Utilização da Arquitectura CORBA num Sistema de Televigilância

Pedro Borges, Telmo Silva, Filipe Patrício, Fernando Ramos

***Resumo* – Este artigo apresenta a experiência tida na utilização da arquitectura distribuída CORBA como middleware de suporte de uma aplicação de televigilância.**
Pretendeu-se compreender a estratégia de construção de aplicações baseadas neste middleware e avaliar o respectivo desempenho.

***Abstract* – This paper presents the experience of usage of the distributed architecture CORBA as support middleware of a telesurveillance system.**
The basic aims of this project were to understand the strategy to build CORBA based applications and to assess its performance.

I. O CONCEITO

A. *Viabilidade*

A viabilidade do conceito de televigilância do ponto de vista económico é óbvia; os recursos humanos empregues são menores, o que por si só, implica uma redução acentuada dos custos do sistema.

Considerando um sistema com *n* máquinas remotas em constante funcionamento e alguns postos de trabalho (muito menor que n) em alerta, verifica-se que o número de recursos humanos empregues, se resume ao número de postos de trabalho mais os elementos encarregues da manutenção, o que comparado com 3*n vigilantes necessários para assegurar o sistema 24 horas por dia, é uma opção bastante mais racional. Considerando, também, a probabilidade de erro no sistema, esta é inversamente proporcional ao número de máquinas usadas, aumentando assim a lógica da opção. Falta ainda abordar o aspecto social e humano; é de notar, e é óbvio que o número de empregos disponíveis diminui, no entanto, em caso de falhas, a substituição de uma máquina não se compara à impossível substituição de uma vida, tornando a opção por um sistema de televigilância, mais correcta .

Estes sistemas, implicam transferências de informação entre máquinas distantes para que se possa efectuar a vigilância. Existe então uma distribuição de informação, que para circular necessita de "*algo*" que interligue os seus fornecedores.

B. *Sistemas Distribuídos*

Os sistemas distribuídos foram, nesta última década, um dos temas que maior interesse tem despertado e que se tem traduzido numa evolução constante nos sistemas computacionais. Primeiramente na comunidade cientifica, em particular na ligada aos sistemas operativos e bases de dados, onde numerosos sistemas experimentais foram desenvolvidos e analisados.

Quando do aparecimento das redes locais e dos computadores pessoais, as arquitecturas distribuídas começaram a influenciar a estrutura informática das grandes empresas e organizações e, mais recentemente com a expansão da Internet, tornou-se um assunto do domínio público, despertando um interesse generalizado.

É óbvio que os sistemas operativos evoluíram para responder aos desafios colocados por estas novas arquitecturas. Um dos problemas verificou-se ao nível da comunicação baseada em mensagens; num sistema operativo, a comunicação efectua-se através de um espaço de endereçamento partilhado. Quando se considera um ambiente distribuído, a utilização de meios de comunicação que não partilham uma memória física comum, introduz características diferentes no modelo de comunicação.

A comunicação à distância fica sujeita a um conjunto de factores que podem afectar a sua fiabilidade. Apesar da taxa de erro nas comunicações de dados registar uma assinalável diminuição, continua a ser diferente da que se obtém internamente numa máquina. Em diversos tipos de redes, as relações de ordem entre as mensagens enviadas e recebidas não são determinísticas, devido a perdas, duplicações de mensagens e atrasos que as várias entidades que constituem o sistema, podem introduzir. As falhas nos sistemas distribuídos têm, simultaneamente, um maior número de causas e uma maior probabilidade de ocorrência devido ao maior número de componentes (controladores de rede, ligações físicas e equipamentos de transmissão).

As arquitecturas distribuídas implicam uma distribuição do software que controla o sistema o que dificulta a sincronização e o acesso à informação de gestão; por isso tem também de existir um protocolo de sincronização para facilitar e aumentar a fiabilidade do acesso à informação.

É lógico que tendo uma máquina ligada a uma rede num sistema distribuído, a segurança dessa máquina é posta à

prova, ficando vulnerável porque não existe nenhum mecanismo de segurança física que impeça a escuta de mensagens ou mesmo a sua substituição ou falsificação. Este é assim um grande problema dos sistemas distribuídos.

Mas afinal, onde estão as vantagens?…

No fundo, com um sistema distribuído, pretende-se melhorar a produtividade dos utilizadores e obter um custo mais reduzido.

Tendo uma maior quantidade de informação disponível, dispendemos muito menos tempo para a conseguir, reduzindo o custo. Para além disto, temos serviços disponíveis que nos facilitam determinadas tarefas, aumentando a produtividade. Deve-se considerar também, que a capacidade de evolução de um sistema é limitada. Numa máquina, é sempre possível aumentar a memória central e a capacidade em disco, mas a estrutura básica do processador e do *bus*, limita o incremento do processamento. Numa aproximação distribuída, a capacidade de processamento pode ser aumentada facilmente.

Justifica-se assim a popularidade dos sistemas distribuídos.

C. Programação

Na perspectiva do desenvolvimento de aplicações para funcionar num ambiente distribuído, o programador, tem de entender o protocolo de comunicação para aceder a máquinas remotas. Assim, o nível de abstracção é pequeno ou nulo, obrigando-o a gerir muito bem a informação enviada. O programador não se consegue desligar do conceito *máquina remota*.

Numa tentativa de aumentar a abstracção, nos últimos anos, tem-se assistido à tentativa de desenvolvimento de plataformas de comunicação que permitam ao programador transferir informação de uma forma mais simples, de modo que, o que é remoto pareça local. Foi neste conceito que se baseou este projecto.

II. A EVOLUÇÃO

Nos primórdios da utilização das redes, os sistemas operativos limitavam-se às máquinas individuais. Para que a comunicação fosse possível, o computador necessitava de ter uma interface de '*hardware*' com a rede e o respectivo gestor de periférico que permitisse aos processos enviar e receber dados. As aplicações distribuídas eram programadas, utilizando directamente a interface do gestor de periférico, cabendo ao programador resolver os problemas de gestão de recursos.

A evolução natural que permitiu diminuir as dificuldades de programação e transporte das aplicações foi a virtualização destas interfaces, oferecendo uma API (aplicação normalizada) estável suportada pelo sistema operativo. Do ponto de vista do programador, o suporte à comunicação materializa-se numa biblioteca de funções para o envio e recepção de mensagens que não esconde totalmente os detalhes dos protocolos de transporte.

A. Arquitectura Cliente/Servidor

Num ambiente distribuído, uma programa é uma cooperação entre duas aplicações, o que vulgarmente se designa por arquitectura Cliente/Servidor. Esta designação, estabelece a distinção entre dois tipos de processos com comportamentos diferentes e, portanto, assimétricos na sua estrutura, que podem ser sinteticamente caracterizadas:

- os servidores implementam um conjunto de funções de interesse geral para outros processos que remotamente lhes podem aceder;
- os clientes efectuam os interfaces com os utilizadores, podem executar parte das aplicações localmente e aceder remotamente a processos servidores para executarem as funções mais complexas de manipulação de dados, cálculo, entrada/saída, impressão, etc.

Os servidores devem publicitar a existência dos seus serviços e quais os protocolos de comunicação utilizados. Normalmente, a interface é descrita em termos das funções acessíveis e dos respectivos parâmetros de entrada e de retorno. A identificação do serviço deve ser lógica e não pressupor uma localização física, possibilitando a mudança de máquina do servidor sem necessidade de mudar o código dos seus clientes.

Para diminuir a incidência de algumas das limitações apontadas à utilização das bibliotecas das interfaces de comunicação, procurou-se criar um ambiente de suporte que facilite a programação, introduza uma normalização para as funções de base, dê mais garantias de segurança e tolerância a faltas e procure optimizar o desempenho. Esta evolução tem sido apresentada como objectivo das plataformas Cliente/Servidor, sendo vulgarmente considerados os seguintes serviços:

- comunicação entre clientes e servidores por uma chamada de procedimento remoto;
- linguagem de descrição de interfaces e respectivo ambiente de desenvolvimento;
- a gestão de nomes;
- mecanismos de segurança para autenticação e privacidade das comunicações;
- sistema de ficheiros distribuído;
- sincronização de relógios.

No tocante à comunicação, o modelo de chamada de procedimento remoto corresponde a um ambiente de suporte à programação mais evoluído, utilizando-se uma linguagem de descrição de interfaces a partir da qual se geram, automaticamente, as rotinas de adaptação que são ligadas ao código dos clientes e dos servidores.

O '*software*' sistema é complementado por diversos servidores que oferecem um serviço de gestão de nomes e políticas de segurança. A gestão de nomes é uma peça fundamental num sistema distribuído, dado ser necessária a existência de uma entidade encarregue de registar traduzir os nomes em identificadores utilizados nos diversos protocolos e aplicações.

No domínio da segurança, as plataformas oferecem normalmente uma base computacional de confiança, que disponibilize canais seguros de comunicação e autenticação.

A evolução das plataformas Cliente/Servidor deu-se na direcção dos sistemas distribuídos de objectos, que podem ser considerados como uma síntese de vários conceitos: o modelo Cliente/Servidor, a programação por objectos e as bases de dados orientadas aos objectos. Nestas plataformas, procura-se integrar num modelo de programação uniforme centrado nos objectos, o tratamento da distribuição e da persistência. Apesar de estar ainda longe uma total e transparente integração dos objectos, vários sistemas comerciais já reflectem uma aproximação centrada nos objectos na definição dos mecanismos de intercomunicação entre aplicações remotas (ex.: CORBA). Muitos dos sistemas actuais que necessitam de intercomunicação entre aplicações remotas, utilizam como interface da sua comunicação, '*sockets*'.

B. Sockets

Desde há muito tempo que os sistemas operativos tentam responder ao problema da distribuição. O sistema UNIX teve, desde as suas versões iniciais mecanismos de suporte para a distribuição. A versão de *Berkeley* introduziu em 1982 uma proposta inovadora de interface de comunicação designada por '*socket'*, que é hoje em dia utilizada em larga escala nas plataformas de desenvolvimento de software.

Os '*sockets'* propunham-se dar resposta a alguns requisitos considerados importantes na evolução dos sistemas operativos:

- independência de protocolo: a interface devia ser independente dos protocolos de transporte utilizados;
- transparência: a comunicação não deve depender da localização dos processos, devendo ser idêntica para o caso local ou distribuído;
- compatibilidade: o novo mecanismo devia inserir-se na interface clássica de comunicação.

Os '*sockets'* apresentaram-se como uma evolução dos '*pipes'* para manter a compatibilidade com os programas existentes, utilizando assim, a interface dos '*pipes'*, baseada nos descritores de ficheiros.

Com este inovador interface procurou-se garantir alguma independência entre o modelo de comunicação e o protocolo de transporte, permitindo que os '*sockets'* possam ser utilizados em ambientes baseados em protocolos diferentes. Esta característica materializou-se na noção de domínio.

Exemplos desses domínios são:

- Domínio Internet utiliza os protocolos TCP e UDP;
- Domínio Unix permite a comunicação entre processos numa única máquina, sendo semelhante aos '*pipes'* com nome;
- Domínio Xerox N conjunto de protocolos proposto pela Xerox.

Na criação de um '*socket'*, para além da definição de domínio, é especificado o tipo de serviço pretendido, o que tem essencialmente a ver com a distinção entre o estabelecimento de um canal com ou sem ligação.

C. RPC

Assim pode-se constatar que o modelo Cliente/Servidor pode ser implementado directamente sobre a interface de comunicação designada por '*sockets'*. Contudo, apesar de, em termos das noções habituais de redes, a interface de programação suportada em mensagens seja intuitiva, a sua ligação ao modelo de programação das linguagens procedimentais é delicada, devido à semântica totalmente diferente das interacções. O modelo de mensagens é também pouco transparente, expondo excessivamente muitos dos mecanismos em que o protocolo de comunicação se baseia.

Esta falta de abstracção do protocolo de comunicação levou ao desenvolvimento de uma metodologia de programação distribuída baseada na chamada de um procedimento remoto (RPC – '*Remote Procedure Call*').

A ideia base deste modelo é simples e parte de duas constatações: a chamada de um procedimento é o mecanismo mais conhecido para a transferência de controlo e dados dentro de um programa e a interacção habitual no modelo Cliente/Servidor pressupõe a troca de pelo menos duas mensagens: uma requisitando o serviço e outra de resposta ao pedido, o que permite criar um protocolo eficiente para a suportar.

O RPC consiste numa generalização deste mecanismo que visa permitir uma transferência de controlo e dados entre espaços de endereçamento diferentes. A interacção do tipo pergunta/resposta, implícita no RPC, permite simplificar alguns dos requisitos de fiabilidade dos níveis de comunicação inferiores, tornando possível a utilização de protocolos de transporte sem ligação.

É com base neste conceito inovador que surge uma nova tecnologia, o CORBA ('*Common Object Request Broker Arquitecture*').

D. CORBA

O CORBA é um importante projecto de '*midleware*' realizado por um consórcio chamado OMG ('*Object Managment Group'*). Em paralelo com este projecto, a Microsoft desenvolveu um '*Broker*' próprio chamado DCOM ('*Distributed Component Object Model*'). Estas tecnologias, ao utilizarem um BUS de software para comunicação entre objectos, tornam mais transparente ao utilizador o acesso a estes e aos serviços remotos que estes prestam.

A vantagem do CORBA reside no modo como consegue unificar as aplicações existentes, incluindo-as no BUS (dado que as especificações de um serviço estão separadas da sua implementação).

O CORBA foi projectado para permitir que componentes inteligentes se consigam descobrir uns aos outros e operar

entre si. No entanto, o CORBA é muito mais do que isto, também especifica um conjunto extenso de serviços para criar e apagar objectos; para lhes aceder por nome, para os organizar de um modo particular para além de outros serviços.

Esta arquitectura permite também criar um objecto qualquer e em seguida torná-lo transaccionáveis, seguro e coerente de acordo com o serviço com ele relacionado. Isto significa que um objecto pode ser projectado para realizar uma função qualquer e depois inserir o '*midleware*' correcto quando é construído ou quando é criado em tempo real. É óbvio que esta flexibilidade permitida pelo CORBA oferece múltiplas facilidades na construção de plataformas Cliente/Servidor.

O CORBA, é no fundo uma arquitectura de integração de aplicações que permitem esquecer os sistemas, as linguagens e a rede, com uma arquitectura transparente ao utilizador, compreende uma linguagem de definição de interfaces IDL, compiladores para COBOL, C, C++, Smalltalk e JAVA, e um nível de transporte dos pedidos trocados entre os objectos (ORB).

O ORB ('*Object Request Broker*'), é um '*middleware*' que permite o acesso a serviços distribuídos. O modo de comunicação entre objectos é através do ORB. Um objecto que quer aceder a serviços de outro, recorre à utilização de métodos previamente definidos no interface (IDL). A linguagem utilizada para escrever uma IDL é neutra e define no fundo os protótipos dos serviços prestados. Para que se consiga garantir distribuição, tem de se permitir o acesso de outras linguagens, ferramentas, sistemas operativos e redes.

Os objectos CORBA, são essências inteligentes que vivem em qualquer sítio de uma rede, funcionando como um pacote de informação que pode ser acedida por clientes através da invocação de métodos do objecto (definidos na IDL).

III. ANATOMIA DO CORBA

Para descrever o CORBA, é necessário abordar de um modo mais ou menos geral todas as partes desta arquitectura.

O BOA, o '*Agent*' e o ORB, suportam a manipulação dos objectos, que são a plataforma mais elevada nesta hierarquia. O BOA (Basic Object Adapter) e o '*Agent*' são serviços de que o ORB depende, para que o funcionamento do sistema seja o mais correcto. É também importante perceber que o CORBA oferece interfaces estáticos e dinâmicos para os seus serviços. Para que não existam dúvidas sobre como o sistema está implementado, é feita em seguida uma descrição pormenorizada da arquitectura. É necessário por isso, fazer uma análise do lado do cliente e outra do lado do servidor.

A. Cliente

Analisando em primeira instância o lado do cliente, ilustrado na figura 1, nota-se que:

- Os '*stubs*' do cliente providenciam interfaces estáticos para os serviços do objecto. Estes 'stubs' pré-compilados definem o modo como o cliente pode invocar serviços nos servidores. Na perspectiva do cliente, o '*stub*' actua como uma chamada local. Os serviços estão definidos na IDL; tanto os '*stubs*' do cliente como os do servidor são gerados pelo compilador de IDL. O cliente tem de ter obrigatoriamente um '*stub*' IDL por cada interface que usar no servidor. Este codifica e descodifica a operação e os seus parâmetros, em mensagens com formato específico para que possam ser enviadas ao servidor. Os '*stubs*' incluem também '*header files*' para seja possível invocar métodos do servidor através de linguagem de mais alto nível, tal como, C, C++, Java, ou Smalltalk sem preocupação com os protocolos de transporte. É tão simples como invocar o método para obter os serviços remotos desejados.
- O interface de invocação dinâmica (DLL) permite descobrir e invocar métodos em tempo real. O CORBA define APIs (API - aplicação normalizada) standart que permitem procurar a metadata que define o interface do servidor, gerar os parâmetros, promover chamadas remotas, e receber os resultados
- O depósito de interfaces ('*Interface Repository*'), permite obter e modificar todos os componentes de interfaces registados, os métodos que eles suportam e os parâmetros que eles necessitam. O CORBA designa estas descrições como "assinaturas dos métodos". O depósito de interfaces é uma base de dados em tempo real ('*run time*'), distribuída, que contem versões dos interfaces definidos na IDL. As APIs permitem aos componentes dinâmicos aceder, gravar e actualizar informação. Assim todos os componentes que "vivem" no ORB tem os seus interfaces bem definidos. O ORB é, então, ele próprio, um BUS auto-descritivo.
- O Interface ORB consiste em algumas APIs para serviços locais que tenham interesse para uma aplicação. Por exemplo, o CORBA fornece APIs para converter uma referência para um objecto numa '*string*' e vice-versa. Estas características podem ser úteis se forem necessárias, por exemplo, guardar e comunicar com referências para objectos.

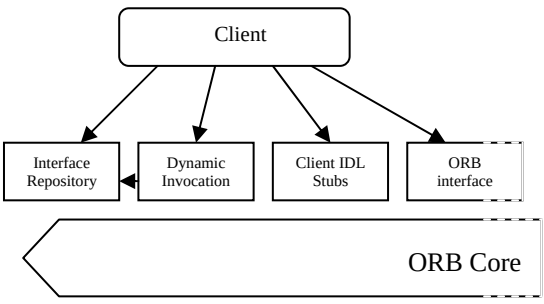

Fig. 1 – Arquitectura do lado do Cliente

O suporte para invocações Cliente/Servidor estáticas e dinâmicas, fornece ao CORBA uma mais valia em relação a outros '*midlewares*' concorrentes. As invocações estáti-

cas são mais fáceis de programar, mais rápidas e auto-documentadas. As invocações dinâmicas oferecem uma flexibilidade máxima, mas são de difícil programação; são muito úteis para as ferramentas que tentam descobrir serviços em tempo real.

B. Servidor

Do lado do servidor é impossível descobrir diferenças entre invocações dinâmicas ou estáticas, dado que, têm ambas a mesma semântica de mensagem, como se pode verificar na figura 2.

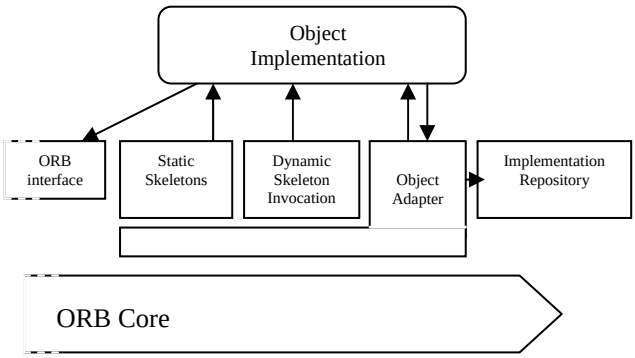

Fig. 2 – Arquitectura do lado do Servidor

Em ambos os casos, o ORB localiza um adaptador de objecto servidor, transmite os parâmetros e transfere o controlo para a implementação do objecto através do '*stub'* IDL do servidor. Em seguida, descrevem-se as funções dos elementos CORBA, do lado do servidor:

- Os '*stubs'* IDL do servidor (o que a OMG designa por esqueletos) providenciam interfaces estáticos para cada serviço exportado pelo servidor. Estes '*stubs'*, assim como os do cliente, são criados pelo compilador de IDLs.
- O interface de esqueletos dinâmico (DSI – '*Dynamic Skeleton Interface'*) possui um mecanismo de ligação em tempo real ('*run-time*') para servidores que precisem de pedidos de acesso a métodos, que não sejam baseados em esqueletos (ou '*stubs'*) gerados a partir de IDLs. Os esqueletos dinâmicos verificam os parâmetros de entrada das mensagens para determinar o seu destino, ou melhor, o objecto destino e o método correspondente. Em contraste, os esqueletos compilados são normalmente definidos para um objecto particular, em que, a cada método está associado um código de implementação, de acordo com a IDL. Os esqueletos dinâmicos são muito úteis para a implementação de "pontes" entre ORBs. Eles podem também, ser usados para gerar dinamicamente as implementações dos métodos dos objectos. Uma DSI é, no servidor, equivalente a uma DLL no cliente. Este interface pode receber invocações dinâmicas e estáticas por parte dos clientes.
- O adaptador de objectos ('*Object Adapter*') é a parte principal dos serviços de comunicação, de recepção e aceitação de pedidos de serviços dos objectos do servidor. Ele fornece um ambiente em tempo real ('*run-time'*) para a inicialização de objectos do servidor, fazendo-lhes um pedido e atribuindo-lhes uma identificação (ID - o CORBA chama a estes IDs referência a objectos). O adaptador de objectos, regista as classes que suporta e as suas instancias em tempo real (objectos) no depósito de implementações. O CORBA especifica que cada ORB tem que suportar um adaptador standart chamado '*Basic Object Adapter'* (BOA). Os servidores podem suportar mais do que um adaptador de objectos.
- O depósito de implementações ('*Implementation Repository'*) contém uma base de dados em tempo real ('*run-time*') com informação acerca das classes que o servidor suporta, os objectos que estão inicializados e os seus IDs. Serve também como local para guardar informação adicional associada com a implementação de ORBs.
- O interface do ORB consiste em algumas APIs para serviços locais que são idênticos aos existentes do lado do cliente.

De um modo simples, pode-se descrever o ORB como uma entidade viva na rede (o referido "algo " na pág. 3) que actua como intermediário entre aplicações; é um manipulador de informação, que presta serviços aos clientes; é um interpretador de pedidos e um gerador de soluções. Pode-se dizer, em forma de resumo, que é uma entidade globaliza a particulariza a informação ao mesmo tempo, tornando a tarefa comunicação mais fácil e ausente de qualquer programador.

Concluímos assim, a nossa breve descrição dos componentes de um ORB e dos seus interfaces. Falta neste momento, explicar uma parte desta arquitectura, o '*Agent*'.

C. Agent

O '*Agent*' é uma extensão ao '*Basic Object Adapter'* definido pelo CORBA. É uma directoria de serviço dinâmica e distribuída, que oferece facilidades para as aplicações do cliente e para as implementações do objectos. Múltiplos '*Agents'* numa rede, cooperam para oferecer um acesso fácil do cliente aos objectos do servidor. O '*Agent'* monitoriza os objectos que estão na rede, e localiza-os para as aplicações cliente, em tempo real ('run-time'). O '*Agent'* pode determinar se uma ligação da aplicação cliente ao objecto se perdeu, devido a um erro, como por exemplo, o servidor falhou, ou houve alguma quebra na rede. Quando uma falha é detectada, é imediatamente efectuada uma tentativa de ligar o cliente a outro servidor localizado num outro qualquer endereço.

Para que uma ligação via CORBA se possa efectuar é necessário que pelo menos um '*Agent'* esteja inicializado em algum local da rede. Quando um cliente se pretende conectar a um objecto, invocando o método _bind(), o '*Agent'* é automaticamente consultado. Este, localiza a implementação especificada, para que a conecção se possa estabelecer.

A comunicação com o '*Agent*' é completamente transparente ao programa cliente. Quando a implementação de

um objecto chama as funções BOA::obj_is_ready() ou BOA::impl_is_ready(), o '*Agent*' regista o objecto ou a implementação para que possa ser usado por aplicações cliente. Quando um objecto é desactivado, o '*Agent*' remove o objecto da lista de objectos activos. Assim, como em programas cliente, a comunicação com o '*Agent*' é completamente transparente da implementação do objecto.

D. Objectos CORBA

Um objecto CORBA, tem três características principais:

- encapsulamento (Encapsulation)
- herança (Inheritance)
- polimorfismo (Polymorphism)

(i) Encapsulamento

Um objecto CORBA é um módulo de software encapsulado que consiste em duas partes distintas; o seu interface, que apresenta o módulo ao mundo exterior e a sua implementação que é mantida em privado.

Esta característica, permite software plug&play, ou seja, quando é feita uma ligação ao objecto, podem-se imediatamente, invocar os seus métodos, devido à transparência do seu interface. Este, mostra-nos as funcionalidades dos objectos, de modo que quando um cliente invocar uma delas, o objecto lhe responda com uma informação num formato previamente acordado.

O modo como o objecto trata a informação que lhe é enviada e também como a resposta é processada, faz parte da implementação e são dados que não podem ser acedidos do mundo exterior.

Nos objectos pode existir mais do que uma implementação para a mesma interface. A implementação do código pode ser feita numa linguagem diferente. Para os clientes do objecto, esta operação é perfeitamente transparente, desde que a resposta seja devolvida no formato preestabelecido. Pode-se assim integrar num mesmo sistema vários clientes para um mesmo objecto, ainda que utilizem linguagens distintas.

(ii) Herança.

A herança é uma propriedade que facilita a reutilização das classes. As relações de herança permitem especializar, redefinir ou acrescentar funcionalidades a uma classe já existente. As classes podem ser organizadas em relações hierárquicas que permitem a herança da especificação e/ou da implementação das classes antecessoras ou, como é vulgar serem designadas, das superclasses. As relações entre as classes podem ser de herança simples ou múltipla, em que, uma classe herda de múltiplas superclasses.

A herança é um dos conceitos mais complexos nos modelos de objectos. Nela pode-se herdar só as interfaces ou só a implementação.

A herança de interfaces implica que o objecto herda a definição da interface das superclasses, mas não a sua implementação. A implementação pode ser efectuada escrevendo o respectivo código na nova classe ou redireccionando as invocações a métodos desta interface para outros objectos.

No caso de se herdar a implementação, a nova classe fica indissociavelmente ligada ao código e às estruturas de dados da ou das superclasses suas antecessoras, ou seja, a nova classe é construída adicionando novas funções e novas variáveis à estrutura já existente. Qualquer mudança nas classes antecessoras, tem implicações na classe que herda a implementação, obrigando a recompilá-la. O contrato estabelecido entre a classe base e as suas derivadas não obedece a uma definição clara, podendo o sistema ter comportamentos indefinidos se a classe base for modificada, sem que as classes que herdam a interface tenham hipótese de conhecer a alteração.

(iii) Polimorfismo

Quando é efectuada uma invocação, o objecto invocado pode não ser da classe que o invocador espera, mas desde que consiga responder à mensagem de invocação pode actuar, como se de outro se tratasse. O polimorfismo, permite redireccionar a invocação para outro tipo de objectos, sem ter de modificar o pedido dos clientes

Como usar um Objecto CORBA?

O objecto Cliente, activa um método na direcção do '*stub*' correspondente ao serviço procurado; este por sua vez, transmite-o ao ORB que interroga o Depósito de interfaces de forma a localizar o objecto em questão. Em seguida, estabelece a ligação ao objecto por intermédio do adaptador de objectos (BOA) e do esqueleto.

O ORB cria uma imagem chamada proxy desse objecto servidor.

O objecto cliente, só irá dialogar com o proxy por intermédio de métodos, mesmo que este continue a efectuar de forma transparente pedidos junto do objecto Servidor. Nota-se que a distribuição entre cliente e servidor é puramente pedagógica, as relações entre os objectos são potencialmente simétricas.

Podemos então afirmar, que se trata de uma estrutura Cliente/Servidor em que o Cliente utiliza de uma forma simples os recursos do objecto disponível sem se preocupar com as características do sistema, da rede, ou da linguagem. Todo o serviço é prestado pelo servidor através do objecto, sem que o Cliente se aperceba disso. Aqui se mostra o aspecto transparente da arquitectura CORBA. O grau de abstracção do utilizador é enorme em relação a outro qualquer tipo de arquitectura de comunicações.

IV. O PROJECTO

Com este projecto, pretendeu-se construir um módulo de comunicações CORBA para suportar o sistema de Televigilância, de modo que substituindo o núcleo de comunicações se mantivessem todas as funcionalidades do sistema.

O projecto existente, SIVIL II, foi desenvolvido pelo INESC em cooperação com o CET e financiado pela PT. É uma aplicação com uma estrutura Cliente/Servidor que

corre em ambiente Windows 95 e foi implementado na plataforma de trabalho, Visual C++. As comunicações são suportadas através do protocolo TCP/IP numa rede RDIS. Toda a transferência de informação entre estações é feita usando Sockets.

O novo módulo de comunicações suporta toda a transferência de informação entre o objecto Cliente e o objecto Servidor de uma forma equivalente ao actual, podendo as aplicações manter as suas funcionalidades. Ou seja, todo o sistema de transferência de informação foi alterado.

A. O novo módulo de comunicações

No novo módulo, a transferência de informação, não usa mensagens, tornando-se mais simples. Esta simplicidade advém do facto de que a informação a transmitir, já não tem de ser organizada em estruturas, pode agora, ser apenas, parâmetros de entrada de funções de um objecto.

Todo o reportório de mensagens utilizado foi substituído por funções que permitissem a transferência da mesma informação.

Foi necessário, portanto, implementar métodos nos novos objectos, de modo que a transferência de informação fosse garantida.

Num sistema que utilize mensagens como base para as comunicações, é típico, o uso de um cabeçalho com a informação em attachement. Esta, tem de ser organizada de uma forma específica de modo que se consiga uma leitura e escrita coerente.

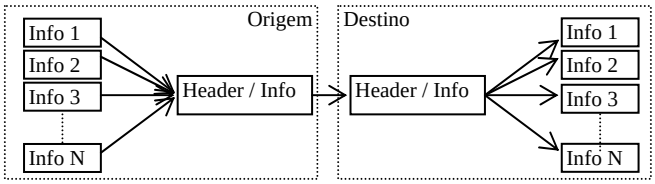

Fig. 3 – Modo de comunicação original

No código antigo, toda a informação era encapsulada e transmitida com uma única mensagem de notificação; a informação teria então de ser descapsulada e utilizada.

No CORBA, este processo é transparente (lateral), dado que, ao introduzir a informação como parâmetros de entrada/saída de uma função, a organização é inerente à própria operação.

Teve de ser feita uma equivalência quase unívoca entre uma mensagem (antes) e uma função específica (agora).

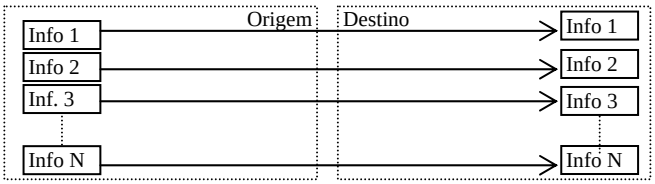

Fig. 4 – Modo de comunicação via CORBA

Devido às facilidades do CORBA, associámos a cada implementação dos métodos uma mensagem Windows, a qual é utilizada para notificar a sua janela de diálogo.

V. APRECIAÇÕES GLOBAIS

O CORBA é um esforço de normalização de múltiplas empresas reunidas num consórcio designado OMG (‘*Object Management Group’*), que iniciou em 1989 a definição de um ambicioso projecto de normalização, que procura criar um ambiente genérico de desenvolvimento de aplicações baseadas em objectos.

No CORBA, a definição das classes é efectuada através de uma linguagem de definição de interfaces (IDL). A IDL é puramente declarativa e corresponde a um subconjunto do C++, a que foram adicionadas algumas palavras chave para suportar os conceitos associados à distribuição. No CORBA estão previstos mecanismos de ligação às linguagens de programação mais utilizadas. A linguagem de definição das interfaces suporta herança múltipla. Como a IDL é só declarativa, a herança diz respeito às interfaces e não à implementação, que é interna aos objectos.

A peça central da arquitectura é o ORB (‘*Object Request Broker’*) que medeia o mecanismo de invocação entre objectos. Na literatura do CORBA, o ORB aparece descrito como ‘Bus’ a que se ligam as componentes de software para interactuarem entre si.

As facilidades mais importantes do ORB são as seguintes:

- Invocação estática e dinâmica de objectos;
- Independência da linguagem de alto nível, suportando os múltiplos formatos de invocação das linguagens mais utilizadas;
- Inventário das interfaces disponíveis (‘*Interface Repository’*); é um serviço de registo de interfaces que permite dinamicamente conhecer métodos, parâmetros, etc;
- Mensagens polimórficas;
- Segurança;
- Suporte para ligação a monitores transaccionais.

A invocação estática é semelhante à invocação de um RPC com uma rotina de adaptação previamente criada. Na invocação dinâmica, a mensagem de invocação é produzida no momento da invocação, a partir da informação existente sobre o método. O *depósito de interfaces* contém as assinaturas dos métodos (assinatura no sentido de descrição dos parâmetros) das componentes que se encontram acessíveis no BUS.

A invocação ser estática ou dinâmica tem apenas a ver com o cliente. No caso do cliente conhecer todo o detalhe da invocação na altura da escrita do código, a invocação estática permite um melhor controlo da verificação de tipos e optimiza o desempenho. Na invocação dinâmica, o cliente pergunta ao depósito de interfaces pela descrição do método e constrói, com base nessa descrição, uma lista de parâmetros que serão utilizados na invocação.

No lado do servidor, temos elementos recíprocos: as rotinas de adaptação definidas estaticamente (*static skeleton*) e o suporte para as invocações dinâmicas (dynamic skeleton). No servidor mantém-se a separação

entre as interfaces e as implementações dos objectos que não foram escritos em linguagem de objectos.

O Object Adapter é o ambiente de suporte do servidor que permite a instanciação e a invocação dos objectos. A sua ligação ao depósito de implementações, permite-lhe carregar o código das classes para instanciar os respectivos objectos.

O Object adapter corresponde ao suporte à execução dos servidores de RPC. Quando recebe uma mensagem, deve entregá-la a rotina de adaptação (‘*skeleton’*) corresponde para que esta efectue o tratamento dos parâmetros e os coloque numa forma aceitável pelo código que implementa o objecto. O esqueleto efectua então a chamada (Upcall) ao código que implementa o método.

O CORBA especifica vários tipos de Object Adapter e define um Basic Object Adapter (BOA) que deverá existir em todas as implementações do ORB.

Comparando a arquitectura CORBA com uma arquitectura que utiliza ’*Sockets*’ podemos afirmar, que o nível de abstracção oferecido pelo CORBA é indiscutivelmente superior, devido, não só à facilidade de invocações dinâmicas mas também ao facto de que podemos manter referências a objectos sem nos preocuparmos com o estabelecimento de nova comunicação. Esta abstracção na programação é também conseguida devido à possibilidade de invocar directamente os métodos dos objectos.

A Visigenic disponibiliza um exemplo em que se compara a utilização de ’*Sockets*’ com a utilização de objectos CORBA. Os resultados são apresentados nos gráficos seguintes:

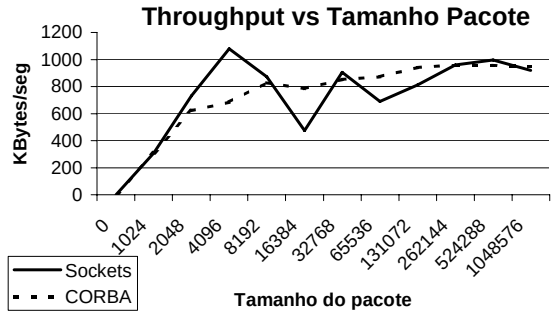

Fig. 5 – Variação do *throuput* com o tamanho do pacote

Estes demonstram que, em pacotes mais pequenos, o CORBA é menos eficiente do ponto de vista do número de ‘*bytes*’ de informação transferidos por segundo (‘*throughtput’*), no entanto é equivalente aos ’*sockets*’ para pacotes maiores, o que sugere uma estratégia de maximização do tamanho dos pacotes.

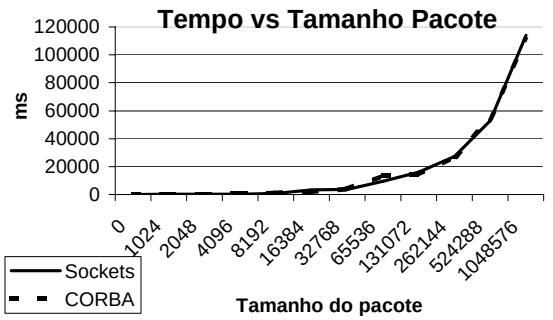

Fig. 6 – Variação do tempo de transmissão com o tamanho do pacote

Observando o gráfico tempo/tamanho pacote, podemos afirmar que o CORBA tem um desempenho igual em toda a gama de tamanho de pacotes.

A. Comentários finais

Em todos os aspectos focados, o CORBA não desiludiu. Mesmo sendo uma tecnologia em desenvolvimento, o pacote de software disponibilizado não apresentou falhas de maior.

Em relação às vantagens e desvantagens do CORBA, o texto elucida de um modo geral todas elas, podendo destacar a transparência e facilidade da programação, a acessibilidade a recursos remotos e a poderosa arquitectura simétrica Cliente/Servidor que apresenta.

Concluindo, podemos afirmar que as tecnologias que utilizam objectos distribuídos, entre elas o CORBA são o futuro das redes de informação e das suas comunicações.

VI. BIBLIOGRAFIA

- Robert Orfali, Dan Harkey, Jeri Edwards, *“Instant CORBA”*, Wiley Computer Publications, 1997
- Jon Sieget, Ph. D., “C*ORBA Fundamentals and Programing”*, Wiley Computer Publications Group, 1996
- José Alves Marques, Paulo Guedes, *“Tecnologia de sistemas distribuídos”*, FCA Colecção Tecnologias de Informação, 1998
- Fah-Chung Cheong, “*Internet Agents”*, New Riders
- Visigenic Software, *“Visigenic Programmer’s Guide version 2.0”*, Visigenic Software
- Visigenic Software, *“Visigenic Programmer’s Guide version 3.0”*, Visigenic Software
- Bjorn Stroustrup, “*The C++ Programing Language”*, Addison Wesley
- David Bennet, “*Visual C++ 5.0”*, Sams Publications
- Ivor Horton, “*Visual C++ 4.2”*, Wrox Press Ltd.