilidade.

Em Fevereiro de 1998 é tornada pública a versão 2.2. Esta nova versão tem como aspectos relevantes a introdução de interfaces para suporte do POA e o suporte de interoperação com a arquitectura DCOM. Surge também nesta versão o mapeamento IDL para a linguagem Java.

No final do verão de 1998 surge a versão 2.3. Considerada de pouca importância, teve como finalidade o melhoramento de alguns aspectos pontuais das especificações nomeadamente:

- O mapeamento IDL/Java.
- A portabilidade do ORB relativamente à IDL/Java.
- A parte nuclear da arquitectura.
- A interoperabilidade entre ORBs.
- A segurança.

Espera-se a todo momento que seja tornada pública uma nova versão desta arquitectura, a **CORBA 3.0**. O OMG não fazia alterações tão significativas na especificação da arquitectura desde 1996, altura em que foi introduzida a interoperabilidade na versão 2.0.

De acordo com o OMG as novas características e especificações a ser introduzidas na CORBA 3.0 podem ser divididas em três categorias principais:

- Integração Java e Internet.
- Qualidade de Serviço.
- Arquitectura CORBAcomponent.

3.14.1. INTEGRAÇÃO JAVA E INTERNET

Três novas especificações melhoram a integração da arquitectura CORBA com as cada vez mais populares linguagem de programação Java e Internet nomeadamente:

- Mapeamento Java/IDL.
- Especificação *Firewall*.
- INS (*Interoperable Naming Service*).

MAPEAMENTO JAVA/IDL[8]

Ao já suportado mapeamento IDL/Java a arquitectura CORBA nesta sua nova versão junta o mapeamento Java/IDL. Este novo mapeamento define interfaces IDL para objectos Java tal que:

- Permite que os programadores Java utilizem o protocolo IIOP para as suas invocações remotas.
- Permite que servidores Java sejam invocados por clientes CORBA, clientes estes escritos em qualquer uma das linguagens suportadas pela arquitectura CORBA.

ESPECIFICAÇÃO FIREWALL[9]

A especificação *firewall* da arquitectura CORBA 3.0 define *firewalls* de nível transporte, *firewalls* de nível aplicacional e uma conexão GIOP bidireccional útil para *callbacks* e notificações de eventos.

As *firewalls* nível transporte vão actuar ao nível do TCP (*Transport Control Protocol*). Através da definição dos portos 683 para o IIOP e 684 para o IIOP sobre SSL, esta especificação permite aos administradores configurar as *firewalls* de forma a controlarem o tráfego CORBA sobre o protocolo IIOP.

Na arquitectura CORBA os objectos necessitam com alguma frequência fazer *callbacks* ou notificar o cliente que o invocou. Se tivermos em consideração que as conexões especificadas na CORBA apenas transportam as invocações num sentido, uma *callback* implica o estabelecimento de uma segunda conexão TCP para circulação do tráfego no sentido oposto. Nesta nova especificação é permitido à mesma conexão IIOP o transporte de invocações em ambos os sentidos de acordo com certas restrições de forma a que a segurança não seja comprometida. A principal vantagem associada a este facto é a possibilidade de utilização da CORBA através da Internet sem comprometer a questão da segurança.

INTEROPERABLE NAMING SERVICE[10]

A referência de objectos é uma das características mais importantes da arquitectura CORBA. O INS (*Interoperable Naming Service*) define um novo formato URL (*Uniform Resource Locator*) de referência de objectos, o iioploc, que vem complementar o IOR[11] (*Interoperable Object Reference*). Este novo formato pode ser utilizado em programas para a obtenção de serviços definidos em localizações remotas incluindo o

[8] A especificação pode ser consultada em:
http://www.omg.org/cgi-bin/doc?orbos/98-04-04.

[9] A especificação pode ser consultada em:
http://www.omg.org/cgi-bin/doc?orbos/98-05-04.

[10] A especificação pode ser consultada em:
http://www.omg.org/cgi-bin/doc?orbos/98-10-11.

[11] IOR (*Interoperable Object Reference*). O GIOP define um formato para as IORs. O ORB deve criar uma IOR (de uma referência de objecto) sempre que uma referência de objecto é transmitida entre ORBs. As IORs associam uma colecção de perfis às referências de objectos, servindo estes perfis para descrever o objecto e a forma de o contactar. Mais concretamente o perfil disponibiliza dados auto-descritivos que identificam o domínio ORB ao qual a referência está associada e os protocolos que suporta.

próprio Serviço de Nomes da arquitectura CORBA. Um segundo formato URL, o iiopname, tem por função a invocação do Serviço de Nomes remoto utilizando para tal o nome que o utilizador junta à URL obtendo a IOR para o objecto pretendido, isto é, com o nome especificado.

A sintaxe do formato URL do iioploc é da forma: iioploc://www.omg.org/NameService. Neste exemplo é determinado o Serviço de Nomes CORBA a ser executado na máquina cujo endereço IP corresponde ao domínio www.omg.org.

3.14.2. QUALIDADE DE SERVIÇO

As especificações QoS (*Quality of Service*) foram planeadas para gerir e seleccionar as várias possibilidades no que respeita a transporte de acordo com as necessidades das aplicações.

Acerca deste tópico foram considerados os aspectos:

- Mensagens Assíncronas.
- CORBA Mínima.
- CORBA Tempo-Real.
- Tolerância a Falhas.

MENSAGENS ASSÍNCRONAS[12]

A nova especificação relativamente a mensagens assíncronas define um conjunto de modos de invocação assíncronos e independentes do tempo permitindo que tanto as invocações dinâmicas como as estáticas utilizem todos os modos. No que respeita aos resultados das invocações estes podem ser recuperados quer por *pooling* quer por *callback*, sendo a escolha efectuada pela forma utilizada pelo cliente na invocação original.

CORBA MÍNIMA[13]

A CORBA Mínima foi primeiramente projectada para sistemas embebidos. Logo que estes sistemas sejam finalizados e se inicie a sua produção, tendo em conta que as suas interacções com o exterior da rede são previsíveis, não têm necessidade de implementar os aspectos dinâmicos da CORBA como é o caso da DII ou do IR, pelo que estes não se encontram incluídos na CORBA Mínima.

CORBA TEMPO-REAL[14]

A CORBA Tempo-Real uniformiza o controle de recursos nomeadamente *threads*, protocolos, conexões e outros recorrendo para tal a modelos de prioridade para conseguir comportamentos previsíveis para os ambientes tempo-real. O agendamento dinâmico (*dynamic scheduling*) uma vez que não fazia parte da especificação inicial está agora a ser acrescentado.

---

[12] A especificação pode ser consultada em:
http://www.omg.org/cgi-bin/doc?orbos/98-05-05.

[13] A especificação pode ser consultada em:
http://www.omg.org/cgi-bin/doc?orbos/98-08-04.

[14] A especificação pode ser consultada em:
http://www.omg.org/cgi-bin/doc?orbos/98-02-12.

TOLERÂNCIA A FALHAS

Esta especificação encontra-se ainda em fase de desenvolvimento, podendo contudo o RFP ser consultado em http://www.omg.org/techprocess/meetings/schedule/Fault_Tolerance_RFP.html.

3.14.3. COMPONENTES

Este ponto compreende:

- Passagem de Objectos por Valor.
- CORBAComponents.
- Linguagem de *Scripting* CORBA.

PASSAGEM DE OBJECTOS POR VALOR[15]

A Passagem de Objectos por Valor - *Termed valuetypes* - vem trazer novas potencialidades à arquitectura CORBA que antes unicamente suportava passagem de objectos por referência. Esta nova capacidade é conseguida através da passagem dos objectos como parâmetros nas operações CORBA. O desempenho geral é melhorado pois a passagem de objectos por valor é na maior parte da vezes mais eficiente que a passagem de objectos por referência.

CORBACOMPONENTS[16]

Este é considerado o principal desenvolvimento da arquitectura CORBA depois da introdução em Agosto de 1996 na versão 2.0 do protocolo IIOP.

Duas das principais partes da CORBAComponents são:

- Um Ambiente Contentor que engloba transacções, segurança e persistência.
- Integração com EJB (*Enterprise Java Beans*) e Controlos ActiveX.

O Ambiente Contentor CORBAComponents é persistente, transaccional e seguro. Na perspectiva do programador estas funções estão embutidas no ambiente facultando desta forma um nível de abstracção mais elevado que o disponibilizado pelos próprios Serviços CORBA. Assim os programadores podem orientar os seus conhecimentos para questões especificas das aplicações e não para questões da arquitectura.

Em resumo este modelo disponibiliza um mecanismo mais completo para descrever entidades de software orientadas por objectos e para as agrupar para formar aplicações. Este modelo será capaz de

[15] A especificação pode ser consultada em: http://www.omg.org/cgi-bin/doc?orbos/98-01-18.

[16] A especificação pode ser consultada em: http://www.omg.org/techprocess/meetings/schedule/CORBA_Component_Model_RFP.html, não estando contudo ainda concluída.

interoperar com outras tecnologias de componentes onde se incluem as tecnologias Java Beans e Controlos ActiveX, o que significa que tanto a Java Beans com a ActiveX podem ser instalados em CORBAComponents e agir como tal.

LINGUAGEM DE SCRIPTING CORBA[17]

A função da Linguagem de *Scripting* CORBA é a composição de componentes em aplicações de uma forma transparente. De uma forma geral as linguagem de *scripting* são de mais fácil compreensão pelo que são acessíveis a uma maior audiência. A grande vantagem é que geralmente as ferramentas de *scripting* não necessitam ser compiladas pelo que o código pode ser imediatamente executado logo a seguir à sua especificação. Este tipo de linguagens permite aos construtores de aplicações a criação de novas *scripts* em tempo de execução.

[17] Esta especificação pode ser consultada em: http://www.omg.org/techprocess/meetings/schedule/CORBA_Scripting_Language_RFP.html não estando contudo ainda concluída.

CAPÍTULO 4

4. DCOM - DISTRIBUTED COMPONENT OBJECT MODEL

4.1. INTRODUÇÃO

A história do DCOM (*Distributed Component Object Model*) tem inicio em 1990 quando a Microsoft introduz a tecnologia OLE (*Object Linking and Embedding*) para suportar a integração de várias aplicações e diferentes tipos de dados multimédia numa mesma ferramenta de composição de documentos. Esta tecnologia apresentava contudo algumas deficiências e tinha como suporte o DDE (*Dynamic Data Exchange*) [Orfali - 1996].

Na versão 2 do OLE (1993), também designada por OLE Automation, a maioria das deficiências da versão anterior são resolvidas recorrendo a uma nova tecnologia de encapsulação de objectos designada COM (*Component Object Model*).

O COM passa então a ser publicitado pela Microsoft como um modelo de programação orientado por objectos e desenhado para facilitar a interoperabilidade do software, isto é, permitir que duas ou mais aplicações ou componentes cooperem facilmente entre si, mesmo que tenham sido desenvolvidas por diferentes vendedores, em diferentes alturas, em diferentes linguagens de programação, ou mesmo se estão a ser executadas em máquinas diferentes por sua vez a executar diferentes sistemas operativos. Para suportar as características de interoperabilidade anunciadas, o COM define e implementa mecanismos que permitem às aplicações interligarem-se como objectos de software, no entanto uma vez estabelecida a comunicação o COM já não é mais necessário ficando o cliente e o objecto a comunicar directamente (Figura 30).

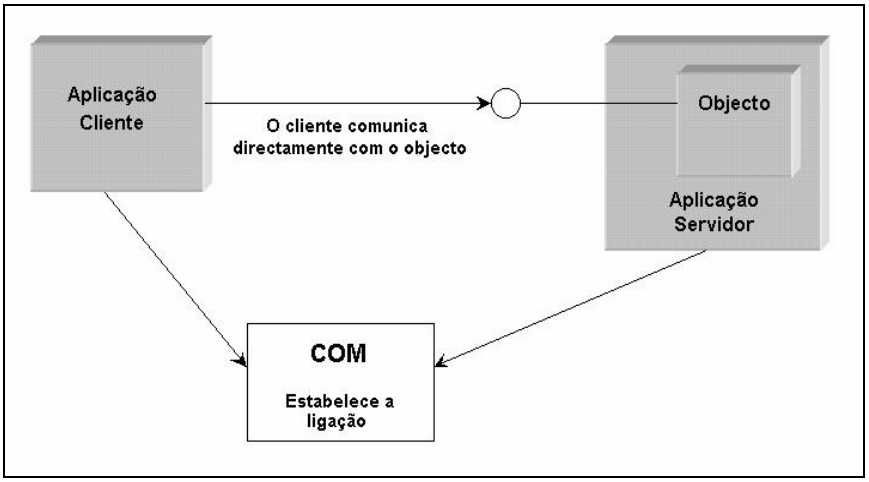

Figura 30 - Uma vez estabelecida a comunicação entre o cliente e o objecto através do COM, estes podem comunicar directamente.

Algumas das características atribuídas ao COM são [MSFT - 1995]:

- É uma especificação que define as normas a seguir para criar componentes COM capazes de interoperar entre si. Estas normas descrevem como são os objectos e como se comportam.
- É um conjunto de serviços (APIs) que são fornecidos por uma biblioteca (biblioteca COM). No caso das plataformas Win32 (Windows 98, Windows NT) é parte integrante do sistema operativo enquanto que para outros sistemas operativos está disponível num pacote próprio.
- Permite a programação modular pois os componentes COM são encapsulados em ficheiros EXEs ou DLLs (*Dynamic Link Library*). O COM disponibiliza o mecanismo de comunicação para que componentes de diferentes aplicações possam comunicar.
- É orientado por objectos porque os componentes COM são objectos - possuem identidade, estado e comportamento.
- Permite a personalização e actualização das aplicações. Os componentes ligam-se uns aos outros dinamicamente através do COM que por sua vez define a forma de os localizar e identificar as suas funcionalidades. Desta forma podem trocar-se componentes sem haver necessidade de recompilar toda a aplicação.
- Transparência da distribuição. Esta transparência consiste no facto de as aplicações poderem ser escritas sem a preocupação da localização dos seus componentes.
- Independência da linguagem de programação. Os componentes COM podem ser escritos em qualquer linguagem uma vez que o COM define uma norma binária para interoperabilidade. Desta forma qualquer linguagem que entenda esta norma, pode criar e utilizar objectos COM. O número de linguagens e ferramentas que suportam o COM tem vindo a aumentar disponibilizando a Microsoft ligações para todos os seus ambientes de desenvolvimento: Visual C++, Visual Basic, Visual J++, Delphi/Pascal entre outras [Orfali - 1998].

Esta tecnologia apresentava contudo uma grande desvantagem que era o facto de poder ser unicamente utilizada de forma isolada, isto é, num mesmo computador pois não possuía qualquer protocolo para interoperabilidade em rede.

Posteriormente surge o DCOM, também designado por *COM with a longer wire*, que é uma extensão do COM e que vem resolver o problema do isolamento, isto é, vai permitir transportar as facilidades COM para uma rede de computadores.

O agrupamento destas três tecnologias por parte da Microsoft - COM, DCOM e OLE Automation - dá origem à designação ActiveX.

Actualmente a designação desta plataforma é COM+, que representa a junção da tecnologia MTS (*Microsoft Transaction Server*) ao COM/DCOM. Este aspecto irá ser focado no final deste capítulo num ponto reservado ao COM+ onde serão abordadas de forma sucinta as suas diferenças relativamente às tecnologias que lhe dão origem - COM e DCOM.

Nos pontos seguintes vão ser apresentados de forma resumida os aspectos considerados de maior relevância da tecnologia DCOM. Nesta apresentação a designação COM aparece com alguma frequência, pois o DCOM tem como suporte esta tecnologia.

4.2. A TECNOLOGIA DCOM

O DCOM é uma extensão do COM que define a interacção entre os componentes e os seus clientes de modo a que estes se possam ligar sem a ajuda de um sistema intermediário (Figura 31).

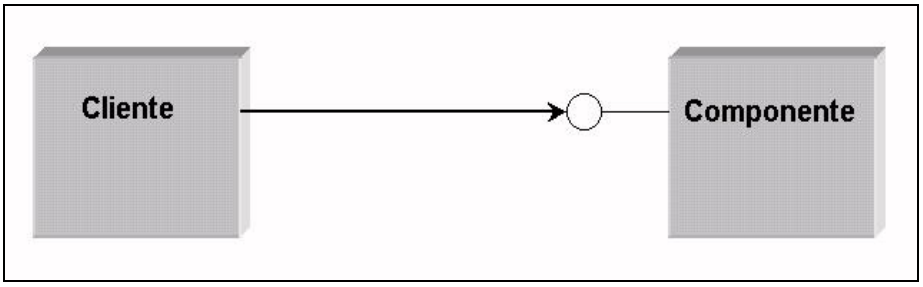

Figura 31 - Interacção cliente - componente.

Nos sistemas operativos actuais os processos estão protegidos uns dos outros, pelo que um cliente que precise de comunicar com um componente num processo diferente não o pode fazer directamente tendo que utilizar uma forma de comunicação entre processos fornecida pelo sistema operativo. O DCOM disponibiliza esta capacidade de comunicação de uma forma transparente interceptando as invocações do cliente e direccionando-as para o componente no outro processo (Figura 32).

No caso do cliente e do componente se situarem em máquinas diferentes, o DCOM substitui a comunicação local entre processos por um protocolo de rede.

A Figura 32 ilustra a arquitectura DCOM. O bloco COM fornece serviços orientados a objectos aos clientes e aos componentes e utiliza o DCE RPC e mecanismos de segurança para gerar pacotes de rede de acordo com a norma DCOM.

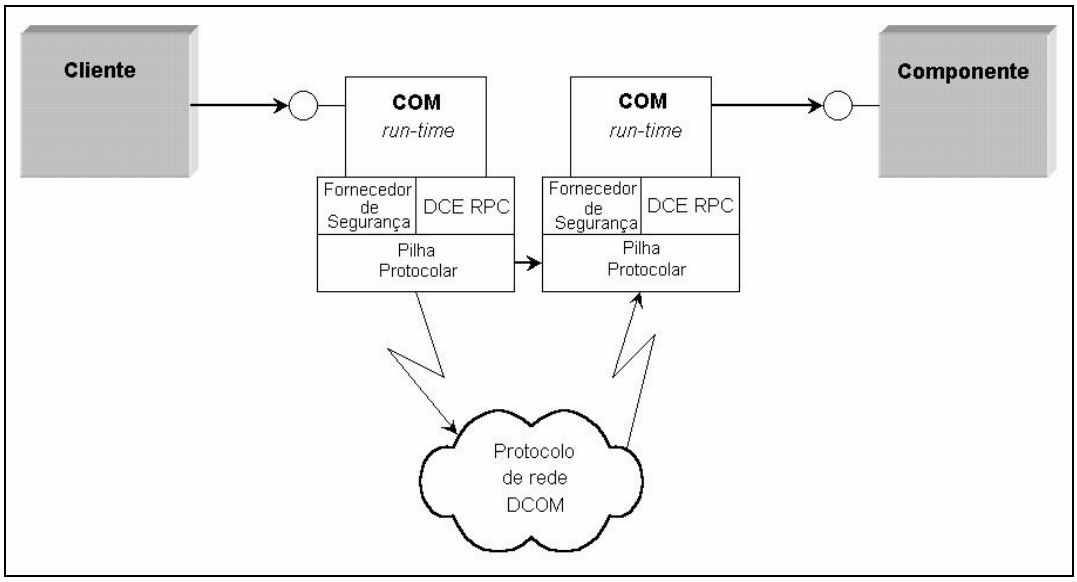

Figura 32 - Arquitectura DCOM - Comunicação entre componentes DCOM.

4.2.1. SIMBOLOGIA DCOM

A representação gráfica padrão para objectos e interfaces consiste em desenhar cada interface num objecto como um conector. Normalmente as interfaces são desenhadas fora do rectângulo que representa o objecto, quer do lado esquerdo quer do direito. Os nomes das interfaces são colocados perto dos conectores (Figura 33).

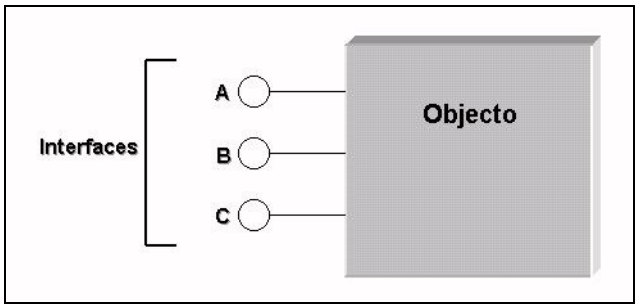

Figura 33 - Representação gráfica de um objecto que suporta 3 interfaces diferentes: A, B e C.

Note-se que as interfaces são os mecanismos utilizados pelo objecto para mostrar as suas funcionalidades, consistindo numa tabela de ponteiros para funções implementadas pelo objecto que são os métodos dessa interface (Figura 34).

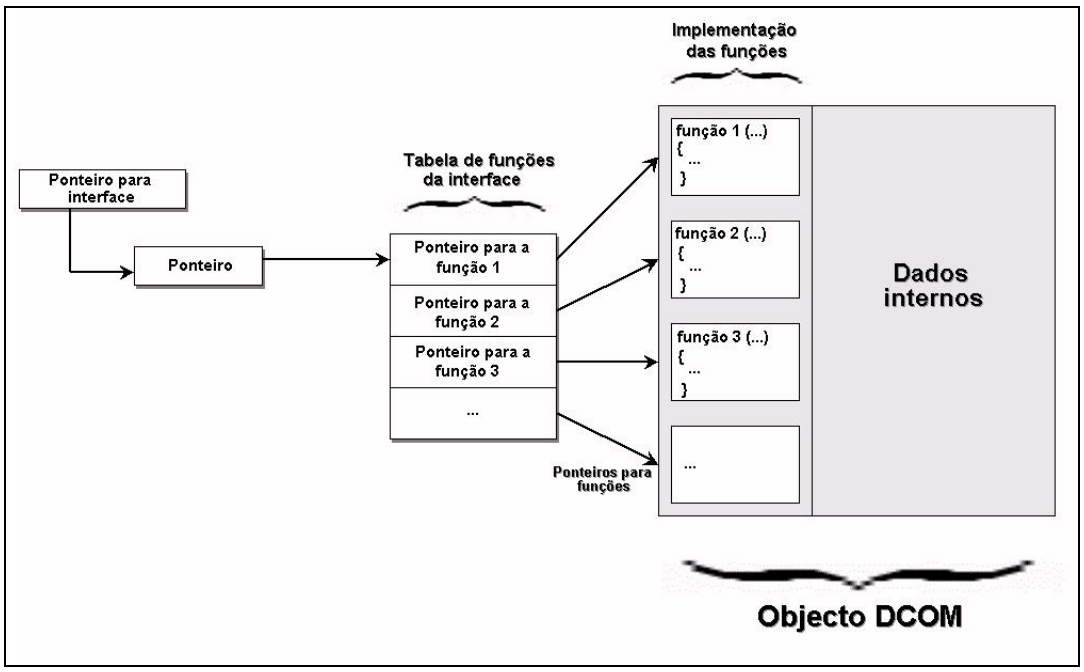

Figura 34 - Estrutura das interfaces.

Se na representação gráfica o cliente não estiver presente, por norma representam-se as interfaces do lado esquerdo como se mostra na Figura 33. Caso contrário são representadas na direcção do cliente, assumindo que este possui pelo menos um ponteiro para uma interface desse objecto (Figura 35).

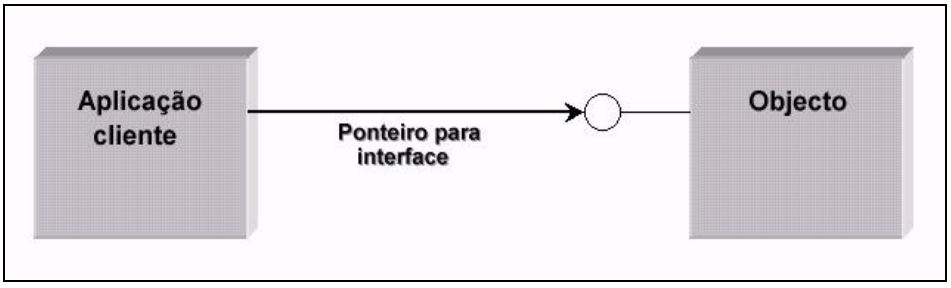

Figura 35 - As interfaces dirigem-se para os clientes que estão ligados ao objecto.

O cliente pode ele próprio implementar um objecto que disponibilize funções a outro objecto. Nestes casos, o cliente é um criador de objectos e o objecto é um cliente (Figura 36).

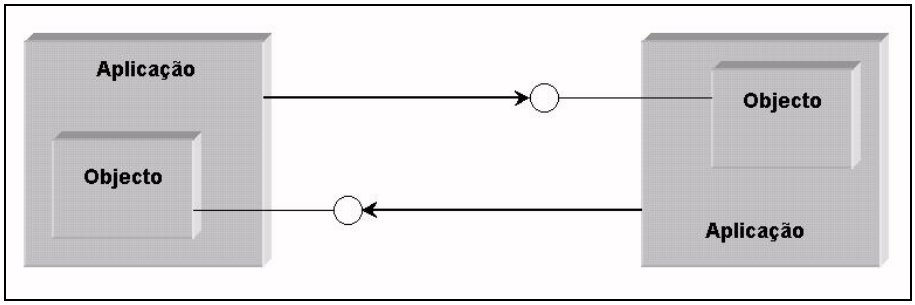

Figura 36 - Duas aplicações podem ligar-se aos seus objectos mutuamente, dirigindo as suas interfaces uma para a outra.

A interface *IUnknown* é a interface base de todas as interfaces DCOM e por convenção representa-se no topo do objecto (Figura 37). Define os métodos *QueryInterface*, *AddRef* e *Release* (Figura 38), em que o primeiro método permite ao utilizador obter um ponteiro para uma das interfaces do objecto e os dois últimos implementam a contagem de referências do objecto.

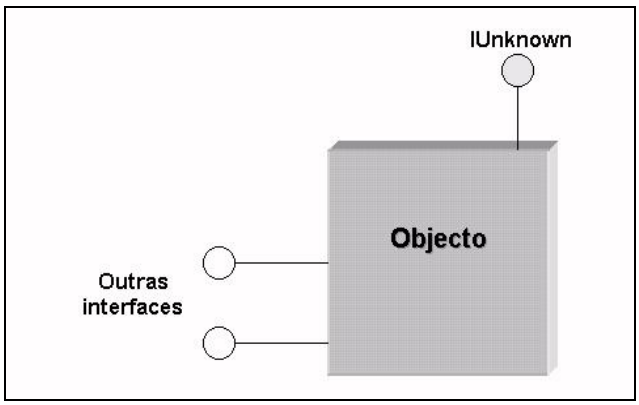

Figura 37 - A interface *IUnknown* é representada no topo do objecto.

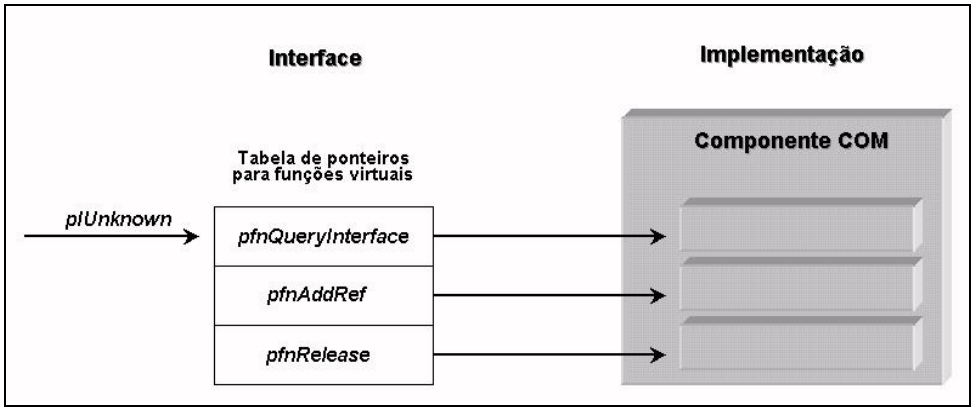

Figura 38 - Interface *IUnknown*.

OBJECTOS COM VÁRIAS INTERFACES

No DCOM um objecto pode suportar várias interfaces, isto é, disponibilizar ponteiros para mais do que um grupo de funções. A existência de múltiplas interfaces foi a principal inovação do COM, a qual permitiu resolver o problema da incompatibilidade entre versões [Orfali - 1998].

Todavia a existência de múltiplas interfaces levanta uma questão. Quando um cliente inicialmente acede a um objecto, é lhe devolvido um e apenas um ponteiro para a interface. A pergunta que se pode colocar é então a seguinte: Como é que o cliente pode aceder às outras interfaces do mesmo objecto?

A resolução desta questão encontra-se na função *QueryInterface* que está presente em todas as interfaces DCOM e pode ser invocada em qualquer interface por polimorfismo. A função *QueryInterface* é o suporte para um processo denominado negociação de interface onde o cliente questiona o objecto sobre quais os serviços que este lhe pode fornecer, invocando a função *QueryInterface* e passando-lhe o identificador único que representa os serviços de interesse.

Quando o cliente acede a um objecto recebe um ponteiro para a interface *IUnknown*. Como a interface *IUnknown* não tem capacidade para executar qualquer operação, pois não possui funções para esse efeito, as operações são executadas por outras interfaces. O cliente deve portanto negociar as interfaces com os objectos invocando a função *QueryInterface*, para obter aquela que execute as operações desejadas. Se o objecto aceitar o pedido do cliente a função *QueryInterface* devolve um novo ponteiro para a interface solicitada pelo cliente. Por outro lado, se o objecto rejeitar o pedido do cliente a função *QueryInterface* devolve um ponteiro nulo, significando erro, e assim o cliente não pode aceder às funções desejadas uma vez que não possui o ponteiro correspondente.

Quando um objecto rejeita um pedido à função *QueryInterface* o cliente fica impedido de solicitar ao objecto a realização das operações que essa interface poderia oferecer. O cliente deve ter um ponteiro para a interface para poder invocar as funções disponibilizadas por esta. Se o objecto recusar a cedência de um ponteiro, o cliente deverá estar prevenido. Contrariamente a outros sistemas orientados por objectos onde só é possível saber se uma função trabalha depois de esta ser chamada, no DCOM a função *QueryInterface* permite ultrapassar este problema, uma vez que permite saber se uma determinada função existe ou não antes de a invocar [MSFT - 1995].

4.2.2. MODELO DE OBJECTOS

Um Modelo de Objectos define conceitos que permitem facilitar o desenvolvimento de aplicações distribuídas. Sob este ponto de vista poder-se-á afirmar que a arquitectura DCOM é também um Modelo de Objectos atendendo a que [MSFT - 1995]:

- Utiliza identificadores globais únicos para identificar classes de objectos e as interfaces por estes suportadas.
- Disponibiliza métodos para reutilização do código.
- Possui um único modelo de programação para interacção de componentes de software, quer no mesmo processo, quer em processos diferentes, quer remotamente sobre uma rede.
- O ciclo de vida dos objectos é encapsulado através da contagem de referências.
- Fornece um suporte flexível para segurança ao nível do objecto.

IDENTIFICADORES GLOBAIS ÚNICOS

A identificação dos objectos num sistema distribuído deve ser garantida através de identificadores únicos.

O DCOM utiliza GUIDs (*Globally Unique Identifier*), que consistem em números inteiros de 128 bits, para garantir que cada classe de objectos e interfaces sejam identificadas de forma única evitando desta forma que componentes DCOM se liguem acidentalmente a objectos que não são os pretendidos. Estes identificadores são os mesmos que os UUIDs definidos pelo DCE.

REUTILIZAÇÃO E HERANÇA

Um factor importante num modelo de objectos é a possibilidade de reutilização e a possibilidade de extensão dos componentes. Uma forma de reutilização é a utilização de herança, ou seja, ao criar-se um novo objecto este pode herdar algumas das suas funcionalidades de outro componente enquanto redefine outras funções. Este mecanismo tem-se mostrado bastante útil na construção de determinados tipos de aplicações, no entanto, e depois de vários anos de experiência, este método não se tem mostrado muito robusto em sistemas de componentes de software de grande dimensão e em constante evolução. Por esta razão o DCOM define outros mecanismos de reutilização [Orfali - 1998].

Nos mecanismos a seguir descritos, o objecto que é reutilizado é denominado objecto interior (*inner object*) e o objecto que faz uso deste é denominado objecto exterior (*outer object*).

MECANISMO DE CONTENÇÃO/DELEGAÇÃO

Neste mecanismo (Figura 39) o objecto exterior comporta-se como um cliente do objecto interior. Neste caso o objecto exterior contém o objecto interior e utiliza os seus serviços para se implementar a si próprio. Não é obrigatório que ambos os objectos possuam as mesmas interfaces, pode mesmo acontecer que o objecto exterior utilize uma interface do objecto interior para facilitar a implementação de partes de outra interface especialmente se a complexidade de ambas for muito diferente.

MECANISMO DE AGREGAÇÃO

Neste mecanismo (Figura 40) o objecto exterior expõe de forma transparente as interfaces do objecto interior como se tivessem sido implementadas nele próprio. Este mecanismo é útil quando as invocações às interfaces do objecto exterior são todas destinadas às interfaces correspondentes do objecto interior. Desta forma, evita-se o *overhead* que existiria em implementações extra no objecto exterior.

Um pormenor importante relativamente a estes dois mecanismos é a forma como o objecto exterior se apresenta aos clientes. Sob o ponto de vista dos clientes, ambos os objectos implementam as interfaces A, B e C e tratam-nas como caixas negras porque não necessitam de conhecer a sua implementação interna mas apenas o seu comportamento.

No mecanismo de contenção, o objecto exterior durante a sua criação, cria todos os objectos interiores de que precisa, tal como um cliente faria. É dos dois mecanismos o mais utilizado [MSFT - 1995].

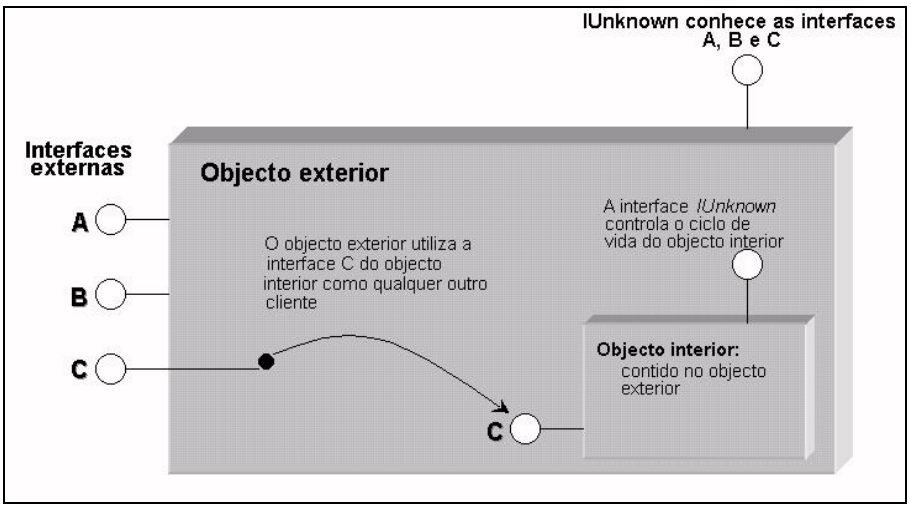

Figura 39 - Contenção de um objecto interior e delegação das suas interfaces.

No mecanismo de agregação, a principal diferença relativamente ao mecanismo de contenção reside na implementação das três funções da interface *IUnknown*: *QueryInterface*, *AddRef* e *Release* (Figura 38). Sob o ponto de vista do cliente, qualquer função *IUnknown* no objecto exterior tem de afectar este objecto, isto é, *AddRef* e *Release* afectam o objecto exterior e a *QueryInterface* expõe todas as suas interfaces disponíveis. Contudo se o objecto exterior expuser apenas as interfaces do objecto interior as invocações aos membros *IUnknown* do objecto interior através dessa interface irão ter um comportamento diferente das mesmas invocações pelo objecto exterior.

A solução é o objecto exterior passar um ponteiro *IUnknown* para o objecto interior para este lhe redireccionar as invocações *IUnknown* efectuadas nas suas próprias interfaces.

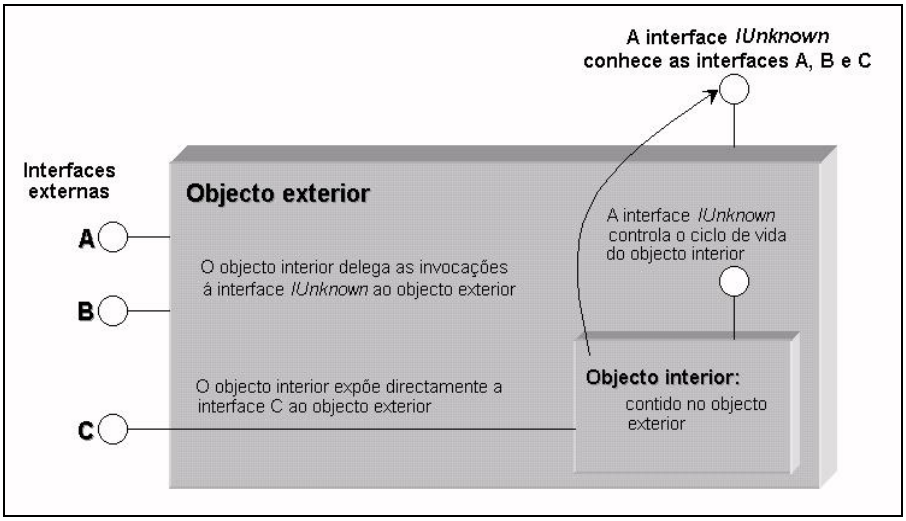

Figura 40 - Agregação de um objecto interior, onde o objecto exterior expõe uma ou mais interfaces do objecto interior como se fossem suas.

ENCAPSULAMENTO DO CICLO DE VIDA DOS OBJECTOS

Nos sistemas tradicionais baseados em objectos a gestão do ciclo de vida é determinada explicitamente pelo programador ou implicitamente pela linguagem de programação. Por outras palavras, há sempre alguém ou algo que tem o conhecimento de quando os objectos devem ser criados ou quando devem ser destruídos.

Num sistema distribuído a criação e destruição de objectos não se processa da mesma forma. Um objecto é criado quando o cliente requer a sua criação mas a sua destruição é mais complexa porque mais do que um cliente pode estar a utilizar o objecto. Assim, os clientes devem notificar os objectos quando os estiverem a utilizar e quando não necessitarem mais deles devem notificá-los também para que possam ser destruídos. A este mecanismo de notificação dá-se a designação de Contagem de Referências.

SEGURANÇA

O DCOM fornece segurança às aplicações independentemente da sua execução local ou remota, isto é, fornece meios seguros de acesso aos objectos e aos dados por estes encapsulados mantendo a integridade do sistema. Esta segurança é fornecida a vários níveis: utiliza as permissões do sistema operativo para determinar se um cliente (que corre num dado contexto de segurança atribuído a um utilizador) pode aceder a um objecto; utiliza as permissões da aplicação ou do sistema operativo para determinar se um determinado cliente pode ou não carregar o objecto. Em caso afirmativo, determina se o acesso é apenas de leitura, leitura e escrita, ou outro, o que significa que as aplicações podem controlar quem acede aos seus serviços e o tipo de acesso a esses mesmos serviços.

Tendo em conta que a arquitectura de segurança do DCOM é baseada na arquitectura de segurança do DCE RPC, significa que as características de autenticação de serviços disponibilizados por esta arquitectura podem ser utilizados pelo DCOM [MSFT - 1995].

Outro aspecto a ter em consideração é que quando vários utilizadores requerem serviços a um único servidor não seguro é executada uma

instância separadamente para cada utilizador. Cada ligação cliente/servidor é independente das outras evitando desta forma que um cliente possa aceder aos dados de outro(s). Todos os servidores não seguros são executados segundo o princípio de segurança de quem os invocou. O exemplo da Figura 41 envolve 4 clientes que acedem ao mesmo servidor. Como dois dos clientes (cliente 2 e cliente 3) correspondem ao mesmo utilizador (utilizador 2) é criada apenas uma instância do servidor para ambos os clientes.

A tecnologia utilizada no DCOM para distribuição implementa este sistema de segurança com os serviços de autenticação fornecidos pelo mecanismo RPC como já referenciado. Estes serviços são acedidos pelas aplicações através da biblioteca COM ao invocar a função *CoInitialize*. Este sistema de segurança impõe a restrição relativamente ao local onde podem correr as aplicações não seguras. Dado o sistema não poder iniciar uma sessão noutro computador sem as credenciais apropriadas, todos os servidores que correm no contexto de segurança do cliente são normalmente executados onde é executado o cliente.

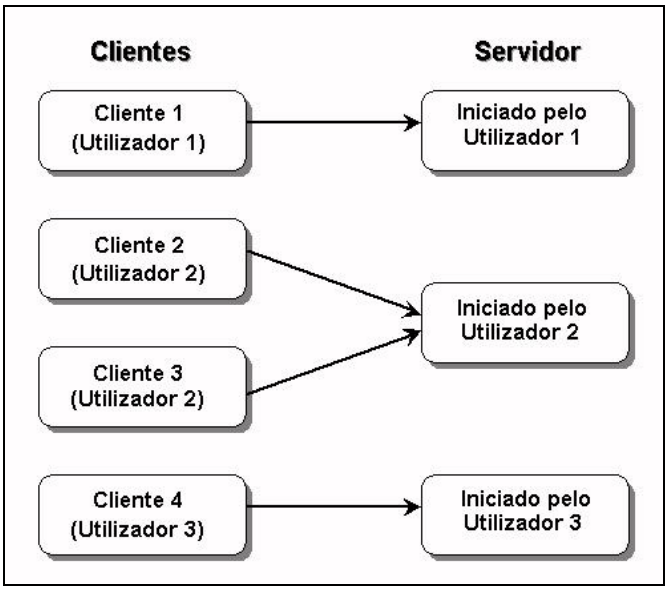

Figura 41 - Servidor não seguro.

Servidores seguros são aquelas aplicações que não permitem acesso global aos seus serviços. Podem correr na mesma localização do cliente, onde estão armazenados os dados, ou noutra localização dependendo de um conjunto de regras de activação.

4.3. O MODELO DCOM CLIENTE/SERVIDOR

O DCOM suporta um modelo de interacção cliente/servidor entre um utilizador de serviços de um objecto, o cliente, e a entidade que implementa o objecto assim como os seus serviços, o servidor. O cliente não é obrigatoriamente uma aplicação, no entanto obtém de alguma forma um ponteiro para aceder aos serviços do objecto e invoca-os quando necessário. De forma semelhante o servidor implementa o objecto e outras

estruturas de modo a que a biblioteca COM relacione essa implementação com um identificador de uma classe (CLSID - *Class Identifier*).

A biblioteca COM utiliza os CLSIDs para fornecer serviços de localização de implementações aos clientes. O cliente necessita apenas informar o DCOM do CLSID e tipo de servidor pretendidos. Seguidamente o DCOM localiza a implementação dessa classe e estabelece uma ligação entre ela e o cliente (Figura 42).

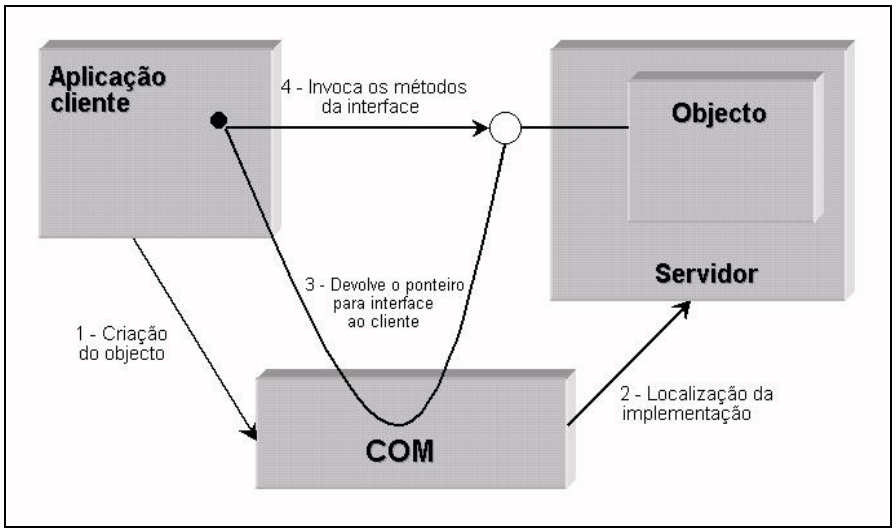

Figura 42 - Os clientes localizam e acedem aos objectos através dos serviços de localização de implementações do DCOM. Seguidamente o DCOM liga o cliente ao objecto no servidor.

Uma classe DCOM consiste numa implementação de um dado número de interfaces. A forma como o DCOM foi desenvolvido, permite que as classes sejam utilizadas por diferentes aplicações, incluindo aquelas que desconhecem a sua existência. Desta forma o código das classes pode estar contido em DLLs ou numa aplicação (EXE). O DCOM especifica um mecanismo que permite que o código das classes possa ser utilizado por várias aplicações diferentes [MSFT - 1995].

Um objecto DCOM é identificado por um CLSID único de 128 bits, que associa uma classe de objectos com uma DLL ou aplicação (EXE) no sistema de ficheiros. Um CLSID é semelhante ao GUID e por isso não existem CLSIDs iguais, independentemente do vendedor. Na implementação dos servidores, os CLSIDs são obtidos a partir da função DCOM *CoCreateGUID* ou a partir de uma ferramenta que invoque esta função internamente.

A utilização de CLSIDs únicos impossibilita a colisão de nomes entre classes uma vez que os CLSIDs não se encontram ligados aos nomes utilizados nos níveis inferiores da implementação. Por exemplo, diferentes vendedores podem escrever classes diferentes às quais dão o mesmo nome, `xpto`, no entanto cada uma delas terá um CLSID diferente o que impossibilitará todas as possibilidades de colisão, isto é, os CLSIDs não se relacionam com os nomes das classes.

No sistema hospedeiro o DCOM mantém uma base de dados denominada *system registry* que contém o registo de todos os CLSIDs correspondentes aos servidores instalados no sistema, isto é, existe um registo entre cada CLSID e a localização da DLL ou EXE. O DCOM consulta esta base de dados sempre que um cliente quer criar uma instância de uma classe DCOM e utilizar os seus serviços. O cliente necessita unicamente conhecer o CLSID,

o que o mantém independente da localização especifica da DLL ou EXE no sistema. Se o CLSID solicitado não for encontrado na base de dados local, são utilizados algoritmos para a localização da implementação na rede à qual o sistema se encontra ligado. A entidade responsável pela localização é o serviço SCM (*Service Control Manager*).

4.3.1. CLIENTES DCOM

Qualquer aplicação que passe ao DCOM um CLSID e lhe solicite para devolver uma instância de um objecto é um cliente DCOM. Independentemente do tipo de servidor que vá utilizar, o cliente solicita ao DCOM para instanciar os objectos sempre da mesma maneira, sendo a forma mais simples de criar um objecto a invocação da função DCOM *CoCreateInstance*. Esta função cria um objecto com o CLSID dado e devolve um ponteiro para a interface do tipo solicitado pelo cliente. Alternativamente o cliente pode obter um ponteiro para a interface invocando a função *CoGetClassObject*. Esta função é um construtor de classes que suporta uma interface chamada *IClassFactory* a partir da qual um cliente pede para criar um objecto. Neste ponto, o cliente possui dois ponteiros para interfaces de dois objectos diferentes, um para o construtor de classes e outro para o objecto da classe (Figura 43).

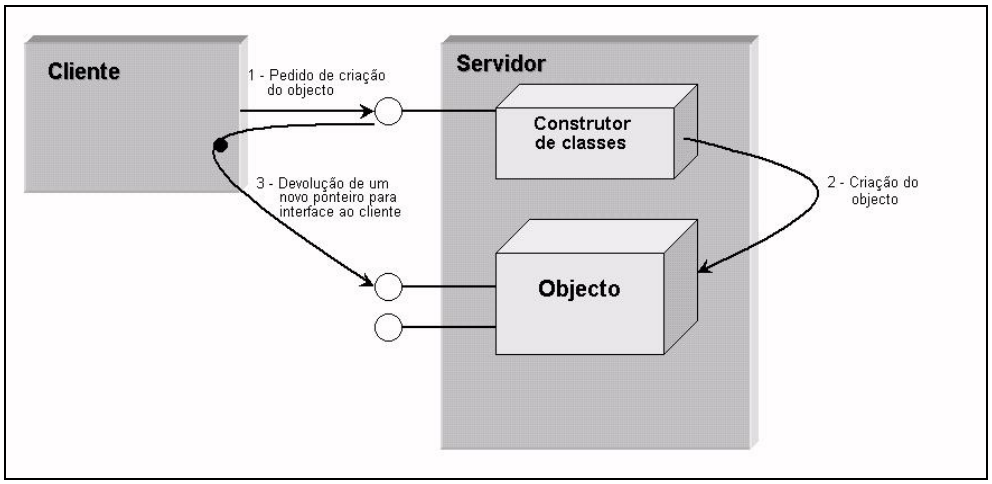

Figura 43 - O cliente DCOM cria os objectos a partir de um construtor de classes.

Resumindo, o cliente DCOM além de ser uma aplicação DCOM é responsável por utilizar o DCOM para obter um construtor de classes, por lhe solicitar para criar o objecto, por inicializar esse objecto e invocar a função *Release* do objecto e do construtor quando já não necessitar deles [MSFT - 1995].

4.3.2. SERVIDORES DCOM

Existem basicamente dois tipos de objectos servidores:

- em DLLs (Servidores *in-process*): o servidor é implementado num módulo que pode ser carregado e executado no espaço de endereçamento do cliente. O termo DLL é aqui utilizado para referir o mecanismo de partilha de uma biblioteca existente numa plataforma DCOM.
- em EXEs (Servidores *out-process*): o servidor é implementado como um módulo executável independente, isto é, o código do servidor é executado noutro processo no mesmo computador ou num computador remoto.

  Neste último caso temos ainda a distinção entre servidor local e remoto. Quando o cliente e o servidor são executados no mesmo computador diz-se servidor local, quando o cliente e o servidor são executados em computadores diferentes diz-se servidor remoto.

  Um servidor local corre num processo diferente do do cliente e serve objectos locais. Este tipo de servidor consiste numa aplicação (módulo executável - EXE) que corre no seu próprio processo, ao contrário das DLLs que têm que ser carregadas em processos existentes.

  Um servidor remoto corre num computador remoto e como tal num processo separado servindo objectos remotos. Este tipo de servidores podem ser implementados quer como DLLs quer como EXEs; se o servidor remoto for implementado numa DLL, é criada uma imagem do processo no computador remoto.

A Figura 44 ilustra a estrutura geral de um servidor DCOM (DLL ou EXE).

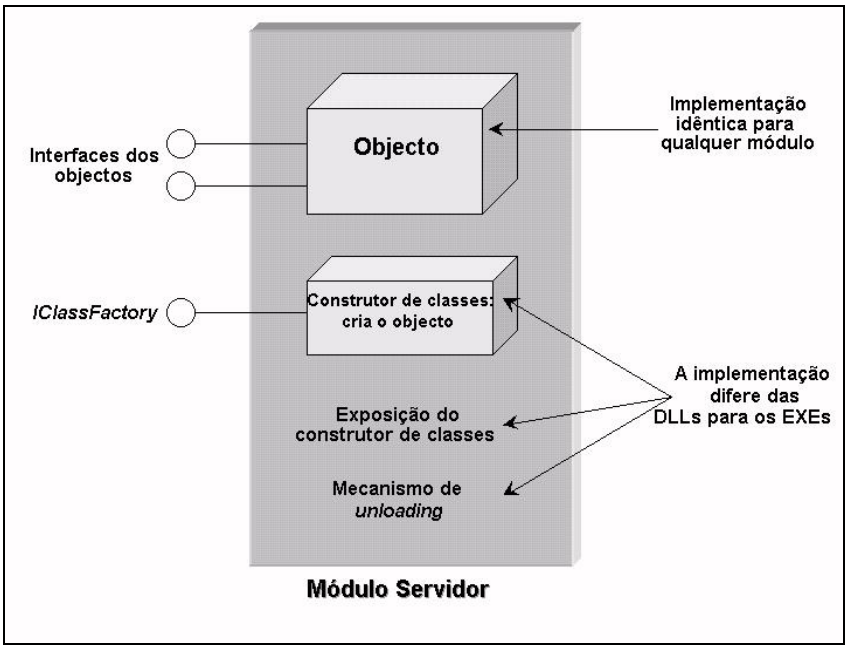

Figura 44 - Estrutura geral de um servidor DCOM.

Além dos servidores apresentados existe ainda um tipo especial de servidor chamado *Custom Object Handler* que funciona em conjunto com um servidor local fornecendo uma implementação parcial *in-process* de uma classe de objectos[18]. Estes servidores têm como principais características ajudar a melhorar o desempenho das operações gerais sobre objectos bem como a qualidade das mesmas.

4.3.3. BIBLIOTECA COM/DCOM

Além de ser uma especificação (ponto 4.1), o DCOM é também uma implementação disponibilizada através de uma biblioteca (Biblioteca COM) que no Microsoft Windows se apresenta como uma DLL, a qual inclui:

- Um conjunto de funções API para a criação de aplicações DCOM cliente e servidor. Aos clientes o DCOM fornece funções de criação básicas de objectos e aos servidores a facilidade de exposição dos seus objectos.
- Serviços de localização de implementações através dos quais o DCOM, a partir de um identificador da classe (CLSID), determina que servidor implementa essa classe e onde esse servidor se encontra localizado.
- Chamadas transparentes a objectos via RPC independentemente de estes se encontrarem a correr em servidores locais ou remotos.
- Um mecanismo normalizado que permite às aplicações controlar como a memória é reservada relativamente aos seus processos.

Desta forma e como conclusão, será necessário apenas uma empresa fazer a implementação da biblioteca COM para cada um dos sistemas operativos disponíveis. Alguns exemplos de sistemas operativos para os quais o DCOM já se encontra disponível são: Windows 95, 98 e NT, Apple Macintosh, Solaris, AIX, MVS e Linux [DCOM - 1996][Meta - 1998].

4.3.4. ARQUITECTURA PARA OBJECTOS DISTRIBUÍDOS

Quando um cliente pretende ligar-se a um objecto servidor, o nome desse servidor deve estar armazenado na *system registry.* No caso de objectos distribuídos o servidor pode ser implementado de três maneiras diferentes como referenciado (ver ponto 4.3.2), sendo o componente SCM o responsável pela sua localização e execução (ver ponto 4.3.5).

Depois de localizado o servidor, a invocação de um método de uma interface vai requer a cooperação de vários componentes, nomeadamente:

- a interface *proxy*, que é a parte específica do código das interfaces que reside no espaço de endereçamento do cliente e prepara os parâmetros da interface para a transmissão, efectuando o seu empacotamento ou *marshalling* de modo a que os pacotes possam ser reconstruídos e entendidos por parte do processo receptor;
- a interface *stub,* também ela uma parte específica do código das interfaces, reside no servidor e efectua o trabalho inverso da *proxy.* A *stub* recebe e descodifica os dados que foram transmitidos e

[18] Um *handler* não é mais que a representação de um objecto remoto que reside no processo cliente o qual internamente contém a ligação remota.

redirecciona-os para o servidor. No entanto também codifica e envia informação para responder ao cliente;

- a transmissão dos dados pela rede é efectuada recorrendo ao mecanismo RPC.

Este processo, o fluxo da comunicação, é ilustrado na Figura 45. Do lado do cliente a invocação de um método passa pelo *proxy* e seguidamente pelo canal. O canal envia o *buffer* com os parâmetros codificados para a biblioteca RPC que o transmitirá. As bibliotecas RPC e COM existem em ambos os lados do processo.

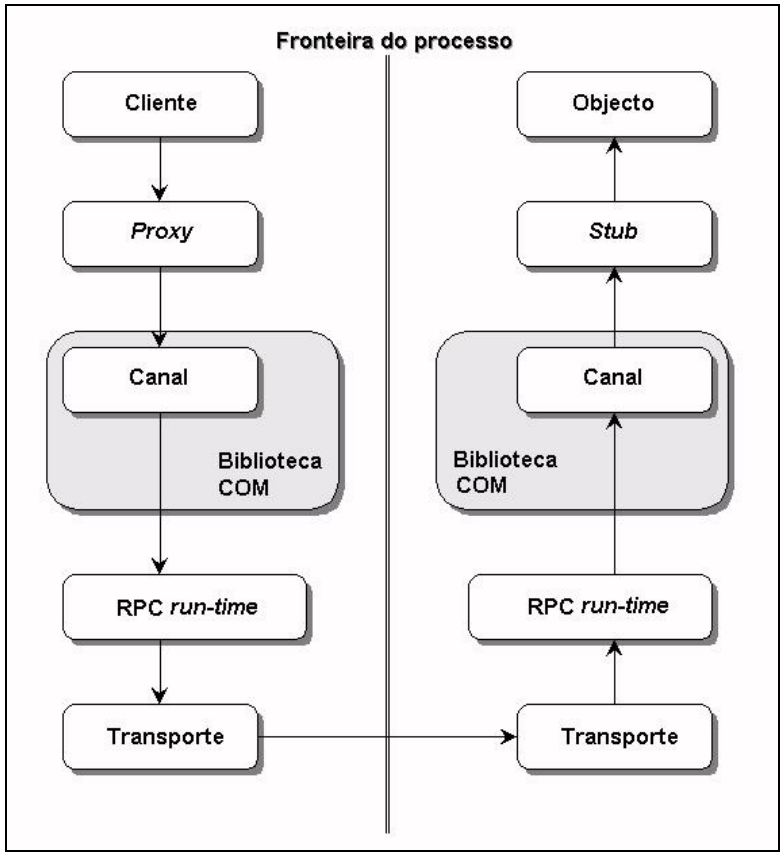

Figura 45 - Componentes da arquitectura distribuída DCOM.

4.3.5. SERVIÇO DE GESTÃO DE CONTROLO

O SCM (*Service Control Manager*) assegura que quando um cliente faz um pedido o servidor apropriado é conectado e tornado apto a receber esse pedido. O SCM mantém uma base de dados de informação sobre as classes com base na *system registry*, que o cliente guarda localmente recorrendo à biblioteca COM [MSFT - 1995].

Quando um cliente faz um pedido para criar um objecto de um dado CLSID, a biblioteca COM contacta o SCM local (no mesmo computador) e solicita a localização e execução do servidor apropriado sendo então devolvido um construtor de classes à biblioteca COM que o cliente pode utilizar para criar o objecto (Figura 46).

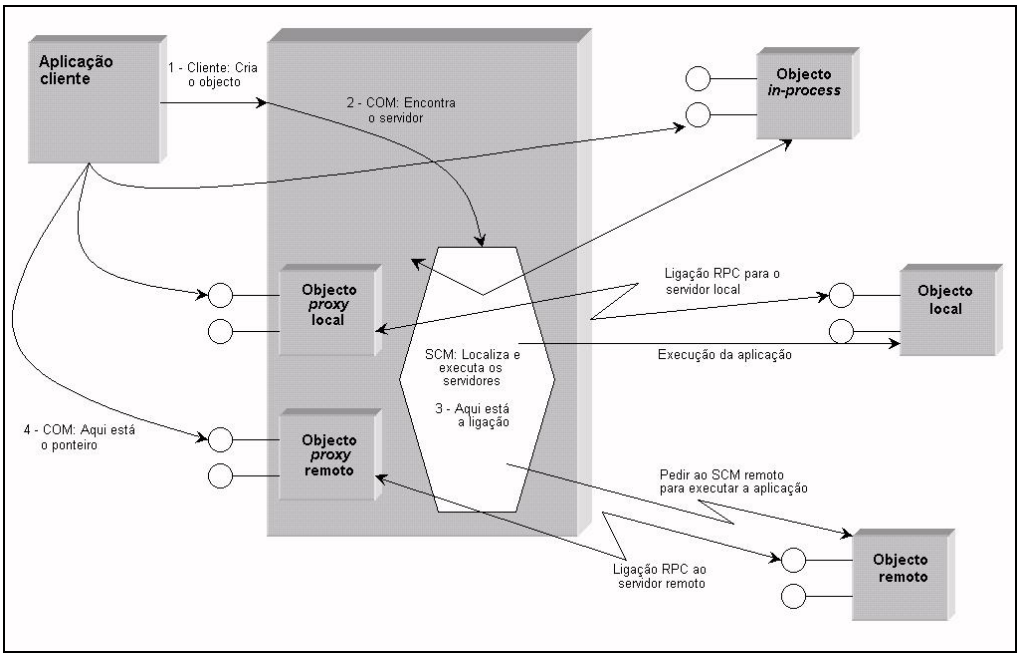

Figura 46 - O DCOM delega a responsabilidade de carregamento e execução dos servidores no SCM.

As acções do SCM dependem do tipo de servidor que está registado para o CLSID em causa nomeadamente [MSFT - 1995]:

In-Process: O SCM devolve o caminho do ficheiro DLL que contém a implementação do objecto servidor. A biblioteca COM faz seguidamente o carregamento da DLL e solicita-lhe o seu ponteiro para a interface do construtor de classes.

Local: O SCM inicia no arranque o módulo executável local que regista um construtor de classes disponibilizando assim o ponteiro ao DCOM.

Remoto: O SCM local contacta o SCM que está a ser executado no computador remoto redireccionando-lhe o pedido. O SCM remoto executa o servidor que regista um construtor de classes como o do servidor local na máquina remota. O SCM remoto mantém então uma ligação com esse construtor de classes e devolve uma ligação RPC ao SCM local. O SCM local devolve essa ligação ao DCOM que cria um *proxy* construtor de classes que internamente redirecciona os pedidos para o SCM remoto (consequentemente para o servidor remoto) via ligação RPC.

Se o SCM remoto detectar que o servidor remoto é um servidor *in-process*, carrega um servidor intermédio (*surrogate server*), que por sua vez carrega o servidor *in-process.* A função do servidor intermédio é a de passar todos os pedidos para a DLL que foi carregada.

4.4. OBJECTOS CONECTÁVEIS E EVENTOS

Sob o ponto de vista dos objectos as suas interfaces são *incoming* o que significa que as funções recebem dados do exterior. Além deste tipo de interfaces o DCOM define também interfaces *outgoing* para permitir que os objectos possam comunicar nos dois sentidos com o cliente. Quando um objecto suporta uma ou mais interfaces deste tipo diz-se que é conectável. Uma das utilizações mais frequentes das interfaces *outgoing* é para a notificação de eventos. Objectos deste tipo, também denominados fonte (*source*), podem ter o número de interfaces *outgoing* que desejarem, sendo cada interface composta por funções distintas em que cada uma representa um único evento, notificação ou pedido.

O mecanismo através do qual o DCOM efectua a comunicação entre o cliente e o servidor nos dois sentidos baseia-se em pontos de conexão (Figura 47).

Em qualquer dos casos deve haver sempre um cliente que escute o objecto e utilize essa informação. É o cliente que implementa estas interfaces em objectos chamados *sinks*. Sob o ponto de vista destes objectos as interfaces são *incoming* o que significa que os *sinks* estão à escuta através delas. Um objecto conectável possui o mesmo papel de um cliente no que diz respeito ao *sink*. Então, um *sink* é o que o cliente utiliza para escutar o objecto.

Um objecto não possui necessariamente uma relação de um para um com um *sink.* Na realidade uma instância de um objecto possui normalmente mais de uma ligação a *sinks* relativamente a um qualquer número de clientes. A isto chama-se *multicasting*.

Assim um objecto conectável deve ser capaz de:

- Possuir interfaces *outgoing*.
- Enumerar os IIDs (*Interface Identifier*) das interfaces *outgoing*.
- Ligar e desligar *sinks* do objecto para esses IIDs *outgoing*.
- Enumerar as ligações existentes para uma determinada interface *outgoing*.

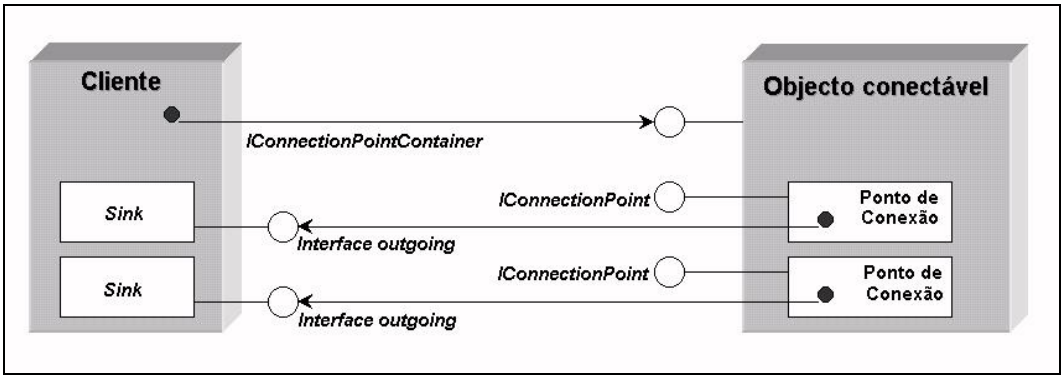

Figura 47 - Pontos de conexão.

4.5. ARMAZENAMENTO PERSISTENTE

Os serviços DCOM definem um conjunto de interfaces relacionadas com armazenamento e que foram designadas colectivamente como Armazenamento Persistente ou Armazenamento Estruturado. Atendendo à definição de interface, estas não transportam implementação, descrevem uma forma de criar um "sistema de ficheiros dentro de um ficheiro (*a file system within a file*)" e disponibilizam algumas características importantes para aplicações como acesso incremental, transacções e um meio partilhado que pode ser utilizado para troca de dados ou para armazenamento persistente de objectos.

4.5.1. SISTEMA DE FICHEIROS DENTRO DE UM FICHEIRO

Há alguns anos atrás antes de haver sistemas operativos as aplicações tinham que escrever os seus dados directamente no disco através do envio de comandos directamente para o controlador de hardware do disco. Estas aplicações eram responsáveis pela gestão da localização dos dados no disco e tinham que assegurar que os mesmos não eram escritos em locais onde já existissem dados. Esta situação não era problemática, uma vez que, uma única aplicação geria a totalidade do sistema, onde obviamente os discos estavam incluídos.

A partir do momento em que os sistemas computacionais passaram a poder executar mais do que uma aplicação, surgiram alguns problemas pois as aplicações teriam que assegurar que não escreviam sobre os dados já armazenados. Devido a este facto tornou-se necessário a adopção de uma norma que estabelecesse a marcação dos sectores de disco livres e ocupados. Estas normas deram origem aos sistemas operativos munidos de um sistema de ficheiros. Actualmente em vez de lidar directamente com os sectores do disco, as aplicações unicamente indicam ao sistema de ficheiros para escrever blocos de dados no disco. Além disso, o sistema de ficheiros permite às aplicações criar uma hierarquia de informação utilizando directorias que podem conter além de ficheiros outras directorias (sub-directorias) que por sua vez podem também conter mais ficheiros, mais directorias (sub-directorias), etc.

O sistema de ficheiros disponibiliza um único nível de indirecção entre as aplicações e o disco. Como resultado cada aplicação vê um ficheiro como uma *stream* de bytes única e contígua no disco. No entanto nos níveis inferiores, o sistema de ficheiros guardou o ficheiro, em disco, de forma descontígua e em diferentes sectores de acordo com algum algoritmo. O nível de indirecção disponibilizado pelo sistema de ficheiros permite que as aplicações não se preocupem com o posicionamento dos dados (armazenamento dos dados) nos dispositivos de armazenamento.

Actualmente as APIs de sistema de entrada e saída de ficheiros disponibilizam aplicações com capacidade para escrever informação em ficheiros *flat* de forma a que as aplicações vejam essa informação como uma única *stream* de bytes que pode crescer tanto quanto necessário até ser atingida em último caso a capacidade máxima do disco. Durante muito tempo estas APIs foram suficientes para o armazenamento das informações persistentes das aplicações. Com a evolução das aplicações foram introduzidas inovações relativamente à forma como estas lidam com uma *stream* de informação para a disponibilização de novas características.

A característica principal do DCOM é a interoperabilidade, a base para a integração de aplicações. Esta integração trouxe consigo a necessidade de ter várias aplicações a escrever informação no mesmo ficheiro. Este é

exactamente o mesmo problema que a indústria de computadores teve que resolver à alguns anos atrás quando diferentes aplicações tiveram que começar a partilhar o mesmo disco rígido. A solução encontrada então foi a criação de um sistema de ficheiros que possibilitasse um nível de indirecção entre o ficheiro e os sectores de disco.

A solução para o problema da integração passa uma vez mais pela inclusão de mais um nível de indirecção: um sistema de ficheiros dentro de um ficheiro. O DCOM define a entidade - sistema de ficheiros - como uma colecção estruturada de dois tipos de objectos - *storage* e *stream* - que agem como directorias e ficheiros respectivamente, não necessitando assim que uma grande sequência de bytes contígua no disco seja manipulada através de um único titulo de ficheiro com um único ponteiro.

4.5.2. OBJECTOS STORAGE E STREAM

De acordo com a definição DCOM para armazenamento persistente, existem dois tipos de elementos de armazenamento: objectos *storage* e objectos *stream*. Estes objectos são geralmente implementados pela própria biblioteca COM, só muito raramente é que têm que ser as aplicações a implementa-los[19]. Estes objectos como todos os outros no DCOM implementam interfaces: a *IStream* para objectos *stream* e a *IStorage* para objectos *storage*.

Um objecto *stream* é o equivalente a um ficheiro. Através da sua interface *IStream* pode ser solicitado ao objecto para ler, escrever, procurar e realizar outras operações nos seus dados. As *streams* têm um nome e podem conter qualquer estrutura interna uma vez que são simplesmente cadeias de octetos.

Um objecto *storage* é o equivalente a uma directoria. Cada objecto *storage* como as directorias pode conter qualquer número de objectos *sub-storage* (sub-directorias) como um qualquer número de objectos *streams* (ficheiros). Cada objecto *storage* tem os seus próprios direitos de acesso (permissões). A interface *IStorage* descreve as capacidades de um objecto *storage* como: enumeração dos seus elementos, mover, copiar, dar novo nome, criar, destruir, etc. Um objecto *storage* não pode armazenar dados das aplicações, apenas pode guardar os nomes dos elementos (*storages* e *streams*) em si contidos.

Quando os objectos *storage* e *stream* são implementados pelo DCOM num sistema, os processos podem partilhá-los. Esta é a característica chave que permite aos objectos que são executados *in-process* ou *out-process* ter acesso idêntico ao disco rígido. Tendo em conta que o DCOM é carregado em cada processo separadamente, precisa de utilizar alguns mecanismos de partilha de memória do sistema operativo para comunicação entre processos acerca dos elementos (*storage* e *stream*) abertos e os seus modos de acesso.

O armazenamento estruturado DCOM construído sobre os objectos *storage* e *stream* torna muito mais fácil o desenho de aplicações, que pela sua natureza produzem informação estruturada. Por exemplo, considere-se um programa agenda que permita ao utilizador registar entradas para qualquer dia, de qualquer mês, de qualquer ano. As entradas são realizadas na forma de um objecto de um determinado tipo que gere alguma informação, isto é, os utilizadores que desejarem escrever texto na agenda devem guardar um

[19] A especificação recomenda que a implementação COM das diferentes plataformas (Windows, Macintosh, etc.) inclua uma implementação de armazenamento normalizada para ser utilizada por todas as aplicações.

objecto texto; se desejarem guardar a digitalização de uma fotografia devem utilizar objectos *bitmap* e assim sucessivamente.

Sem um meio eficaz para a estruturação da informação, a aplicação agenda poderia ser forçada a gerir uma estrutura extremamente pesada (Figura 48).

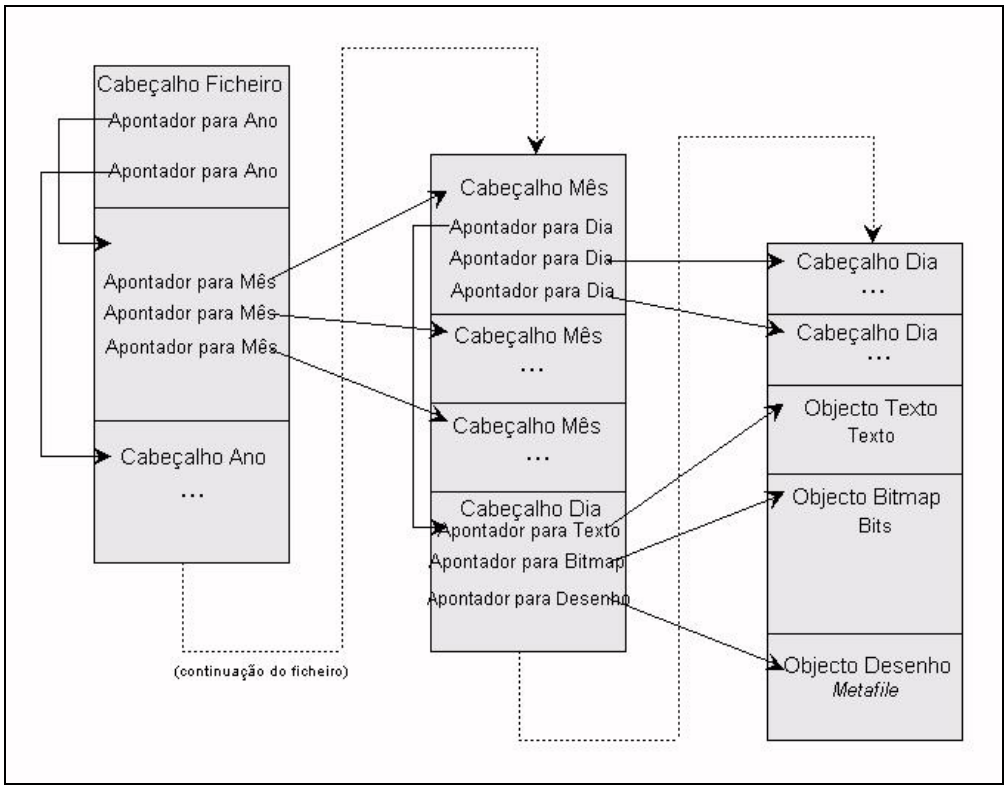

Figura 48 - Estrutura de um ficheiro *flat* para a aplicação agenda. Este tipo de estrutura é de difícil gestão.

A tecnologia de armazenamento persistente do DCOM resolve os problemas associados a este tipo de ficheiros através da introdução de um nível extra de indirecção. Com a tecnologia DCOM a aplicação agenda pode criar uma hierarquia estruturada onde o ficheiro raiz tem sub-ficheiros (*sub-storages*) para cada um dos anos. Cada *sub-storage* ano tem um *sub-storage* para cada mês e cada mês tem um *sub-storage* para cada dia. Cada dia deverá ainda ter outro *sub-storage* ou talvez uma *stream* para cada pedaço de informação que o utilizador pretenda armazenar nesse dia (texto, bitmap e outros) (Figura 49).

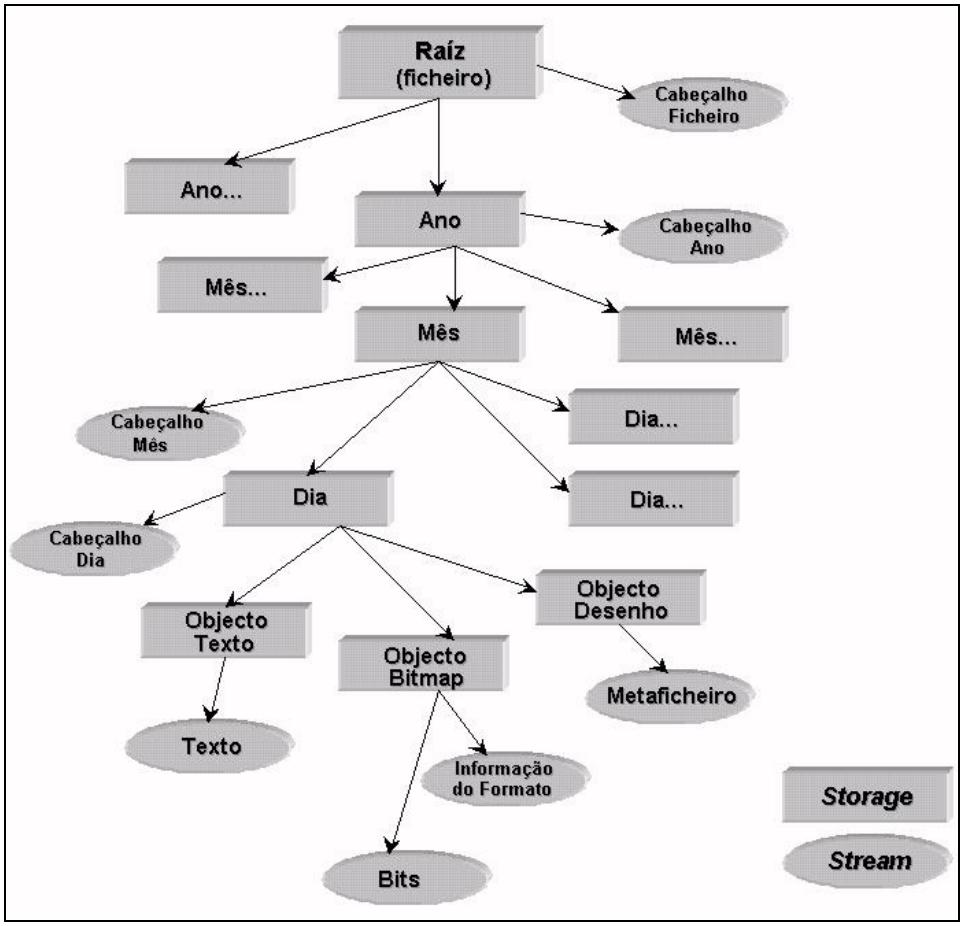

Figura 49 - Esquema de armazenamento estruturado para uma aplicação agenda. A cada objecto com conteúdo é dado o seu próprio armazenamento ou elemento *stream* para sua exclusiva utilização.

Esta estrutura resolve o problema da expansão da informação: o próprio objecto expande a *stream* que está sob o seu controle e a implementação DCOM do armazenamento calcula onde armazenar toda a informação na *stream*. Desta forma a aplicação agenda não tem que fazer nada. Além disso a implementação DCOM também gere automaticamente a totalidade do espaço livre.

4.5.3. NOMES

Cada objecto *storage* e *stream* tem um nome que os identifica. Estes nomes são utilizados para informar as funções da interface *IStorage* acerca do elemento a abrir, destruir, mover, copiar, dar novo nome, etc..

Os nomes dos objectos raiz (Figura 49) correspondem aos nomes dos ficheiros no sistema de ficheiros e como tal têm que obedecer às convenções e restrições por este impostas. Os nomes dos elementos contidos nos objectos de armazenamento são geridos pela implementação do próprio objecto.

4.5.4. ACESSO DIRECTO VS ACESSO TRANSACCIONAL

Os elementos *Storage* e *Stream* suportam dois modos diferentes de acesso: modo de acesso directo e modo de acesso transaccional. A diferença fundamental entre estes dois modos é que alterações efectuadas em modo directo são imediatamente e de forma permanente realizadas no objecto em causa, enquanto que em modo transaccional as alterações são armazenadas em *buffer* de forma a que só é possível o seu armazenamento (fazer o *commit*) ou voltar ao estado anterior, quando as modificações estiverem completas.

4.5.5. PESQUISA DE ELEMENTOS

O armazenamento estruturado do DCOM separa as aplicações da representação da informação nos ficheiros. Cada elemento de informação no ficheiro é acedido utilizando funções e interfaces implementadas pelo DCOM.

4.5.6. OBJECTOS PERSISTENTES

Uma vez que o DCOM permite aos objectos ler e escrever no seu local de armazenamento, tem que haver uma forma de os clientes comunicarem aos objectos para fazerem isso. A forma é recorrer a interfaces adicionais que estabelecem um contrato de armazenamento entre o cliente e os objectos. Quando o cliente deseja comunicar ao objecto para este guardar informação, o cliente interroga o objecto solicitando uma das interfaces de persistência que se adapte ao contexto. As interfaces que os objectos podem implementar e combinar entre si são as seguintes:

IPersistStorage: Recorrendo a esta interface o objecto pode ler e escrever o seu estado persistente num objecto *storage*. Esta interface inclui acesso incremental.

IPersistStream: Recorrendo a esta interface o objecto pode ler e escrever o seu estado persistente num objecto *stream*.

IPersistFile: O objecto pode ler e escrever o seu estado persistente num ficheiro directamente no sistema.

4.6. NOMES INTELIGENTES E PERSISTENTES: MONIKERS

Considere-se um nome de ficheiro tal como são vulgarmente conhecidos. Este nome, refere-se a um conjunto de dados que se encontra armazenado algures no disco rígido. O nome do ficheiro descreve a localização. A capacidade de identificar o que significa este nome e como pode ser utilizado, bem como pode ser guardado de forma persistente se necessário, está contido em todas as aplicações cliente desse nome de ficheiro. O nome do ficheiro não é mais do que uma parte dos dados nesse cliente. Isto significa que o cliente deve possuir código específico para manipular nomes de ficheiros.

Introduza-se agora um tipo de nome que descreva uma interrogação numa base de dados. Introduzam-se outros que descrevam um ficheiro e um

intervalo específico de dados nesse ficheiro, como por exemplo um conjunto de células numa folha de cálculo ou um parágrafo num documento. Por outras palavras, os clientes têm de saber o que um nome significa de forma a poderem utilizá-lo, o que significa que necessitam de código específico para cada tipo de nome o que por sua vez implica o crescimento monolítico da aplicação, quer em tamanho, quer em complexidade. Este problema é ultrapassado utilizando a tecnologia DCOM.

No DCOM a capacidade de conseguir trabalhar com um determinado nome está encapsulada no próprio nome, desta forma, o nome torna-se um objecto que implementa interfaces relacionadas com o nome. Estes objectos são apelidados de *monikers*[20]. A implementação dos *monikers* fornece uma abstracção para um mecanismo de ligação. Cada classe *moniker* (com um CLSID diferente) tem a sua própria semântica no que respeita a cada tipo de objecto ou operação. Enquanto que a própria classe *moniker* define as operações necessárias para localizar os objectos ou realizar algum tipo de acção, cada objecto *moniker* individual (cada instância) mantêm o seu próprio nome que identifica outro objecto ou operação. A classe *moniker* define a funcionalidade e o objecto *moniker* mantém os parâmetros.

Com os *monikers*, os clientes trabalham sempre com os nomes através de uma interface, em vez de manipularem eles próprios directamente as *strings* (ou qualquer outra coisa). Assim, quando um cliente desejar realizar uma operação sobre um nome, deverá chamar o código respectivo para o fazer em vez de o fazer ele próprio. Este nível de indirecção significa que o *moniker* pode disponibilizar de uma forma transparente um conjunto vasto de serviços e que o cliente pode interoperar com várias implementações de *monikers* que implementam estes serviços de diferentes maneiras.

4.6.1. OBJECTOS MONIKER

Um *moniker* é simplesmente um objecto que suporta a interface *IMoniker*. A interface *IMoniker* inclui a interface *IPersistStream*; deste modo os *monikers* podem ser guardados em *streams* e carregados de *streams*. A forma persistente de um *moniker* inclui os dados que compreendem o seu nome e o CLSID da sua implementação o qual é utilizado durante o processo de carregamento. Isto permite a criação transparente de novos tipos de *monikers* para os clientes.

A operação mais elementar da interface *IMoniker* é a operação de ligação ao objecto para o qual aponta [MSFT - 1995]. Os *monikers* também suportam uma operação chamada redução através da qual o *moniker* se reescreve ele próprio num outro *moniker* equivalente que ficará também ligado ao mesmo objecto.

4.6.2. TIPOS DE MONIKERS

Os objectos *moniker* podem ter vários tipos ou classes dependendo da informação que contêm e do tipo de objectos a que se podem dirigir. Uma classe *moniker* é definida pela informação que mantém de forma persistente e a operação de ligação que utiliza nessa informação.

[20] O nome *moniker* é sinónimo de pseudónimo (*nickname*).

O DCOM contudo especifica unicamente um *moniker* genérico (*generic composite moniker*). Este é um moniker especial por duas razões:

- os seus dados são compostos pelos dados persistentes de outros *monikers*, isto é, um *moniker* composto é uma colecção de outros *monikers*.
- a ligação de um *moniker* composto corresponde à ligação a cada *moniker* que contém na sequência respectiva.

Diz-se que o *moniker* composto é genérico porque não possui qualquer conhecimento das suas partes constituintes, sabe apenas que são também *monikers*. Um *moniker* composto pode inclusivamente ter nas suas partes constituintes outros *monikers* compostos.

O Microsoft OLE por exemplo, define quatro *monikers* específicos: *file*, *item*, *anti* e *pointer*, que utiliza especificamente para ajudar na implementação de objectos ligados na sua tecnologia de documentos compostos [MSFT - 1995].

Figura 50 - *Moniker* composto constituído por um *moniker* ficheiro e dois *monikers* item. Descreve a fonte de ligação, a qual é um intervalo de células numa folha específica de um ficheiro folha de cálculo.

4.7. TRANSFERÊNCIA DE DADOS UNIFORME

O DCOM além de disponibilizar interfaces para lidar com o armazenamento e nomes de objectos também disponibiliza interfaces para a troca de dados entre aplicações. Construída sobre o COM e sobre a tecnologia de armazenamento persistente a UDT (*Uniform Data Transfer*) tem a capacidade de representar todos os dados a transmitir através de um único objecto de dados. Os objectos de dados implementam uma interface chamada *IDataObject* a qual contém as operações normalizadas *get/set*, os formatos *query/enumerate* bem como funções através das quais o cliente de um objecto de dados pode estabelecer um *notification loop* para detectar alterações dos dados no objecto. Além disto esta tecnologia permite a utilização de qualquer meio de armazenamento como meio de transmissão, por exemplo memória e ficheiros.

4.7.1. SEPARAÇÃO DOS PROTOCOLOS DE TRANSFERÊNCIA

O termo uniforme na designação desta tecnologia resulta do facto da interface *IDataObject* separar todas as operações comuns de troca do que é chamado protocolo de transferência. Com a UDT os protocolos unicamente necessitam trocar um ponteiro com uma interface *IDataObject*. A fonte dos dados - o servidor, necessita apenas implementar um objecto de dados o qual é passível de ser utilizado em qualquer protocolo de transferência. O consumidor - o cliente, necessita unicamente de implementar uma parte do

código para solicitar os dados de um objecto de dados logo que receba um ponteiro *IDataObject* de um qualquer protocolo. Assim que a troca de ponteiro tenha ocorrido, ambos os lados (cliente e servidor) trocam entre si dados de forma uniforme através da *IDataObject*.

Esta uniformidade reduz o código e simplifica-o. Antes do COM ter sido implementado pela primeira vez no OLE 2, cada protocolo disponível no Microsoft Windows tinha o seu próprio conjunto de funções. Agora que a funcionalidade de troca está separada do protocolo, a negociação com cada protocolo requer unicamente uma quantidade mínima de código, a qual é absolutamente necessária para a semântica do protocolo em causa.

4.7.2. FORMATOS DE DADOS E MEIOS DE TRANSFERÊNCIA

Anteriormente à UDT todos os protocolos para transferência de dados usavam a memória como meio de transferência. Este problema revelava-se de uma importância extrema quando se pretendiam trocar grandes quantidades de informação. A menos que se possuísse uma máquina com uma grande quantidade de memória, a troca de um ficheiro com uma dimensão considerável (20Mb por exemplo) através da memória iria provocar um *swapping* considerável para a memória virtual no disco rígido.

O estado latente é um outro problema, em particular no caso das redes. Frequentemente é necessário começar o processamento de um conjunto de dados antes de todos os dados desse mesmo conjunto terem sido recebidos pelo sistema destinatário.

Para resolver estes problemas o DCOM define duas novas estruturas de dados: FORMATECT e STGMEDIUM.

A estrutura FORMATECT é a estrutura que o consumidor (cliente) utiliza para indicar o tipo de dados que deseja da fonte (objecto) e é utilizada pela fonte para descrever que formatos pode fornecer. A estrutura FORMATECT pode descrever quaisquer dados de forma virtual, incluindo outros objectos como *monikers* por exemplo.

A estrutura STGMEDIUM significa que as fontes de dados (*data sources*) e consumidores podem escolher utilizar o meio de troca mais eficiente.

4.7.3. SELECÇÃO DE DADOS

O DCOM suporta dois tipos de objectos de dados: os estáticos e os dinâmicos, utilizando para a troca de dados a interface *IDataObject*.

Os objectos de dados estáticos, como por exemplo os que representam o *clipboard* ou os que são utilizados numa operação arrastar e largar, correspondem a uma selecção específica e estática de dados na fonte, como por exemplo um intervalo de células numa folha de cálculo, uma porção de um *bitmap* ou uma determinada quantidade de texto. Durante o tempo de vida destes objectos de dados, os dados subjacentes mantém-se inalterados.

Os objectos de dados dinâmicos suportam a capacidade de, de uma forma dinâmica, alterarem o seu conjunto de dados. Esta capacidade contudo não é representada pela interface *IDataObject*. Isto significa que os objectos de dados que implementem esta característica têm que implementar outra interface para suportar a selecção de dados dinâmica. Exemplos destes objectos são os que suportam a especificação OLE para Dados de Mercado em Tempo Real (WOSA/XRT - *Real-Time Market Data*).

4.7.4. NOTIFICAÇÃO

Os consumidores de dados de uma fonte externa podem estar interessados em saber quando os dados na fonte são alterados. Para tal é necessário a existência de um mecanismo através do qual o próprio objecto de forma assíncrona notifique um cliente a si conectado que houve alteração nos dados.

O DCOM manipula as notificações deste tipo através de um objecto chamado *advise sink* que implementa uma interface chamada *IAdviseSink*. O objecto *advise sink* e a interface *IAdviseSink* são implementados pelo consumidor. Quando o objecto de dados detecta uma alteração, chama uma função na interface *IAdviseSink* para notificar o consumidor da referida alteração (Figura 51).

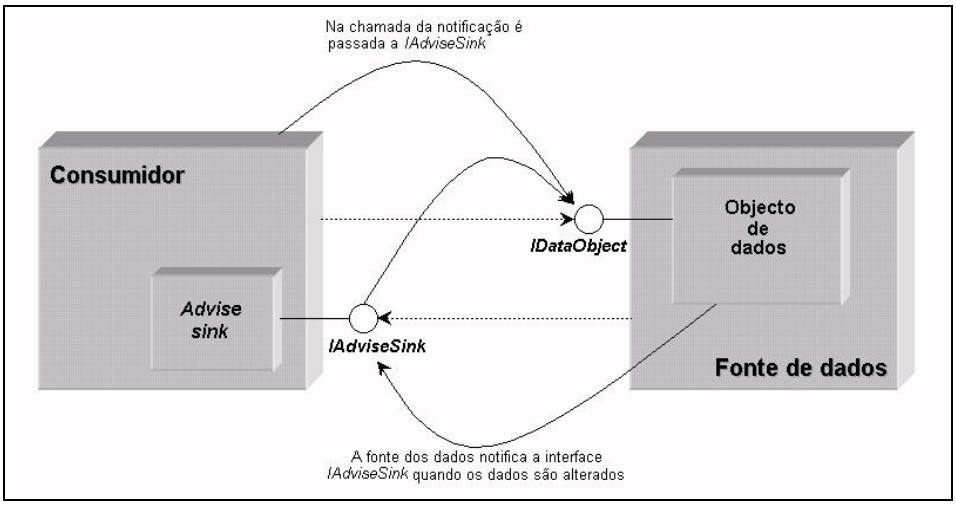

Figura 51 - Um consumidor implementa um objecto com a interface *IAdviseSink*. Através desta interface o objecto de dados notifica o consumidor das alterações nos dados.

4.8. EVOLUÇÃO DA ARQUITECTURA - COM+

Em Outubro de 1997 a Microsoft anunciava a sua estratégia futura no capítulo dos componentes divulgando o sucessor do COM/DCOM, o **COM+**, tendo esta arquitectura resultado da introdução de novas características no modelo de componentes que lhe deu origem.

Esta "nova" tecnologia será disponibilizada pela primeira vez no novo sistema operativo da Microsoft, o Windows NT 5.0 aliás Windows2000.

O que é exactamente o COM+?

O COM+ é a fusão dos Modelos de Programação COM e MTS [Armstrong - 1999](Figura 52).

Figura 52 - COM+ [Armstrong - 1999].

O COM+ surge como o resultado da uniformização das tecnologias COM, DCOM e MTS numa única tecnologia. Apresenta como características mais importantes as seguintes [Petersen - 1998]:

- Catálogo COM+.
- Carregamento Equilibrado.
- Amostragem de Objectos.
- Base de Dados em Memória.
- Novo Modelo de Eventos.
- Componentes em Fila.

É sobre estas características que os próximos parágrafos vão incidir. A descrição feita, é uma descrição de alto nível, não se pretendendo portanto detalhar aspectos relativos a implementação.

4.8.1. CATÁLOGO COM+

Actualmente os componentes COM e MTS colocam todas as suas informações relativamente à configuração no *windows registry* (ou *system registry*). Com o COM+ a maior parte desta informação será armazenada numa nova base de dados denominada Catálogo COM+. É objectivo deste catálogo fazer a uniformização dos modelos de registo de informação do COM e do MTS, assim como, a disponibilização de um ambiente de administração para componentes. A interacção com o Catálogo COM+ por parte do utilizador é feita recorrendo ao COM+ Explorer ou através de uma nova série de interfaces desenvolvidos para o efeito [Armstrong - 1999].

4.8.2. CARREGAMENTO EQUILIBRADO

Actualmente uma das deficiências do DCOM/MTS é não possuir suporte intrínseco do conceito Carregamento Equilibrado.

Para a criação de uma instância de um componente remoto, o cliente tem de especificar explicitamente o nome da máquina onde deseja que a instância seja criada. Esta nova característica vem possibilitar a definição de um *router* num servidor que terá por função remeter a criação dos objectos para a máquina que estiver a ser menos utilizada (Figura 53).

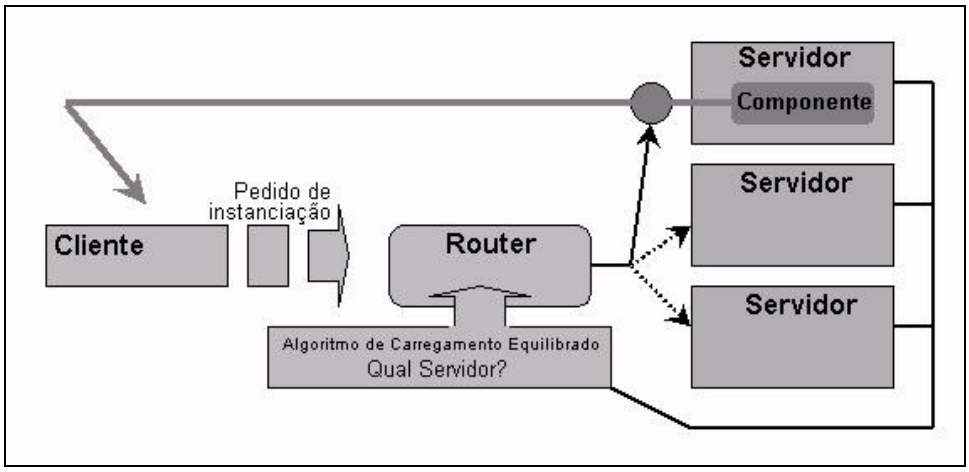

Figura 53 - Carregamento Equilibrado.

O processo utilizado pelo COM+ para determinar qual a máquina menos sobrecarregada está baseado num algoritmo que mede os tempos de resposta das chamadas a métodos nas diferentes máquinas onde o objecto pode ser criado [Armstrong - 1999].

4.8.3. AMOSTRAGEM DE OBJECTOS

A Junção de Objectos é o processo que consiste em manter um conjunto de instâncias de objectos carregadas em memória para que estas estejam imediatamente disponíveis para serem utilizadas por aplicações cliente. Esta nova característica revela-se de grande importância na construção de grandes aplicações.

4.8.4. BASE DE DADOS EM MEMÓRIA

Outro dos serviços introduzidos nesta nova versão foi a Base de Dados em Memória ou COM+ IMDB (*In-Memory DataBase*) (Figura 54).

Uma das formas mais eficazes de melhorar o desempenho das aplicações que trabalham intensivamente com dados é assegurar que tanto quanto possível os seus dados sejam carregados em memória. A COM+ IMDB é uma base de dados transitória e transaccional que trabalha unicamente na memória. O desenvolvimento desta base de dados e a sua integração no COM+ prendeu-se especificamente com factos relativos à Internet, nomeadamente ambientes Web onde existem milhares de utilizadores a aceder a informações em bases de dados. Contudo pode também ser

utilizada em qualquer aplicação que necessite acesso rápido a grandes quantidades de dados.

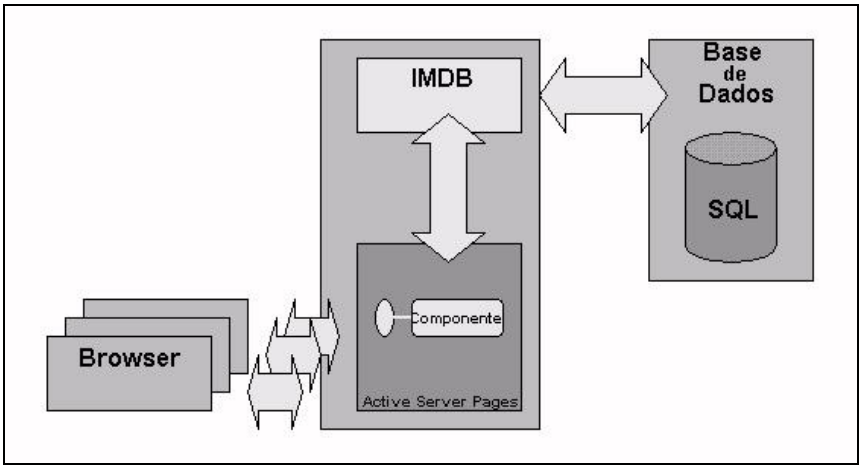

Figura 54 - IMDB COM+.

4.8.5. MODELO DE EVENTOS

O COM/DCOM disponibiliza a capacidade de manipulação de eventos ou *callbacks* entre o componente e os seus clientes.

No COM+, o termo editor é utilizado para indicar o módulo que divulga ou disponibiliza informação, e o termo subscritor para indicar o módulo que deseja receber essa informação [Moeller - 1998].

O Modelo de Eventos COM+ actualiza o anterior modelo de eventos disponibilizado pelo COM/DCOM. Este novo modelo continua a suportar as técnicas dos modelos anteriores, disponibilizando contudo um maior suporte intrínseco aos diferentes cenários divulgar-subscrever.

A grande inovação deste modelo é a introdução de um objecto intermediário chamado classe evento (*event class*), desta forma qualquer entidade que deseje divulgar informação deve fazê-lo através deste objecto (Figura 55).

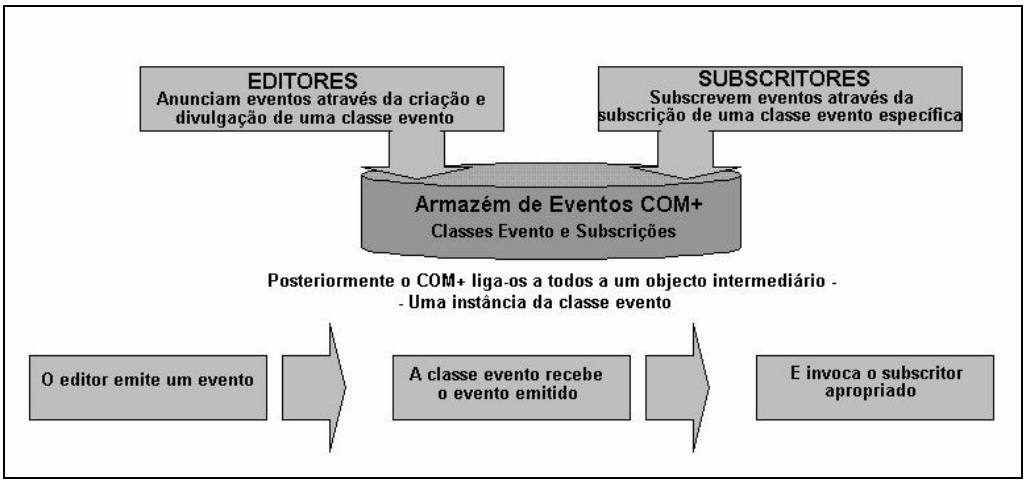

Figura 55 - Modelo de Eventos COM+.

Recorrendo a este objecto intermediário, o novo sistema de eventos consegue [Armstrong - 1999]:

- Disponibilizar cenários divulgar-subscrever que tornam independentes os tempos de vida das entidades envolvidas.
- Acrescentar um modelo de activação ao sistema de eventos, isto é, se um subscritor não está activo quando um determinado evento ocorre o sistema de eventos activa-o e passa-lhe a informação do evento.
- Disponibilizar um ambiente divulgar-subscrever em que a partir do momento que uma classe de eventos é criada, qualquer um se pode tornar um editor ou subscritor de eventos.
- Suportar mecanismos de filtragem. Através do desenvolvimento de objectos filtro pode fazer-se filtragem quer ao nível da edição, quer ao nível da subscrição.

Este sistema de eventos é um dos subsistemas do SENS (*System Event Notification*) do sistema operativo Windows2000.

4.8.6. COMPONENTES EM FILA

O Modelo COM/DCOM está baseado em interacções procedimentais. Um cliente liga-se ao componente, interroga-o sobre a interface adequada e faz chamadas a métodos de forma síncrona através da interface devolvida. O tempo de vida do cliente e da instância do componente estão ligados entre si, sendo a informação obtida do componente via parâmetros *out*.

Os serviços RPC tal como os disponibilizados pelo COM são necessários para a implementação de aplicações distribuídas. Contudo existem determinados tipos de aplicações que podem beneficiar de uma outra técnica de implementação denominada *messaging*. Numa aplicação deste tipo (baseada em mensagens) os tempos de vida do cliente e componentes podem ser diferentes (Figura 56).

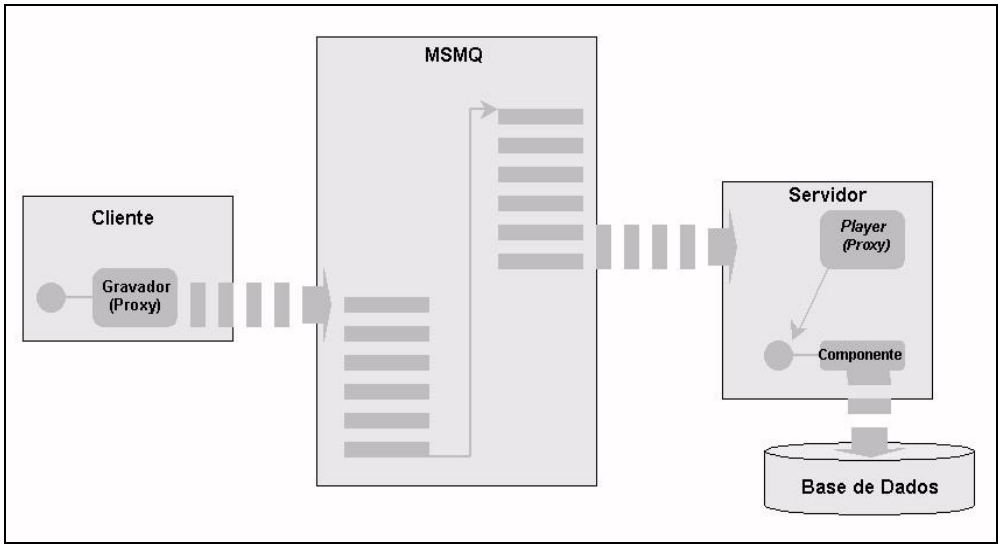

Figura 56 - Componentes em Fila.

Actualmente, quer os serviços RPC quer os serviços baseados em mensagens são necessários nas implementações de aplicações distribuídas. A grande vantagem do COM+ no que se refere a este aspecto, é permitir a possibilidade de escolha aos programadores da tecnologia que melhor se adapta às características das suas aplicações.

Referir que este serviço tem como suporte a tecnologia MSMQ (*Microsoft Message Queue Server*) da Microsoft.

CAPÍTULO 5

5. JAVARMI - JAVA REMOTE METHOD INVOCATION

5.1. INTRODUÇÃO

O desenvolvimento de aplicações distribuídas tem vindo a revelar-se extremamente difícil devido a factores como hardware, sistemas operativos, linguagens de programação e outros, que obrigavam os programadores ao ajustamento das aplicações ao sistema alvo.

Com a aceitação cada vez maior do ambiente de operação Java e da linguagem Java (*write once, run anywhere*) os programadores têm a possibilidade de desenvolver aplicações distribuídas sem terem que se preocupar com as plataformas que as vão suportar sendo apenas necessário que as ditas plataformas suportem JVM (*Java Virtual Machine*), sendo que, o número de plataformas que suportam JVM é cada vez maior e entre outras temos as plataformas Win32 e Solaris [Lindholm].

A JavaRMI (*Java Remote Method Invocation*) tira partido do ambiente de operação Java e da linguagem Java na construção de aplicações cliente/servidor, encontrando-se integrada no pacote JDK (*Java Development Kit*) a partir da versão 1.1. Ao contrário das duas arquitecturas apresentadas anteriormente, a arquitectura JavaRMI é dependente da linguagem que lhe dá origem, a linguagem Java.

5.2. ARQUITECTURA JAVARMI

No Modelo de Objectos Distribuídos Java um objecto remoto é um objecto cujos métodos podem ser invocados a partir de outra JVM a qual pode eventualmente estar localizada num computador diferente.

A invocação de métodos em objectos remotos tem exactamente a mesma sintaxe que a invocação de métodos em objectos locais. Tal como em outras arquitecturas, nomeadamente CORBA e DCOM, os clientes JavaRMI interagem com os objectos remotos via as suas interfaces, não interagindo nunca com as classes que implementam essas interfaces.

Para o suporte destas e doutras características a JavaRMI recorre a uma arquitectura composta por três níveis [Matthews] (Figura 57):

- 1º nível - *Stub/Skeleton*.
- 2º nível - Camada Referência Remota (RRL - *Remote Reference Layer*).
- 3º nível - Transporte.

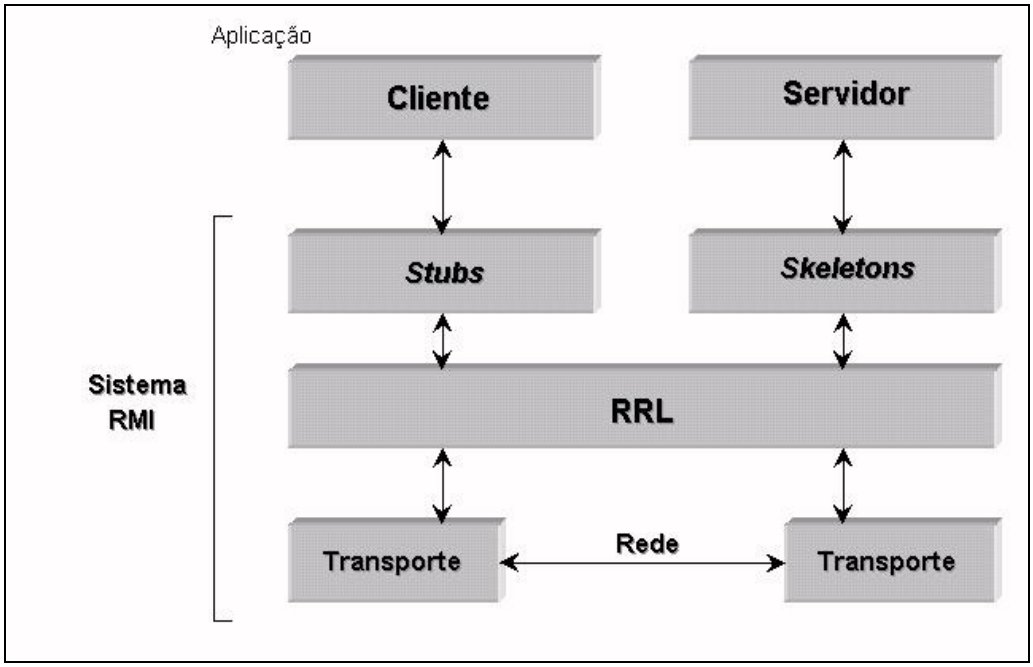

Figura 57 - Arquitectura JavaRMI.

O 1º nível - *Stub/Skeleton* - é responsável pela gestão das interfaces dos objectos remotos entre o cliente e o servidor.

O 2º nível - RRL - é responsável pela gestão da actividade dos objectos remotos. Tem também como função a gestão das comunicações entre os clientes e servidores com as JVMs para objectos remotos.

O 3º nível - Transporte - é responsável pela comunicação entre clientes e servidores. O protocolo utilizado a este nível é o RMP (*Remote Method Protocol*), protocolo proprietário da Sun. Relativamente a este aspecto a Javasoft/Sun e o OMG estão a trabalhar em conjunto na convergência dos dois modelos de objectos [Orfali - 1998] [Rocheleau - 1997]. Esta convergência está a ocorrer a dois níveis[21], sendo um dos pontos a JavaRMI utilizar o protocolo CORBA - IIOP como protocolo por defeito em substituição do RMP [Rocheleau - 1998].

Cada um dos níveis atrás referido é independente do seguinte podendo desta forma serem substituídos por implementações alternativas sem que tal afecte os outros níveis [Sun - 1999].

Como principais características desta arquitectura podem ser referenciadas as seguintes [Javasoft - 1998]:

[21] O segundo nível de convergência situa-se ao nível RMI/IDL [Orfali - 1998].

- Serialização de Objectos.
- Carregamento Dinâmico de *Stubs*.
- Serviço de Nomes.
- Colector de Objectos.

5.2.1. SERIALIZAÇÃO DE OBJECTOS

A transmissão transparente de objectos de um espaço de endereçamento (memória) para outro é conseguida através do Serviço Serialização de Objectos. Este serviço permite passar parâmetros a métodos remotos, como devolver valores de métodos remotos, através da colocação do estado de um objecto Java numa *stream* que por sua vez pode ser passada como um parâmetro no interior de uma mensagem. A técnica descrita anteriormente funciona quer para objectos locais quer para objectos remotos [Orfali - 1998]. Desta forma e tal como acontece com outras arquitecturas, a CORBA por exemplo, a passagem de objectos remotos é feita por referência e não através do recurso à cópia da implementação remota do objecto [Orfali - 1998].

5.2.2. CARREGAMENTO DINÂMICO DE STUBS

Um outro serviço da JavaRMI é o serviço designado por Carregamento Dinâmico de *Stubs*. Os clientes recorrem a este serviço quando o *stub* que o cliente necessita não se encontra na máquina local. Note-se que sem o *stub* apropriado para um determinado serviço, não é possível ao cliente invocar a interface remota. Como complemento a JavaRMI utiliza também os *stubs* para a devolução de argumentos que referenciam interfaces de objectos remotos. Em casos extremos, o código *stub* para uma implementação remota pode ser gerado no momento no servidor e depois ser enviado para o cliente [Orfali - 1998]. Contudo quando este serviço é utilizado é feito um *download* arbitrário de classes para clientes ou servidores o que pode proporcionar a abertura de potenciais falhas na segurança. Sem as devidas precauções esta característica pode ser potencialmente perigosa relativamente a vírus, devido a poder transformar-se numa incubadora de vírus[22].

5.2.3. SERVIÇO DE NOMES

Antes de se poder fazer a invocação a um método num objecto remoto, o cliente primeiro tem que obter uma referência para esse objecto. De uma forma geral esta referência é obtida através da devolução de um valor na chamada ao método. Além deste serviço a JavaRMI também disponibiliza um Serviço de Nomes que permite a obtenção de referências para objectos servidores através do nome destes.

5.2.4. COLECTOR DE OBJECTOS

Num sistema distribuído um Colector de Objectos (*garbage collector*) deve ter a capacidade de automaticamente remover objectos que não estejam a ser referenciados por nenhum cliente, não sendo necessário o programador

[22] Para evitar problemas a Javasoft recomenda que esta característica seja desactivada a não ser que os potenciais riscos que se correm sejam aceites.

preocupar-se com as conexões cliente que ainda se mantêm activas. Para tal a JavaRMI utiliza um esquema de contagem de referências. Assim todo o objecto que tenha pelo menos uma referência não pode ser removido pelo colector de objectos. A contagem de referências baseia-se num contador em que cada vez que um cliente obtém uma referência para um objecto o contador é incrementado de uma unidade e decrementado de uma unidade cada vez que é removida uma referência. Quando o contador - contagem de referências - atingir o valor 0 (zero), a JavaRMI coloca o objecto servidor numa lista denominada *weak reference* podendo o colector de objectos a partir desta altura remover os objectos que aí se encontram inscritos.

CAPÍTULO 6

6. COMPARAÇÃO ENTRE CORBA, DCOM E JAVARMI

6.1. INTRODUÇÃO

Depois de nos três capítulos anteriores se ter feito uma breve descrição de três das mais importantes arquitecturas distribuídas cliente/servidor - CORBA, DCOM e JavaRMI - é objectivo deste capítulo confrontá-las sobre vários aspectos.

Neste estudo comparativo pretende-se essencialmente fazer uma abordagem a um nível superior de funcionalidades e não a um nível interno. Um documento que descreve de forma brilhante o funcionamento interno tanto da arquitectura CORBA como da arquitectura DCOM é o documento [Chung] - "*DCOM and CORBA Side by Side, Step by Step, and Layer by Layer*" que faz parte das Referências Bibliográficas.

Este estudo comparativo incidiu sobre os três pontos seguintes:

- Interoperabilidade.
- Fiabilidade.
- Maturidade da Plataforma.

Com este estudo pretendem referenciar-se alguns dos aspectos que devem ser tidos em consideração quando se pretender fazer a implementação de arquitecturas distribuídas cliente/servidor nas organizações, em especial as arquitecturas CORBA, DCOM ou JavaRMI.

6.2. INTEROPERABILIDADE

Pretende-se neste ponto analisar as capacidades de cada uma das plataformas relativamente a questões como:

- Suporte de linguagens.
- Suporte de plataformas.
- Comunicação em rede.
- Serviços comuns.

6.2.1. SUPORTE DE LINGUAGENS

O suporte de diferentes linguagens é um factor de grande importância no que respeita à questão da interoperabilidade. Enquanto que linguagens como C++, Visual Basic e Java estão "na moda", o COBOL continua a ser a linguagem mais utilizada, estimando-se que cerca de 3 milhões de programadores utilizem esta linguagem, 1.6 milhões utilizem Visual Basic e 1.1 milhões utilizem C e C++ [ActCOR] pelo que o suporte da linguagem COBOL pelas diferentes plataformas deverá ser um factor a ter em consideração.

CORBA

A arquitectura CORBA foi desenhada para ser independente quer da linguagem, quer da plataforma em que assenta, recorrendo para tal a uma IDL comum. No contexto CORBA já existem mapeamentos IDL para linguagens como C, C++, Smalltalk, Ada, Java, OLE (Visual Basic, PowerBuilder, Delphi, etc.) e COBOL [OMG - 1995a].

DCOM

Na arquitectura DCOM a portabilidade no que respeita às linguagens está baseada numa norma de codificação binária. A forma de garantir esta portabilidade é controlar a forma como cada linguagem é traduzida para este código de 0s e 1s. Esta aproximação apresenta algumas desvantagens, como por exemplo o facto de existirem grandes diferenças na forma como as linguagens são traduzidas, isto é, umas serem compiladas e outras interpretadas. O facto de existirem vários compiladores/interpretadores para uma mesma linguagem, cada um tendo uma técnica diferente de tradução, aumenta também esta diversidade.

Referir que uma especificação de compatibilidade a um nível tão baixo do hardware cria vulnerabilidades devido inclusive ao progresso do próprio hardware [Meta - 1998].

Virtualmente qualquer linguagem pode ser utilizada para criar componentes DCOM sendo linguagens como Java/PowerBuilder, Pascal/Delphi e COBOL/MicroFocusCobol alguns exemplos.

JAVARMI

A JavaRMI como extensão da linguagem Java apresentará sempre as limitações inerentes a algo que foi desenhado e desenvolvido com outro fim. A JavaRMI é uma tecnologia Java-Java pelo que esta é a

única linguagem suportada directamente por esta plataforma. Para ultrapassar esta limitação a JavaRMI disponibiliza uma API chamada JNI (*Java Native Interface*) que permite ao código Java chamar ou ser chamado por rotinas implementadas noutras linguagens. O inconveniente em recorrer a esta solução é o aumento significativo da complexidade das aplicações resultante da utilização das várias linguagens [Curtis - 1997].

6.2.2. SUPORTE DE PLATAFORMAS

O suporte de diferentes plataformas, tal como o suporte de diferentes linguagens, é também um factor bastante importante no que respeita à interoperabilidade. As arquitecturas devem ser capazes de suportar quer plataformas recentes, quer plataformas antigas e se necessário suportá-las em conjunto (heterogeneidade).

CORBA

Um dos aspectos centrais desta arquitectura foi sempre o suporte multiplataforma. Existem actualmente ORBs CORBA para mais de 30 plataformas diferentes e, inclusivamente, são suportados por esta arquitectura mais sistemas operativos da Microsoft que pela própria arquitectura proprietária da Microsoft o DCOM [Tallman - 1998]. A título de exemplo, só o ORB Orbix da IONA Technologies suporta actualmente mais de 12 plataformas diferentes [Orbix - 1999a].

DCOM

Até 1998 o DCOM esteve praticamente limitado às plataformas Microsoft Windows 95 e NT [Mitre - 1998], mesmo depois de uma aproximação por parte da Microsoft à empresa alemã Software AG para fazer a passagem do DCOM para Unix. Esta passagem no entanto não incluiu tecnologias como a MTS e a MSMQ, o que tornava inviável o uso da tecnologia DCOM como plataforma intermediária ao nível das grandes empresas [Meta -1998]. A Microsoft anuncia então a intenção de utilizar equipas de trabalho próprias para fazerem a passagem do DCOM para as diferentes plataformas existentes. Actualmente o DCOM além de se estar disponível para as plataformas Microsoft também se encontra disponível para as plataformas Solaris, DEC Unix, HPUX, Linux, MVS, OS/400, AIX, VMS, SINIX, SCO UnixWare e Mac [Kindel][Maloney].

JAVARMI

Tendo em conta a popularidade da linguagem Java, são cada vez mais os fabricantes que anunciam o suporte pelas suas plataformas da JVM. Uma vez que o modelo JavaRMI tem como suporte a linguagem Java, todas as plataformas com suporte da JVM estão aptas a poderem utilizar JavaRMI. Actualmente além da Javasoft e da Microsoft já várias empresas anunciaram portos JVM para as suas plataformas nomeadamente: AIX, HPUX, Linux, MacOS e Windows [Ports].

6.2.3. COMUNICAÇÕES EM REDE

Inevitavelmente as plataformas intermediárias deverão fornecer um suporte robusto e diversificado relativamente aos mecanismos de comunicação para que seja possível a sua integração com as diferentes tecnologia existentes. Para que isto seja possível as plataformas intermediárias deverão ser transparentes relativamente ao protocolo.

CORBA

O modelo de comunicações da arquitectura CORBA está baseado no IIOP. Este protocolo é orientado à conexão e foi desenhado com a intenção de assegurar que todos os ORBs utilizem um protocolo de comunicações comum. No entanto internamente no ORB é possível também a utilização de outros protocolos.

DCOM

O DCOM baseia-se na especificação DCE RPC da OSF com algumas extensões [Chung]. Numa fase inicial o protocolo utilizado pelo DCOM foi o UDP (*User Datagram Protocol*), sendo actualmente o utilizado o protocolo TCP.

JAVARMI

O modelo de comunicações da arquitectura JavaRMI tem-se baseado no protocolo RMP, protocolo proprietário da Javasoft. Este protocolo tem vindo a ser preterido em relação ao IIOP pois segundo alguns autores, a arquitectura JavaRMI deverá ser suportada pelo IIOP [Curtis - 1997][Rocheleau - 1997]. Esta troca de protocolo permitirá à arquitectura JavaRMI (JavaRMI-IIOP) melhorar consideravelmente as suas características, não fazendo sentido por parte da Javasoft continuar a desenvolver o seu protocolo proprietário RMP [Orfali - 1998]. Esta mudança deu origem às designações JavaRMI-IIOP (JavaRMI sobre IIOP) e JavaRMI-RMP (JavaRMI sobre RMP).

6.2.4. SERVIÇOS COMUNS

Os Serviços Comuns são a infra-estrutura base onde assentam as plataformas, dependendo da sua implementação grande parte do sucesso das mesmas.

Se tivermos em conta a terminologia do OMG, uma implementação mínima deve incluir os seguintes serviços genéricos: transacções, directório, mensagens e segurança. Outro factor importante a ter em conta é a facilidade de integração de serviços desenvolvidos por terceiros para cada uma das plataformas.

CORBA

Durante o processo de desenvolvimento da arquitectura CORBA, o OMG deu especial atenção ao capítulo dos serviços comuns tendo no desenho dos mesmos considerado logo o factor interoperabilidade. A norma CORBA define 15 serviços diferentes. Contudo, tal como pode ser visualizado na Tabela VI correspondente a implementações de ORBs disponíveis, só alguns são

implementados. Segundo as empresas que desenvolveram estes ORBs, a implementação dos serviços está dependente das necessidades dos seus clientes.

DCOM

Comparativamente com a CORBA, a DCOM está relativamente atrás pois apenas implementa, e de uma forma limitada, os serviços de nomes, de transacções e de segurança [Meta - 1998]. O serviço de directório deverá estar disponível com o Windows NT 5 [Tallman - 1998]. Serviços como MSMQ e MSCS (*Microsoft Clustering Technology*) encontram-se disponíveis não tendo sido contudo integrados pela Microsoft na especificação da sua arquitectura DCOM. Estes serviços fazem já parte da "nova" arquitectura COM+[23].

JAVARMI

A JavaRMI recorre às APIs JNDI (*Java Naming and Directory Interface*), JMS (*Java Messaging Service*) e JTS (*Java Transaction Service*) para implementação dos serviços de nomes, directórios, mensagens e transacções.

6.3. FIABILIDADE

Pretende-se neste ponto analisar as capacidades de cada uma das plataformas relativamente a questões como:

- Transacções.
- Mensagens.
- Segurança.
- Directórios.
- Tolerância a falhas.

6.3.1. TRANSACÇÕES

Nos últimos anos, este aspecto tem sido o centro das atenções das plataformas intermediárias dada a importância que estas assumiram nos vários contextos das TIs.

CORBA

A norma do serviço de transacções da arquitectura CORBA especifica um vasto leque de serviços para suporte a transacções distribuídas. Estes serviços permitem além da utilização das transacções *flat* tradicionais a utilização de transacções *nested*. Este serviço permite que transacções ORB e transacções não-ORB participem na mesma transacção, permitindo que desta forma transacções de objectos e transacções de procedimentos

[23] O COM+ = MTS (*Microsoft Transaction Server*) + COM.

interoperem. São também suportadas transacções entre ORBs heterogéneos, o que significa que vários ORBs podem participar na mesma transacção [Meta - 1998].

Em conjunto com o serviço controle da concorrência, é suportado *commit* total, *rollback*, fechos e outras capacidades.

DCOM

Neste aspecto a arquitectura DCOM também é uma ferramenta bastante poderosa através da utilização do MTS [Orfali - 1998]. No entanto a sua reduzida disponibilidade (unicamente para plataformas Intel) é um factor de limitação importante.

O MTS é disponibilizado como uma extensão ao DCOM, mas é parte integrante do COM+ [Raj].

JAVARMI

A JavaRMI-RMP não possui suporte transaccional. Contudo tendo em consideração a observação feita anteriormente no ponto 6.2, isto é, a tendência para a descontinuação da JavaRMI-RMP pela Javasoft o suporte transaccional da JavaRMI, neste caso JavaRMI-IIOP, é assegurado pelo serviço de transacções da arquitectura CORBA. Assim todas as características anteriormente descritas relativamente a este aspecto para a arquitectura CORBA são válidas para a JavaRMI-IIOP.

6.3.2. MENSAGENS

A transmissão e recepção de mensagens de forma fiável tem que ser uma garantia das plataformas intermediárias. Sem esta garantia imagine-se por exemplo o que poderia acontecer no emergente comércio electrónico.

Desta forma num sistema de mensagens devem ser cumpridas quatro importantes características: desempenho, fiabilidade, conveniência do utilizador e conveniência do sistema [Meta - 1998].

No sistema de mensagens fiabilidade significa entrega garantida, ou seja, no caso de haver algum problema a mensagem é retida e enviada logo que possível. No que respeita às outras três características elas estão correlacionadas. Por conveniência do utilizador entende-se que as partes receptora e emissora das mensagens não têm que estar num determinado lugar a uma determinada hora para receber e enviar mensagens. A designação técnica associada a este facto é comunicação assíncrona. Com este tipo de comunicação o remetente ou o sistema não têm que esperar que a mensagem seja enviada e recebida para poder voltar a trabalhar normalmente. Para suportar esta característica as plataformas devem disponibilizar um sistema robusto de filas de armazenamento de mensagens.

CORBA

A arquitectura CORBA delega no serviço de eventos esta tarefa. Este serviço é a base de vários protocolos de mensagens onde se destacam os protocolos *push-pull* e *pull-push*. O modelo de mensagens da especificação da arquitectura CORBA tem suporte para comunicações assíncronas.

DCOM

O DCOM não suporta directamente comunicação assíncrona, no entanto este problema foi ultrapassado pela Microsoft recorrendo a um mecanismo complementar intitulado MSMQ ou Falcon que foi o primeiro nome desta tecnologia. O facto deste mecanismo não ser parte integrante do DCOM, faz com que tenha algumas limitações no que respeita a interoperabilidade uma vez que se encontra quase unicamente disponível para plataformas Intel tal como o serviço MTS.

JAVARMI

Recorre ao JMS para a implementação do serviço de mensagens [JMS - 1999]. Este serviço possibilita comunicação assíncrona.

6.3.3. SEGURANÇA

A segurança é actualmente considerada como um factor decisivo no sucesso/insucesso dos sistemas de objectos distribuídos dado que um número crescente de aplicações fornecem serviços fundamentais para o funcionamento normal das organizações.

CORBA

A especificação do serviço de segurança da arquitectura CORBA é uma das melhores especificações de segurança existente para computação distribuída. Este serviço entre outros considera os seguintes aspectos: integridade, responsabilidade, disponibilidade, confidencialidade e não-repudiação [Coimbra - 1998]. Este serviço define 3 níveis de segurança, desde ORBs com poucas preocupações ao nível de segurança (nível 0), a ORBs que requerem a totalidade dos serviços (nível 2): controle de acessos, delegação, auditoria, autenticação e políticas de implementação. Este serviço contempla o suporte para o protocolo SSL [Meta - 1998].

DCOM

O DCOM utiliza o mecanismo de segurança do Windows NT como base do seu suporte de segurança [Orfali - 1998]. A este mecanismo de segurança foi atribuído o nível C2 de segurança do NSCS (*National Computer Security Center*). Além do nível C2 proporcionado pelo Windows NT o DCOM também disponibiliza a *CryptoAPI* que possibilita serviços de encriptação de informação.

A combinação Windows NT, MTS e COM disponibiliza um ambiente com um bom grau de segurança apresentando contudo uma grande deficiência que é o facto da segurança DCOM estar dependente do sistema operativo Windows NT [Meta - 1998].

JAVARMI

O suporte de segurança da JavaRMI (JavaRMI-IIOP) tem como base o serviço de segurança da arquitectura CORBA pelo que o que foi apresentado anteriormente relativamente a esta arquitectura é válido para a JavaRMI.

6.3.4. DIRECTÓRIOS

Uma das características de relevo a ter em conta nas diferentes plataformas intermediárias é a sua capacidade de registar e manter a localização dos serviços. Exemplos de serviços de directórios (nomes) são: o DNS (*Domain Name System*), o X.500, o Novell NDS e o Microsoft NTDS (*Windows NT Directory Service*) cada um acedido por uma interface especializada.

CORBA

Para este efeito, o OMG especificou o serviço de nomes que é um serviço similar às páginas amarelas [Orfali - 1998]. Este serviço vai permitir a um componente procurar um serviço pelo nome. A especificação deste serviço teve em conta a sua possível utilização com os serviços convencionais de directórios especificados anteriormente.

DCOM

A resposta da Microsoft a esta necessidade foi o ADS (*Active Directory Service*). Este serviço combina características dos serviços X.500 e NDS [Meta - 1998]. Tal como na arquitectura CORBA o objectivo deste serviço é a abstracção das diferenças entre os vários serviços de directórios através da disponibilização de uma interface normalizada.

JAVARMI

Para este efeito, o serviço disponibilizado pela JavaRMI foi o RMIRegistry. A especificação deste serviço teve em conta a sua possível utilização com serviços convencionais como LDAP, NDS, DNS e NIS [JNDI - 1999].

6.3.5. TOLERÂNCIA A FALHAS

A tolerância a falhas pode ser caracterizada pela forma como as plataformas intermediárias são capazes de se superar a si próprias. São vários os mecanismos de suporte que podem contribuir para esta capacidade, sendo um desses serviços a comunicação assíncrona.

CORBA

A CORBA não faz referência directa a serviços de tolerância a falhas, contudo vários ORBs disponibilizam suporte para esta característica. A maioria das empresas que disponibilizam ORBs CORBA, para ultrapassar este problema recorreram a um mecanismo de *timeout* para a detecção de clientes "mortos" ou desligados [Meta - 1998].

DCOM

O suporte de tolerância a falhas no DCOM é disponibilizado ao nível do protocolo. O mecanismo utilizado é bastante simples e baseia-se

num contador de referências. Sempre que um objecto se liga a um servidor o contador é incrementado de uma unidade. O modo como o DCOM determina se a ligação objecto-servidor ainda se mantém é através do recurso a mensagens *keep alive* e *pinging* ou seja, de 2 em 2 minutos é enviada uma mensagem entre o cliente (objecto) e o servidor; sempre que a mensagem é transmitida com sucesso, significa que a conexão se mantém activa, caso contrário e ao fim de 3 mensagens sem sucesso (= 6 minutos) o servidor declara o cliente "morto", termina a ligação e decrementa o contador de uma unidade. Ao atingir-se o valor 0 no contador de referências significa que o servidor pode ser removido.

JAVARMI

Neste caso tal como no anterior é utilizado um contador de referências. As referências são consideradas válidas quando existe uma conexão entre um cliente e um servidor através de uma sessão TCP/IP. Cada vez que um cliente obtém uma referência, o contador de referências é incrementado de uma unidade e decrementado de uma unidade quando o cliente deixa de referenciar o objecto. Quando este contador atingir o valor zero a JavaRMI coloca o objecto servidor na lista *weak reference* podendo de seguida o colector de objectos fazer a remoção desse objecto [Orfali - 1998].

6.4. MATURIDADE DA PLATAFORMA

Se a maturidade de uma tecnologia pudesse ser medida pelo número efectivo de sistemas desenvolvidos e em desenvolvimento, a arquitectura CORBA levaria uma grande vantagem relativamente aos seus concorrentes DCOM e JavaRMI. Um outro factor importante a favor da arquitectura CORBA tem a ver com o *know-how* já adquirido pela comunidade de desenvolvimento CORBA. Talvez até mais importante que a gama de serviços disponibilizados pelas diferentes plataformas é o factor humano que tem nas suas mãos o poder de decidir da aceitação ou não de uma determinada tecnologia.

Assim, relativamente a este aspecto poder-se-á considerar que a arquitectura CORBA está largamente à frente de todas as suas concorrentes onde se incluem as tecnologias DCOM e JavaRMI [Tallman - 1998].

CAPÍTULO 7

7. CONCLUSÕES

As Arquitecturas Distribuídas Cliente/Servidor vivem momentos de grande agitação que são indicativos de rápidas e significativas mudanças. Como elementos catalisadores desta dinâmica temos por um lado os avanços tecnológicos e por outro a competição feroz entre os desenvolventes destas arquitecturas com o objectivo de conseguirem a maior fatia de mercado neste sector.

Assim, julgou-se oportuno apresentar e analisar, de uma forma tão global quanto possível o estado da arte nesta área da computação distribuída. Tal análise pretendeu identificar os objectivos básicos destas arquitecturas e as suas principais vantagens comparativamente a outras soluções.

As arquitecturas escolhidas foram a CORBA e a DCOM pela representatividade de ambas neste sector e a JavaRMI devido à crescente popularidade da linguagem Java. Apesar de se poder afirmar que estas arquitecturas estão agora a atingir a sua maturidade, decorrem ainda vários esforços por parte das empresas desenvolventes, no sentido de munir as suas plataformas com cada vez mais características, tendo em vista o ganhar vantagem relativamente às plataformas concorrentes.

Neste ponto a plataforma CORBA leva ainda uma grande vantagem relativamente às suas duas concorrentes deste documento, podendo ser considerada a escolha de menor risco, devido à sua arquitectura interna, maturidade e capacidade de interoperação com as plataformas DCOM e JavaRMI.

É de salientar também o esforço da Microsoft no sentido de fazer vingar a sua tecnologia - o DCOM. Embora se encontre num estado de desenvolvimento inferior ao da arquitectura CORBA mas mais avançado do que o da arquitectura JavaRMI, o DCOM leva uma vantagem considerável nos ambientes Windows/Intel onde a Microsoft detém um quase monopólio relativamente a sistemas operativos.

No que respeita à arquitectura JavaRMI é das três plataformas a que se encontra num estado de desenvolvimento inferior. O futuro desta tecnologia revela-se um tanto ou quanto incerto se atendermos à estreita colaboração que tem vindo a sentir-se entre o OMG e a Javasoft. Esta incerteza não tem a ver com a possibilidade de desaparecimento da JavaRMI, mas sim com a possibilidade de diluição desta arquitectura na arquitectura CORBA, atendendo a algumas das soluções encontradas para ultrapassar deficiências manifestadas pela JavaRMI.

A título de conclusão final, a escolha para uma solução global e de uma forma resumida recairia em:

- Arquitectura CORBA para ambientes heterogéneos (multi-plataforma), com utilização de *bridges* COM para integração dos *desktops*.

- Arquitectura COM/DCOM/COM+ para ambientes homogéneos baseados em Windows/Intel, com utilização da CORBA para determinadas capacidades de distribuição.

- Arquitectura JavaRMI para ambientes onde o modelo de objectos Java ocupe lugar de destaque.

REFERÊNCIAS BIBLIOGRÁFICAS

[ActCOR] OMG. *Comparing ActiveX and CORBA/IIOP*. http://www.omg.org/library/activex.html.

[Armstrong - 1999] Armstrong, T., 1999. *COM + MTS = COM+, Next Step in the Microsoft Component Strategy*. http://www.vcdj.com/magazine/febmag99/commts1.asp.

[Baker - 1997] Baker, S., 1997. *CORBA Distributed Objects - using Orbix*. Addison-Wesley.

[Berard - 1999] Berard, EV., Março 1999. *Motivation for an Object-Oriented Approach to Software Engineering*. Berard Software Engineering, Inc. http//www.toa.com/pub/net_articles/motivation_article.txt.

[Brooks - 1975] Brooks, FP., 1975. *The Mythical Man Month: Essays on Software Engineering*. Reading, MA: Addison-Wesley.

[Chung] Chung, PE., Huang, Y., Yajnik, S., Liang, D., Shih, J., Wang, CY., Wang, YM.. *DCOM and CORBA Side by Side, Step by Step, and Layer by Layer*. Bell Laboratories, Lucent Technologies, IIS, Academia Sinica - China, Microsoft Research. http://www.bell-labs.com/~emerald/dcom_corba/Paper.html.

[Coimbra - 1998] Coimbra, CM., Setembro 1998. *Segurança na Arquitectura CORBA*. Departamento de Electrónica e Telecomunicações - Universidade de Aveiro.

[CORBAPlus - 1999] Março 1999. *ORB CORBAPlus*. Expertsoft, Inc. http://www.expersoft.com/Products/products.htm.

[Cortés - 1998] Cortés, AR., Novembro 1998. *CORBA: Una Visión General*. Universidad de Sevilla. http://www.lsi.us.es/~aruiz/semcorba/.

[Curtis - 1997] Curtis, D., 1997. *Java, RMI and CORBA*. Object Management Group. http://www.omg.org/library/wpjava.html

[DAIS - 1999] Março 1999. *ORB DAIS*. PeerLogic, Inc. http://www.peerlogic.com/products/dais/f_dais.htm.

[DCOM - 1996] Novembro 1996. *DCOM Technical Overview*. Microsoft Corporation. http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/backgrnd/html/msdn_dcomtec.htm.

[Javasoft - 1998] Outubro 1998. *Java Remote Method Invocation Specification. Revision 1.50, JDK 1.2*. Javasoft. http://www.javasoft.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html.

[JMS - 1999] *JMS - Java Message Service API*. http://java.sun.com/products/jms/.

[JNDI - 1999] *JNDI - Java Naming and Directory Interface API. Overview*. http://java.sun.com/products/jndi/overview.html.

[Kindel] Kindel, C. *An Overview of Active X*. Microsoft Developer Relations Group. Microsoft Corporation. http://www.microsoft.com/com/presentations/default.asp.

[Lindholm] Lindholm, T., Yellin, F. *The JavaTM Virtual Machine Specification, 2nd Edition*. http://www.javasoft.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html.

[Maloney] Maloney, J. *"COM+" Building on the Success of the Component Object Model*. Object Marketing. Microsoft Corporation. http://www.microsoft.com/com/presentations/default.asp.

[Matthews] Matthews, C. *Introduction to Java Remote Method Invocation (RMI)*. http://www.edm2.com/0601/rmi1.html.

[Mens - 1997] Mens, K., Julho 1997. An Introduction to OO. http://progwww.vub.ac.be/prog/general/intro_OO.html.

[Meta - 1998] Março 1998. *CORBA vs. DCOM: Solutions for the Enterprise*. META Group Consulting.

[Mitre - 1998] Abril 1998. *Recommendations for Using DCE, DCOM, and CORBA Middleware*. Mitre Corporation.

[Moeller - 1998] Moeller, M., Abril 1998. *COM+ services add up*. PCWEEK OnLine. http://www.zdnet.com/pcweek/news/0413/13com.html.

[Mowbray - 1995] Mowbray, T., Zahavi, R., 1995. *The Essential CORBA - Systems Integration Using Distributed Objects*. John Wiley & Sons.

[MSFT - 1995] Outubro 1995. *The Component Object Model Specification*. Microsoft Corporation and Digital Equipment Corporation. Draft Version 0.9.

[OAK - 1999] Março 1999. *ORB OAK*. Paragon Software Inc. http://www.paragon-software.com/.

[OMA] *The OMA Reference Model*. OMG. http://www.omg.org/oma/index.htm

[OMG - 1999] OMG. 1999. *History of CORBA*. http://www.omg.org/corba/corbahistory.html

[OMG - 1994] OMG, Siegel, J., 1994. *Common Object Services Specification, Volume I*, by AT&T/NCR, BNR Europe, Digital, Groupe Bull, Hewlett-Packard, HyperDesk, ICL PLC, IBM, Itasca Systems, Novell, O2 Technology SA, Object Design, Objectivity, Ontos, Oracle, Persistence Software, Servio, SunSoft, Teknekron Software, Tivoli Systems, and Versant Object Technology. OMG Document No. 94-1-1, Rev. 1.0, First Edition. New York: John Wiley & Sons.

[OMG - 1995] OMG. 1995, updated March 28, 1996. *CORBAservices: Common Object Services Specification* (self-published).

[OMG - 1995a] OMG. 1995, revision 2.2 February, 1998. *The Common Object Request Broker: Architecture and Specification* (self-published).

[OMG - 1997] OMG. 1997. *A Discussion of the Object Management Architecture* (self-published).

[OMG - 1998] OMG. 1998.
http://www.omg.org/omg/background.html.

[omniORB - 1999] Março 1999. ORB omniORB. AT&T Laboratories.
http://www.uk.research.att.com/omniORB/omniORB.html.

[ORBex - 1999] Março 1999. *ORB ORBexpress*. Objective Interface Systems, Inc.
http://www.ois.com/Products/Items/Oe_and_corba.htm.

[Orbix - 1999a] 1999. *Orbix and OrbixWeb - Tools for Application Integration*. IONA Technologies PLC.
http://www.iona.com/online/support/whitepapers/index.html.

[Orbix - 1999] Março 1999. *ORB Orbix*. IONA Technologies PLC.
http://www.iona.com/info/products/orbixchoice.html.

[Orfali - 1996] Orfali, R., Harkey, D., Edwards, J. 1996. *The Distributed Objects - Survival Guide*. John Wiley & Sons.

[Orfali - 1998] Orfali, R., Harkey, D., 1998. *Client/Server Programming with JAVA and CORBA*. 2nd Edition. John Wiley & Sons.

[Petersen - 1998] Petersen, S., Março 1998. *For Microsoft, COM + MTS = Component Services Architecture*. PCWEEK OnLine.
http://www.zdnet.com/pcweek/news/0323/23mcom.html.

[Pope - 1997] Pope, A., Dezembro 1997. *The CORBA Reference Guide - Understanding the Common Object Request Broker Architecture*. Addison-Wesley.

[Ports] *Java Platform Ports*.
http://java.sun.com/cgi-bin/java-ports.cgi.

[Raj] Raj, GS. *COM+*.
http://www.execpc.com/~gopalan/com/complus.html.

[RCPORB - 1999] Março 1999. *ORB RCP-ORB*. Nortel Networks Corp.
http://www.nortel.com/RCP-ORB/.

[Rocheleau - 1997] Rocheleau, C., Junho 1997. *JavaRMI To Embrace CORBA/IIOP. Sun and OMG Reaffirm Close Working Relationship*.
http://www.omg.org/news/pr97/rmiiop.html.

[Siegel - 1999] Siegel, J., Junho 1999. *What's Coming in CORBA 3?*. OMG in Motion Magazine. OMG.
http://www.omg.org.

[Siegel - 1996] Siegel, J., 1996. *CORBA Fundamentals and Programming*. John Wiley & Sons.

[Sun - 1999] Maio1999. *Java Remote Method Invocation*. Sun Microsystems.
http://www.java.sun.com/marketing/collateral/rmi_ds.html.

[Tallman - 1998] Tallman, O., Kain, JB., Dezembro 1998. *COM versus CORBA: A Decision Framework*.
http://www.quoininc.com/quoininc/COM_CORBA.html.

[TIB/OB - 1999] Março 1999. *ORB TIB/ObjectBus*. Tibco, Inc.
http://www.tibco.com/products/objectbus/index.html.

[Vinoski - 1997] Vinoski, S., Fevereiro 1997. *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments*. IEEE Communications Magazine.

[Vogel - 1997] Vogel, A., Duddy, K., 1997. *Java Programming with CORBA*. John Wiley & Sons.

BIBLIOGRAFIA

Afonso, A., Outubro 1993. *Bases de Dados Distribuídas: Realidades e Perspectivas*. Provas de Aptidão Pedagógica. Universidade de Aveiro.

An Overview of RMI Applications. Javasoft. http://www.javasoft.com/docs/books/tutorial/rmi/overview.html.

Anderson, TE., Culler, DE., Patterson, DA., Fevereiro 1995. *A Case for NOW (Networks and Workstations)*. University of California at Berkeley, IEEE Micro.

Berg, C., Janeiro 1997. *How Do I Use CORBA from Java?*. Dr. Dobb's Journal.

Betz, M., Outubro 1996. *Putting Objects in Their Place*. http://www.sdmagazine.com/breakrm/features/s96of1.shtml.

Carr, DF., Março 1997. *CORBA and DCOM: How Each Works*. http://www.internetworld.com/print/1997/03/24/software/cobra.html.

Cetus Links: Links on Objects and Components. http://www.cetus-links.org/.

Chappel, D., Dezembro 1997. *COM+: The Next Generation*. BYTE Magazine.

Chappel, D., Janeiro 1998. *How Microsoft Transaction Server Changes the COM Programming Model*. Microsoft Systems Journal. http://www.microsoft.com/msj/0198/mtscom/mtscom.htm.

Choosing between CORBA and DCOM. http://www.cerfnet.com/~mpcline/Corba-FAQ/corba-and-dcom.html.

Coelho, P., Novembro 1996. *Programação em Java*. FCA Editora.

Condon, R., Julho 1997. *CORBA no match for Microsoft's DCOM, Ovum report says*. http://www.computerworld.com/home/online9697.nsf/all/970807corba.

Distributed Systems Group of the Information Systems Institute, 1998. Technical University of Vienna. http://www.infosys.tuwien.ac.at.

Distributed Systems Group of the Information Systems Institute. 1998. *The Common Object Request Broker: Architecture and Specification*. Technical University of Vienna. http://www.infosys.tuwien.ac.at/Research/Corba/OMG/corb2prf.htm.

Frey, A., Julho 1997. *Is DCOM Truly the Object of Middleware's Desire?*. http://www.nwc.com/813/813r1.html.

Gonsalves, A., Janeiro 1998. *Iona licenses COM technology from Microsoft*. PCWeek Online. http://www.zdnet.com/pcweek/news/0126/26eiona.html.

Helm, R., Watson, TJ., 1992. *Ensuring Semantic Integrity of Reusable Objects (Panel)*. OOPSLA '92 Conference Proceedings.

Java Distributed Computing. http://www.javasoft.com/products/javaspaces/.

JTS - Java Transaction Service. Javasoft. http://java.sun.com/products/jts/.

Karpinski, R., Outubro 1998. *CORBA 3.0 Eases Object Development*. BYTE Magazine.

Kindel, C. *ActiveX and The Web Architecture & Technical Overview*. Microsoft Developer Relations Group. Microsoft Corporation. http://www.microsoft.com/com/presentations/default.asp.

Lamping, J., 1993. *Typing the Specialization Interface*. OOPSLA '93 Conference Proceedings.

Linthicum, D., Junho 1999. *Mastering Message Brokers*. http://www.sdmagazine.com/breakrm/features/s996f3.shtml.

Loosley, C., Janeiro 1998. *Designing Distributed Applications*. PC Magazine.

Maffeis, S., Schmidt, D., Fevereiro 1997. *Constructing Reliable Distributed Communication Systems with CORBA*. IEEE Communications Magazine.

Marques, J., Guedes, P., Maio 1998. *Tecnologia de Sistemas Distribuídos*. FCA Editora.

Microsoft Distributed Transaction Coordinator (MS DTC). http://www.execpc.com/~gopalan/com/msdtc.html.

Microsoft Queuing Overview and Resources. Microosft. http://www.microsoft.com/ntserver/appservice/exec/overview/MSMQ_Overview.asp.

Montgomery, J., Abril 1997. *Distributing Components*. BYTE Magazine.

Norman, RJ., Outubro 1997. *Middleware: CORBA and DCOM* - White Paper. IDS Dept, San Diego State University.

OMG. *OMG in Motion Magazine*. Junho 1999. http://www.omg.org.

Orfali, R., Edwards, J., Harkey, D., Abril 1997. *CORBA, Java, and the Object Web*. http://www.sdmagazine.com/breakrm/features/s974f1.shtml.

Orfali, R., Harkey, D., 1997. *Client/Server Programming with JAVA and CORBA*. John Wiley & Sons.

Orfali, R., Harkey, D., Abril 1995. *Client/Server with Distributed Objects*. BYTE Magazine.

Pompeii, J., Abril 1997. *Programming with CORBA and DCOM*. BYTE Magazine.

Postel, J., Setembro 1981. *Transmission Control Protocol - DARPA Internet Program Protocol Specification*. RFC-793, Information Sciences Institute.

Quantitative Data Systems. Dezembro 1997. *Introduction to Objects* (self-published).

Raj, GS. *A Detailed Comparison of CORBA, DCOM and Java/RMI*. http://www.execpc.com/~gopalan/misc/compare.html.

RMI versus CORBA. Caribou Lake Software. http://www.cariboulake.com/techinfo/rmi_corba.html.

Romero, D. *Remote Method Invocation*. Univesidad de Chile. http://www.dcc.uchile.cl/~dromero/contenidormi.htm.

Rosenberger, J., 1998. *CORBA in 14 Days*. SAMS Publishing.

Sessions, R., 1998. COM and DCOM. *Microsoft's Vision for Distributed Objects*. Wiley Computer Publishing.

Szperski, C., Maio 1999. *Components and Objects Together*. http://www.sdmagazine.com/breakrm/features/s995f2.shtml.

Technical Overview: Clustering and Windows NT Load Balancing Service (WLBS). Microsoft. http://www.microsoft.com/ntserver/ntserverenterprise/techdetails/overview/ntsee.asp.

Yee, A., Junho 1999. *Making Sense of the COM vs. CORBA Debate*. http://www.performancecomputing.com/features/9906dev.shtml.