MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-8602-TDI/787

UMA ARQUITETURA FLEXÍVEL E DINÂMICA PARA OBJETOS DISTRIBUÍDOS APLICADA AO SOFTWARE DE CONTROLE DE SATÉLITES

Maurício Gonçalves Vieira Ferreira

Tese de Doutorado em Computação Aplicada, orientada pelos Drs. Tatuo Nakanishi e João Bosco Schumam Cunha, apresentada em 23 de março de 2001.

INPE
São José dos Campos
2002

681.3.06

FERREIRA, M. G. V.
Uma arquitetura flexível e dinâmica para objetos distribuídos aplicada ao software de controle de satélites/
M. G. V. Ferreira – São José dos Campos: INPE, 2001.
244p. – (INPE-8602-TDI/787).

1.Objetos distribuídos. 2.Flexibilidade. 3.Distribuição de carga. 4.Programação dinâmica. 5.Mobilidade. I.Título.

Aprovado pela Banca Examinadora em cumprimento a requisito exigido para a obtenção do Título de **Doutor** em **Computação Aplicada.**

Dr. Solon Venâncio de Carvalho
Presidente/INPE-SJC/SP

Dr. Tatuo Nakanishi
Orientador/UMC-Mogi das Cruzes/SP

Dr. João Bosco Schumann Cunha
Orientador/UMC-Mogi das Cruzes/SP

Dr. Carlos Ho Shih Ning
Membro da Banca

Drª Selma Shin Shimizu Melnikoff
Membro da Banca
Convidada POLI/USP-São Paulo/SP

Dr. Edmundo Roberto Mauro Madeira
Membro da Banca
Convidado UNICAMP-Campinas/SP

Candidato (a): Maurício Gonçalves Vieira Ferreira

São José dos Campos, 23 de março de 2001.

Ao Mateus e a Kátia

AGRADECIMENTOS

Ao Dr. Tatuo Nakanishi e ao Dr. João Bosco S. Cunha que têm contribuído de forma significativa pela minha formação acadêmica, e por ter assumido mais este desafio de modo seguro compreensivo e amigo.

A meus pais, Francisco e Maria da Glória pelo amor e sacrifício.

A minha esposa Kátia e meu filho Mateus pela compreensão, apoio e amor.

RESUMO

O Instituto Nacional de Pesquisas Espaciais (INPE), desde sua criação, tem concentrado esforços no desenvolvimento tecnológico-espacial. Na década de 80 foi criado o primeiro projeto que previa o lançamento de quatro satélites brasileiros dentro da Missão Espacial Completa Brasileira (MECB). Parte dessa meta já foi alcançada com o lançamento dos satélites SCD1 (Sistema de Coleta de Dados 1), em 1993, e SCD2 (Sistema de Coleta de Dados 2) em 1998. A complexidade inerente à qualquer missão espacial, aliada à evolução tecnológica empregada em cada novo satélite construído, propicia o desenvolvimento de aplicativos cada vez mais complexos, empregados no controle da missão. O avanço da eletrônica tem aprimorado o desenvolvimento de satélites, e o surgimento de novas tecnologias de desenvolvimento de software tem contribuído para a criação de aplicativos cada vez mais robustos e flexíveis. A tecnologia de distribuição pode ser um exemplo desta evolução tecnológica e pode contribuir de forma significativa na maneira de se projetar, desenvolver e manter sistemas de informações corporativas. Enfim, as mudanças tecnológicas surgidas no desenvolvimento de software podem ser agregadas aos novos aplicativos desenvolvidos para controle de satélites, portanto, sem nenhum demérito, o desenvolvimento de software, tem cada vez mais, assumido o papel de coadjuvante para a alavancagem da tecnologia espacial. Os aplicativos de solo e bordo utilizados no controle e monitoração de um satélite deverão ser flexíveis e robustos para se adaptar à evolução tecnológica desta área. Seguindo os rumos destas inovações tecnológicas, tanto na área espacial quanto na área de desenvolvimento de softwares, o presente trabalho de pesquisa propõe "*Uma arquitetura flexível e dinâmica para objetos distribuídos aplicada ao Software de Controle de Satélites.*"

A DYNAMIC AND FLEXIBLE ARCHITECTURE FOR DISTRIBUTED OBJECTS APPLIED TO SATELLITES CONTROL SOFTWARE

ABSTRACT

The National Institute for Space Research (INPE) since its establishment has concentrated its efforts on the spacial technological development. The first project that would foresee the launching of four brazilian satellites under the Brasilian Total Spacial Mission, was created in the eighties. Part of that goal has already been achieved from the launching of the following satellites: Data Collecting System 1 (DCS 1), in 1993 and Data Collecting System 2 (DCS 2) in 1998. The complexity inherent in any spacial mission together with the technological development used in each new satellite manufacturing, favors the development of more and more complex applications used to control the mission. The electronics progress has improved satellites development as well as the new technologies of software development have contributed to create applications mode and more robust and flexible. The distribution technology may be a sample of this technological development and may contribute significantly towards projecting, developing and maintaining corporate information systems. Thus, the technological changes that have appeared in software development can be joined to the new applications developed to control satellites, so, the software development has more and more cooperated to the spacial technology improvement, the board and ground applications used to control and monitor a satellite must be flexible and robust to adapt themselves to this area technological development. According to the direction of these technological innovations, both in the spacial area and the software development one, this research job proposes “A flexible and dynamic architecture to distributed objects applied to satellites control software.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

ALA	- Software Gerenciador de Eventos
AM	- Agentes Mensageiros
AMO	- Agentes Monitores de Objetos
AO	- Adaptador de Objeto
API	- Application Programming Interface
BD	- Banco de Dados
BDOO	- Banco de Dados Orientado a Objetos
BDR	- Banco de Dados Relacional
BOA	- Adaptador de Objeto Básico
CAS	- Software de Controle de Acesso
CBERS	- China Brazil Earth Resource Satellite
CCS	- Centro de Controle de Satélites
CET	- Software Operacional da Estação Terrena
CLSID	- Class Identifier
CMCD	- Centro de Missão de Coleta de Dados
CMD	- Software de Telecomandos
COM	- Component Object Model
CONSAT	- Controlador de Satélites
CORBA	- Common Object Request Broker Architecture
DA	- Direct Attribute
DCOM	- Distributed Component Object Model
DDL	- Data Definition Language
DDO	- Dynamic Data Object
DII	- Dynamic Invocation Interface
DIRMIS	- Diretor de Missão
DIROP	- Diretor de Operações

DLL - Dynamic Link Library
DT - Distributed Technology
DTM - Software de Visualização de Telemetria
DOT - Tecnologia de Objetos Distribuídos
ENSAT - Engenheiro de Satélites
ET - Estação Terrena
FLOI - Fase de Lançamento de Órbitas Iniciais
GAN - Software Gerenciador da Antena
GRF - Software de Representação Gráfica
HIS - Software de Manutenção de Arquivos Históricos
IA - Inteligência Artificial
IDL - Interface Definition Language
IID - Identificador da interface
IIOP - Inter ORB Protocol
IP - Internet Protocol
INPE - Instituto Nacional de Pesquisas Espaciais
ISO - International Standards Organization
JDK - Java Development Kit
JVM - Java Virtual Machine
MECB - Missão Espacial Completa Brasileira
MON - Software de Monitoração da Estação Terrena
NMS - Network Management Systems
ODL - Linguagem de Definição de Objetos
ODP - Object Distributed Process
OQL - Linguagem de Consulta a Objetos
ODMG - Object Database Management Group
OMA - Object Management Architecture
OMG - Object Management Group
OPS - Software de Operação do Segmento Solo
ORB - Object Request Broker
OSF DCE - Open Software Foundation's Distributed Computing Environment

OSI - Open System Interconnect

OT - Object Technology

PC - Personal Computer

PDS - Persistent Data Services

PID - Identificador de Persistência

PO - Pesistent Objects

POM - Persistent Object Manager

POS - Serviço de Persistência para Objetos

PRE - Software de Preparação de Arquivos e Tabelas

PSDL - Persistent State Definition Language

PSS - Persistent State Service

RAM - Random Access Memory

RAN - Software de Processamento de Medidas de Distância

RECDAS - Rede de Comunicação de Dados

RESOFT - Responsável pelo Software

RPC - Remote Procedure Call

REL - Software de Inicialização e Atualização do Relógio

RMI - Remote Method Invocation

SCD1 - Sistema de Coleta de Dados 1

SCD2 - Sistema de Coleta de Dados 2

SCO - Software de Comunicação

SCS - Software Operacional do CCS

SGBDD - Sistema Gerenciador de Banco de Dados Distribuído

SICS - Sistema de Controle de Satélites

SICSD - Uma arquitetura flexível e dinâmica para objetos distribuídos aplicada ao Software de Controle de Satélites

SIMULADOR- Software responsável pela simulação de um satélite

SQL - Structure Query Language

STA - Software de Transferência de Arquivos

SUP - Software Supervisor da Estação Terrena

TCP/IP - Transmission Control Protocol/Internet Protocol

TMS	- Software de Processamento de Telemetria
UML	- Unified Modelling Language
UDP	- User Datagram Protocol
WT	- Web Technology

CAPÍTULO 1

INTRODUÇÃO

O que as rede de ferrovias, rodovias e canais foram em outra era, as redes de telecomunicações, informações e computação... são hoje em dia. (Bruno Kreisky- chanceler austríaco)

O Instituto Nacional de Pesquisas Espaciais (INPE), desde sua criação, tem concentrado esforços no desenvolvimento tecnológico-espacial. Na década de 80 foi criado o primeiro projeto que previa o lançamento de quatro satélites brasileiros dentro da Missão Espacial Completa Brasileira (MECB). Parte dessa meta já foi alcançada com o lançamento dos satélites SCD1 (Sistema de Coleta de Dados 1), em 1993, e SCD2 (Sistema de Coleta de Dados 2) em 1998. Lançados no espaço por foguetes, esses satélites giram em torno da terra numa velocidade de aproximadamente 28.000 km/h e se encontram numa altitude em torno de 750 km. A missão desses satélites é retransmitir para a terra os sinais emitidos por plataformas de coleta de dados espalhadas por todo o País. As plataformas coletam informações meteorológicas (pressão atmosferica, velocidade do vento etc.) e transmitem esses dados para o satélite em forma de ondas UHF. O satélite é equipado com um "transponder" que tem a capacidade de converter os sinais de UHF para a banda S e retransmiti-los novamente em direção a terra, onde serão coletados por estações terrenas localizadas em Cuiabá (MT) e Alcântara (MA).

Além dos dados colhidos pelas plataformas, os satélites SCD1 e SCD2 também retransmitem para as estações informações que auxiliam na monitoração de seu funcionamento interno(telemetrias). As telemetrias descrevem, com detalhes, o estado interno do satélite, como por exemplo, a voltagem da bateria, a temperatura dos painéis solares, o posicionamento de determinada chave (ligado/desligado) etc..

O sistema solo destes satélites é constituído pelas estações de Cuiabá, Alcântara e pelo centro de operações, denominado Centro de Controle de Satélites (CCS), situado em São José dos Campos - SP. O CCS é o cérebro de todo o sistema solo. Sua função principal é assegurar o funcionamento nominal do satélite desde sua injeção em órbita

até o fim de sua vida útil. Portanto, o CCS assume o papel fundamental na monitoração e controle dos satélites SCD1 e SCD2.

O CCS é um complexo computacional de alta disponibilidade, exigida pela sua função e assegurada por um software aplicativo (SICS - Sistema de Controle de Satélites), utilizado no controle e monitoramento dos satélites SCD1 e SCD2.

O processo de monitoração caracteriza-se pelo acompanhamento de todas as telemetrias recebidas do satélite, enquanto que o processo de controle permite telecomandar o satélite do solo. Ligar/desligar uma determinada chave, modificar a atitude de um satélite são exemplos de telecomandos que podem ser enviados do CCS para um satélite, através das estações e da rede de computadores que os conectam. O sucesso ou não da execução de um telecomando é expresso na telemetria, já que ela descreve o estado interno de um satélite.

A complexidade inerente à qualquer missão espacial, aliada à evolução tecnológica empregada em cada novo satélite construído, propicia o desenvolvimento de aplicativos cada vez mais complexos, empregados no controle da missão. O avanço da eletrônica tem aprimorado o desenvolvimento de satélites, e o surgimento de novas tecnologias de desenvolvimento de software tem contribuído para a criação de aplicativos cada vez mais robustos e flexíveis.

Os últimos anos foram marcados por significativas mudanças na maneira de projetar, desenvolver e manter sistemas de informações corporativas. Tal período iniciou com o nascimento das aplicações monolíticas, onde os programas eram desenvolvidos em um terminal ligado a um mainframe. O SICS foi projetado nesse período, portanto, agrega características de uma aplicação monolítica. Esses ambientes, por razões de ineficiência e alto custo de manutenção, foram superados pela computação client-server. Esse modelo prometia facilitar o desenvolvimento e a manutenção de aplicações complexas, descentralizando os sistemas monolíticos em componentes. Atualmente, a tecnologia de Objetos Distribuídos divide aplicações “client-server” em componentes ou objetos autogerenciáveis, que podem interoperar mesmo com sistemas operacionais e redes heterogêneos, introduzindo vantagens adicionais como aumento de disponibilidade e flexibilidade.

A Tecnologia de Objetos Distribuídos (DOT) abrange três vertentes que, sinergicamente unidas, resultaram em um conceito maior do que a soma de suas partes. Em ordem de emergência, despontaram: Tecnologia de Objetos -> Tecnologia de Distribuição -> Tecnologia Web.

A **Tecnologia de Objetos** ("Object Technology" - **OT**) foi introduzida no fim dos anos 70 com uma linguagem chamada Smalltalk. No modelo orientado a objetos, sistemas são visualizados como objetos cooperantes, que encapsulam estrutura e comportamento, pertencentes a classes hierarquicamente construídas (Purao,Jain e Nazareth, 1998).

A orientação a objetos modificou a maneira de construir e manter sistemas. Existem ferramentas e experiências substanciais para Análise Orientada a Objetos, Projeto Orientado a Objetos e Programação Orientada a Objetos (Jacobson,1992). A transição de modelos e abordagens estruturalmente tradicionais não se completou, nem a OT tem sido totalmente explorada, especialmente nos sistemas legados, como por exemplo o SICS, que precedem os objetos.

A **Tecnologia de Distribuição** (ou "Distributed Technology" - **DT**) , a qual envolve computadores autônomos, não compartilhando memória física, conectados através de uma rede, data do início dos anos 80. O advento de "CPUs" menores, mais poderosas e baratas precipitou o interesse de migrar os aplicativos dos "mainframes" para computadores pessoais e estações de trabalho e, ao mesmo tempo, distribuiu funcionalidade pelos múltiplos computadores interligados.

Com o passar do tempo, a noção de distribuição modificou-se a partir de ambientes rígidos e geograficamente restritos, com máquinas homogêneas para estruturas flexíveis, robustas e geograficamente ilimitadas, com máquinas heterogêneas. Os sistemas distribuídos tradicionais empregavam o modelo cliente-servidor, freqüentemente implementado com os mecanismos de "Remote Procedure Calling"(RPC)-(Vondrak,1998).

Os modelos atuais de distribuição evoluíram a partir da arquitetura tradicional, passando a utilizar a tecnologia "middleware" (Bersntein,1996). Enquanto os

detalhes e definições desse conceito variam consideravelmente, os produtos "middleware" geralmente fornecem uma infra-estrutura "runtime" de serviços para a interoperação de componentes (objetos). Vale registrar o foco dispensado pela DT em revestir o ambiente computacional com a transparência de localização entre os computadores e objetos.

A **Web Technology** (**WT**) nasceu nos anos 90 e rapidamente causou uma explosão no uso da Internet (Etzioni, 1996). A WT universalizou e globalizou previamente a computação de um modo nunca visto. A Internet se apresenta como uma das tecnologias mais populares dos últimos tempos, vislumbrando a possibilidade de criar páginas de informação, que podem ser acessadas em qualquer ponto do planeta, com uma interface humana atraente e descomplicada.

A atividade-fim do INPE se concentra na exploração espacial. Isto se reflete nos futuros projetos de satélites que, cada vez mais, incorporam novas funcionalidades para permitir um maior conhecimento do espaço e da terra. Estas novas funcionalidades tornam cada vez mais complexos o satélite e o software utilizado no seu controle.

As mudanças tecnológicas surgidas no desenvolvimento de software, descritas nos parágrafos anteriores, podem ser agregadas aos novos aplicativos desenvolvidos para controle de satélites, portanto, sem nenhum demérito, o desenvolvimento de software, tem, cada vez mais, assumido o papel de coadjuvante para a alavancagem da tecnologia espacial. Os aplicativos de solo e bordo utilizados no controle e monitoração de um satélite deverão ser flexíveis e robustos para se adaptar à evolução tecnológica desta área.

Assim, seguindo os rumos destas inovações tecnológicas, tanto na área espacial quanto na área de desenvolvimento de softwares, o presente trabalho de pesquisa propõe ***Uma arquitetura flexível e dinâmica para objetos distribuídos aplicada ao Software de Controle de Satélites.***

O Capítulo 2 apresenta alguns conceitos importantes e peculiares aos sistemas distribuídos, enquanto o Capítulo 3 explora as principais arquiteturas existentes no

mercado, que implementam os conceitos de distribuição. A Tecnologia de Agentes, tema central do Capítulo 4, introduz alguns conceitos intrínsecos desta tecnologia, já que a arquitetura proposta se apóia nos recursos propiciados pela utilização dos agentes no desenvolvimento de software. O Capítulo 5 faz um panorama da história do desenvolvimento de software para controle de satélites no INPE, bem como um levantamento das limitações atuais destes aplicativos. O cerne do trabalho e a descrição sucinta do funcionamento da arquitetura proposta encontram-se no Capítulo 6. Este capítulo faz um esboço da arquitetura proposta e os capítulos seguintes, um detalhamento desta arquitetura. O Capítulo 7 descreve os serviços que os agentes devem prestar para concretizar o objetivo da arquitetura proposta. O Capítulo 8 apresenta, com maior detalhes, as características de flexibilidade e dinamismo contidas na arquitetura. Ele faz uma explanação do serviço de balanceamento, criado para atender a estes requisitos da arquitetura. O serviço de persistência, criado para atender aos requisitos dos objetos persistentes e dinâmicos de uma aplicação para controle de satélites, encontra-se descrito no Capítulo 9. Os mecanismos de segurança, bem como as políticas que podem ser criadas para disponibilizar um ambiente confiável e seguro para a arquitetura proposta são elucidados no Capítulo 10. O Capítulo 11 contém os detalhes da implementação prática de um ambiente para o protótipo da arquitetura proposta. As conclusões e considerações finais deste trabalho de pesquisa são delineadas no Capítulo 12.

CAPÍTULO 2

SISTEMAS DISTRIBUÍDOS BASEADOS EM OBJETOS

Prossiga – continue andando (Thomas Morton)

Como já foi mencionado no capítulo anterior, a tecnologia de distribuição de objetos (Distributed Object Technology – DOT) pode ser definida, de forma ampla, como a agregação de três tecnologias sinergicamente acopladas, a saber: Tecnologia de Objetos, Tecnologia de Distribuição e Tecnologia Web.

A combinação do uso destas tecnologias mudou, de maneira fundamental, a forma como esses sistemas são construídos. Objetos são adicionados na redes e representam unidades de distribuição, movimento e comunicação. A tecnologia Web pode ser utilizada para prover portabilidade para a aplicação. Existe, com efeito, um novo paradigma em computação, cujo foco é a interoperabilidade de objetos (Wallnau, 1998), entendendo-se por interoperabilidade a possibilidade de um programa, em um sistema, acessar programas e dados em outros sistemas (Bernstein, 1996). A tecnologia de distribuição de objetos dá a oportunidade de distribuir e globalizar, de forma transparente, uma aplicação.

Várias vantagens técnicas resultam deste novo paradigma, como descrito a seguir:

- Empacotamento em objeto (object wrapper). Esse empacotamento em objeto consiste no encapsulamento de um conjunto de serviços providos por uma aplicação não-orientada a objeto ou no encapsulamento de uma interface de programa de maneira a poder tratar essa aplicação, ou interface, como um objeto. Por exemplo, esse empacotamento poderá ser utilizado para criar uma interface para uma aplicação legada, isto é, já existente, para que esta possa ser tratada como um objeto. Desta forma, um sistema legado pode ser utilizado em um ambiente distribuído, encapsulado como um objeto, poupando o desenvolvimento de um novo sistema orientado a objeto, que ofereça as funcionalidades do sistema já existente. Esse empacotamento pode também

ser aplicado a vários recursos já existentes, simplificando, assim, a comunicação entre eles através da rede.

- Distribuição dos objetos locais e remotos de uma determinada aplicação em computadores que melhor realizem as tarefas a eles definidas. Essa distribuição pode ser feita sem a necessidade de alterar a localização da aplicação que se utiliza desses objetos. Por exemplo, um objeto que realiza uma computação pesada como uma execução tridimensional pode ser colocado em um computador mais rápido, enquanto objetos de apresentação que interagem com o usuário ficariam em uma estação de trabalho mais lenta. Essa distribuição permite uma melhor utilização de recursos de hardware.

- Facilidade na migração da implementação de um objeto de uma plataforma a outra. Isto é possível, pois os objetos, mesmo remotos, podem parecer como sendo locais aos seus clientes. O cliente não sabe onde e em que tipo de máquina realmente reside a implementação de um objeto utilizado por ele. Essa migração pode ser feita em etapas, sem a necessidade de alteração na forma de acesso nos clientes.

- Recursos de hardware e software disponíveis em plataformas heterogêneas podem ser utilizados por uma aplicação. Tem-se a imagem de um sistema único que, na realidade, é formado por uma aplicação construída por objetos distribuídos.

De forma geral, a distribuição de objetos permite um avanço que objetiva tornar a informação distribuída mais eficiente, mais flexível e menos complexa. Entretanto, deve-se ressaltar algumas dificuldades para construir aplicações distribuídas orientadas a objetos, por exemplo (Eastman, 1997):

- Como essas aplicações devem ser implementadas.
- Como esses sistemas irão se comunicar.
- Como manter as informações num estado consistente.
- Como localizar serviços de forma a satisfazer às necessidades dos usuários.
- Como manter a segurança de acesso.

- Como as falhas podem ser resolvidas.
- Como gerenciar a evolução dos sistemas.

A distribuição de objetos pode ser utilizada na construção de base para a solução destes problemas, pois ela permite que (Eastman, 1997):

- Objetos de uma aplicação possam residir em qualquer lugar da rede.
- Serviços de persistência possam armazenar e recuperar objetos eficientemente.
- Serviços de pesquisa possam localizar objetos apropriadamente, onde quer que eles residam.
- Serviços de segurança possam restringir o acesso a objetos sensíveis.
- Serviços de concorrência possam prover isolamento de ações entre usuários.
- Serviços de transações possam coordenar atualizações em múltiplos objetos.

Para ilustrar, considerem-se, algumas características de objetos distribuídos (Sessions, 1998):

- Propósito: O propósito de um objeto distribuído é executar um conjunto de funções, relacionadas entre si, que atendam a múltiplos clientes remotos.
- Visão do cliente: A visão do cliente de um objeto distribuído é uma interface que define somente o comportamento (a funcionalidade) que o cliente pode esperar de um determinado objeto distribuído.
- Desempenho: O custo da invocação de um objeto distribuído, isto é, de um pedido de execução de um dos seus métodos, é medido em milissegundos, assim, para um objeto distribuído, não se pode ignorar o custo da invocação de métodos na determinação do desempenho, devendo esse custo ser cuidadosamente considerado na análise do desempenho geral de um sistema.
- Complexidade: Objetos distribuídos são usualmente complexos, mas essa complexidade é invisível ao cliente.

- Localização: Objetos distribuídos geralmente não estão localizados no mesmo espaço de endereçamento dos seus clientes.
- Concorrência: Objetos distribuídos são utilizados por um grande número de clientes, necessitando, com isto, um gerenciamento muito complexo de problemas de concorrência.

Um sistema distribuído orientado a objeto permite que os objetos sejam instanciados num ambiente distribuído (Andleigh, Gretzinger, 1992; Manola, 1999; Taylor, 1993). Esses sistemas, assim como qualquer sistema distribuído, possuem as seguintes características (Chin, Chanson, 1991):

- Distribuição: O sistema executa numa rede de computadores independentes e heterogêneos.
- Transparência: O sistema esconde o ambiente distribuído e outros detalhes desnecessários ao usuário. Por exemplo, um sistema pode prover a característica de transparência de localização e, com isso, o usuário não precisa se preocupar com a localização física de um objeto para fazer uma invocação.
- Tolerância a falhas: A falha de um computador, ou de um objeto, representa apenas uma falha parcial do sistema; sendo a perda restrita ao computador ou ao objeto. O restante do sistema continua processando.
- Disponibilidade: O sistema assegura a disponibilidade dos objetos, independente de falhas nos computadores.
- Autonomia dos objetos: O sistema permite ao criador do objeto especificar os clientes autorizados a operar sobre ele. Ou seja pode-se criar mecanismos de proteção para os objetos
- Concorrência no processamento: O sistema permite que objetos de um programa possam ser atribuídos a múltiplos processadores, para que eles possam ser executados concorrentemente.

- Concorrência nos objetos: Um objeto pode atender a múltiplas invocações de clientes concorrentemente .

Os objetos, para um sistema distribuído orientado a objeto, podem ser, por exemplo, partes de um programa, um banco de dados monousuário, estruturas de dados (listas ligadas, filas), dados pertencentes a um banco de dados multiusuário ou dados de um determinado tipo (booleanos, inteiros, reais) etc. (Gunji, 1995).

Os objetos são os recursos fundamentais de qualquer sistema orientado a objeto; portanto, o gerenciamento de objetos é uma função essencial destes sistemas. Assim como em qualquer sistema distribuído, os mecanismos de gerenciamento de objetos envolvem:

- Gerenciamento de transações: Tem a função de gerenciar transações, onde uma transação é uma coleção de operações que executa uma única função lógica numa aplicação. As transações têm as seguintes propriedades (Andleigh, Gretzinger, 1992; Eastman, 1997; Gunji, 1995):

- Serialização: Transações concorrentes são escalonadas de forma a serem executadas seqüencialmente em alguma ordem.

- Atomicidade: Uma transação é completada de forma total ou é abortada.

- Sincronização: Tem a função de garantir que atividades de múltiplas transações, invocando o mesmo objeto, não conflitem ou interfiram entre si.

- Segurança de acesso: Tem a função de atribuir diferentes níveis de segurança aos usuários para operar sobre diferentes conjuntos de objetos.

- Confiabilidade de objetos: Tem a função de num sistema detectar e recuperar objetos. Geralmente, a confiabilidade de objetos é obtida por sistemas de recuperação de falhas, podendo ocorrer de duas maneiras. Uma delas é recuperar a falha ocorrida num objeto. A outra é replicar os objetos em vários computadores e utilizar a cópia disponível.

Uma outra função de um sistema distribuído orientado a objeto é gerenciar a invocação entre objetos cooperantes. O sistema deve ser capaz de:

- Localizar um determinado objeto, isto é, prover a propriedade de transparência de localização, que possibilita a um cliente invocar um objeto sem conhecer a sua localização.
- Manipular as interações entre os objetos, isto é, quando um cliente faz uma invocação sobre um objeto, o sistema é responsável por executar os passos necessários para entregar o pedido para o objeto servidor especificado e retornar o resultado para o cliente.
- Detectar falhas de invocação, isto é, detectar falhas de hardware ou software, que ocorram antes de uma invocação ser iniciada ou durante a execução de uma invocação.

Um sistema distribuído orientado a objeto deve prover mecanismos para gerenciar os recursos físicos da rede; recursos esses que incluem:

- Persistência : Um objeto tem seus dados persistentes quando os armazena num meio relativamente permanente, como por exemplo um disco. Tornar os dados do objeto persistentes significa que é possível ao objeto parar de executar e na próxima instanciação utilizar os dados persistentes anteriormente armazenados;
- Balanceamento de carga: O principal objetivo do balanceamento de carga é maximizar a taxa de resposta do sistema. Isto pode ser obtido de duas formas. Primeiramente, os objetos podem ser atribuídos a processadores, que estão mais ociosos para que eles trabalhem concorrentemente, ou objetos que interajam freqüentemente possam ser atribuídos ao mesmo processador, ou a processadores próximos para reduzir o custo de comunicação entre eles.

Todas estas características de um sistema distribuído orientado a objeto devem ser transparentes ao usuário. Deve haver um meio que permite que diferentes objetos comuniquem-se entre si, de forma que eles não precisem conhecer detalhes de

localização e implementação uns dos outros. Isto é possível com a utilização da tecnologia “middleware”.

“Middleware” é um software de conectividade, que consiste em um conjunto de serviços, que permite a interação, através da rede, de múltiplos processos em execução em uma ou mais máquinas. Esse software de conectividade se localiza entre a aplicação e o sistema operacional (Bernstein, 1996). Ele oferece serviços de propósito geral, tais como:

- Ir ao encontro de uma grande variedade de aplicações, por exemplo, ser capaz de traduzir ou facilitar a adição de mensagens de vários formatos a serem utilizadas em diversas aplicações.
- Ser implementados de forma a possibilitar a execução em plataformas distintas. Por exemplo, em sistemas distribuídos, as aplicações localizadas em diferentes plataformas podem usar o serviço “middleware” para se comunicar e/ou trocar dados, aumentando assim a interoperabilidade (Betz-1994).
- Possibilitar serem acessados remotamente por outros serviços ou aplicações.
- Suportar, idealmente, um protocolo padrão, por exemplo, TCP/IP (Transmission Control Protocol/Internet Protocol) ou ISO (International Standards Organization) OSI (Open System Interconnect).

O principal propósito do “middleware” é ajudar na resolução dos problemas de conectividade e interoperabilidade de aplicações; mas é o desenvolvedor que tem a difícil tarefa de decidir quais funcionalidades serão colocadas no lado cliente e quais estarão no lado servidor da aplicação distribuída. Dessa forma, é importante entender o problema que será resolvido pela aplicação e o valor dos serviços “middleware” que permitirão a distribuição desta aplicação (Bray, 1998; Bernstein, 1996).

CAPÍTULO 3

SUPORTE PARA A IMPLEMENTAÇÃO DE SISTEMAS DISTRIBUÍDOS

Diz-se que a parte mais longa da viagem é a passagem pelo portão (Marcus Terentiuns Varro)

O presente capítulo descreve, de forma resumida, os principais sistemas de software existentes no mercado, que poderão ser utilizados no processo de distribuição. Maiores detalhes destas arquiteturas podem ser obtidos na dissertação de mestrado de Costa(Costa-2000).

- DCOM - Distributed Component Object Model;
- CORBA - Common Object Request Broker Architecture e
- RMI - Remote Method Invocation.

3.1 - DCOM

DCOM (Distributed Component Object Model) é uma extensão do COM (Component Object Model). Ele permite que o modelo de objeto definido pelo COM possa ser utilizado para distribuição de objetos através de redes. Desta forma, descreve-se primeiro o COM e posteriormente o DCOM.

O COM é um modelo de programação, baseado em objetos e projetado para promover interoperabilidade de software, isto é, permite que dois ou mais softwares aplicativos cooperem entre si para realizar uma tarefa. Para suportar as características de interoperabilidade, o COM possui um mecanismo que permite aos softwares aplicativos conectarem-se uns aos outros, como se fossem objetos de software.

O DCOM baseia-se no modelo de objeto definido pelo COM, acrescentando a ele novas funcionalidades (Chappell, 1996; Grimes, 1997):

- Técnicas para criação de um objeto remoto.
- Protocolo para invocação dos métodos de um objeto remoto.

3.1.1 Criação de um Objeto Remoto

O DCOM, assim como o COM, permite a um cliente criar um objeto, utilizando-se a biblioteca COM, ou usando "monikers", com particularidades que serão descritas a seguir. Um cliente, tipicamente, cria um objeto chamando "CoCreateInstance" e, então, usa a interface "QueryInterface" para requisitar o ponteiro de que ele necessita. Para criar um objeto remoto no DCOM, o procedimento é o mesmo, mas há a necessidade de especificar um item adicional, isto é, a máquina na qual o objeto remoto será criado.

Para cada objeto criado na mesma máquina do cliente, existe um sistema de registro que liga o CLSID, especificado pelo cliente, ao nome da DLL ou do arquivo executável correspondente. Para um objeto criado em outra máquina, o sistema de registro mapeia o CLSID, ao nome da máquina na qual o objeto foi criado. Assim, para criação de um objeto remoto é necessário contatar a máquina desejada, pesquisar no seu sistema de registro o CLSID passado pelo cliente e disparar o processo de criação do objeto.

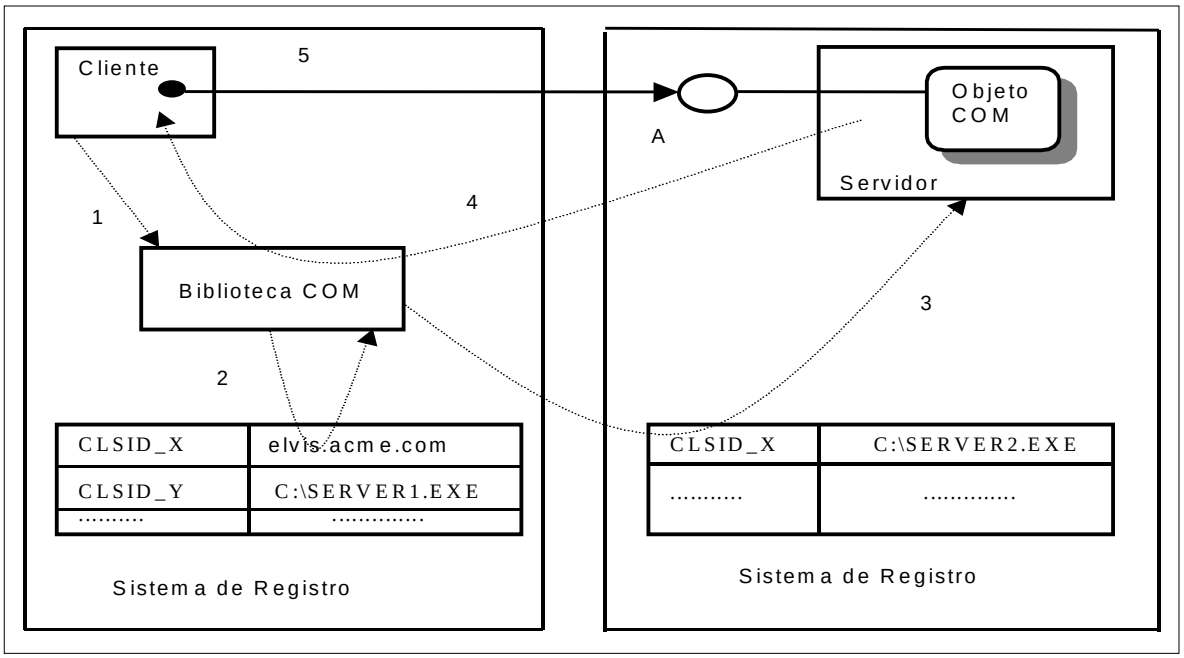


Fig. 3.1 – Criação de objeto COM remoto usando CoCreateInstance.

FONTE: adaptada de Chappell, (1996, p.239).

Como mostra a Figura 3.1, o cliente pede à biblioteca COM para criar o objeto com CLSID X (passo1), requisitando o ponteiro inicial para a interface A do objeto. O registro do CLSID X, na máquina do cliente, contém o nome de uma outra máquina (elvis.acme.com), (passo 2). O DCOM possui várias formas de identificação de máquinas remotas. Esta identificação depende do protocolo de rede que está sendo usado para acessar o sistema remoto. Esses protocolos podem ser TCP/IP, endereço IP (Internet Protocol), NetBios e NetWare's IPX/SPX. Uma vez identificada e contatada a máquina remota, o objeto é criado, usando a informação do CLSID X, encontrada no sistema de registro da máquina remota (Figura 3.1 C:\SERVER2.EXE- passo 3). Criado o objeto e retornado o ponteiro para a interface A (passo 4), o cliente utiliza-se do objeto, como se ele fosse criado localmente (passo 5).

O *sistema de registro* é um banco de dados que deve conter todas as classes de todos os objetos localizados numa determinada máquina. O sistema de registro pode ser implementado de diferentes maneiras, mas deve-se obedecer aos seguintes requisitos (Chappell, 1996):

- Utilizar o CLSID como chave de entrada.
- Indicar que tipo de servidor de objetos está disponível (no mesmo processo, local ou remoto).
- Indicar para servidores de objetos, no mesmo processo, o caminho (pathname) do arquivo, contendo a DLL e, para servidores de objetos locais, o caminho (pathname) do arquivo executável. Para servidores de objetos remotos (aqueles acessíveis em outra máquina, através do DCOM), indicar onde encontrar o arquivo executável.

3.1.2 Acesso a um Objeto Remoto

Criar um objeto COM remotamente é um passo fundamental para a distribuição. Para o cliente, o objeto remoto deve poder ser acessado como um objeto local, sem diferenças.

Como já descrito, um cliente, tendo adquirido os ponteiros de interface de um objeto, pode, através deles, invocar seus métodos. Se o objeto encontra-se no mesmo processo do cliente, ou em processos diferentes, mas na mesma máquina, a invocação dos

métodos é feita diretamente utilizando os recursos de “proxy” ou “stub” e algum tipo de comunicação entre processos. Quando o objeto se encontra em máquinas diferentes, utiliza-se o protocolo RPC (Remote Procedure Call). A Microsoft adotou um já existente, ao invés de criar um novo. Este protocolo chama-se MS RPC e foi emprestado da “Open Software Foundation’s Distributed Computing Environment” (OSF DCE). A MS RPC usada pelo DCOM é conhecida como “Object RPC” (ORPC). O ORPC envia informação para a rede, no mesmo formato que o DCE RPC, isto é, usando a mesma estrutura de pacotes. Apesar de incluir novas convenções para interação entre cliente e objeto, bem como de adicionar novos tipos de dados, etc, ela não mudou a estrutura de pacotes originais (COM, 1998;Chappell, 1996;DCE,1998).

3.2 CORBA

Um dos grandes problemas das empresas é, utilizando seus recursos de hardware e de software, integrar vários e diferentes elementos de trabalho de maneira a resolver problemas de negócios atuais e futuros. Uma parcela significativa deste problema é integrar as aplicações existentes em “mainframes” com os novos ambientes de estações de trabalho. As soluções são caras e demoradas. O padrão CORBA é uma das formas utilizadas para solucionar esses problemas. O CORBA é baseado em orientação a objeto e no modelo de computação distribuída cliente/servidor.

Na computação distribuída, uma requisição de um serviço é feita de um componente de software (cliente) para outro (servidor) através da rede. O padrão CORBA acrescenta a esse modelo um “Broker” (Object Request Broker - ORB), que tem a função de reduzir a complexidade da implementação, necessária para permitir a interação entre cliente e servidor (CORBA, 1998; Mowbray, Ruh, 1997).

A redução de complexidade é obtida primeiramente porque o ORB provê serviços comuns, que incluem troca de mensagens (comunicação) entre cliente e o servidor, e transparência de localização. Ele também isola os objetos da aplicação das configurações específicas de um sistema, tais como, plataformas de hardware e sistemas operacionais, protocolos de rede e linguagens de programação (CORBA, 1998). Os ORBs têm uma função análoga a de um “bus” de hardware, isto é, eles provêem um caminho para o fluxo de toda a informação que flui entre os objetos.

3.2.1 Conceitos

Com a utilização do ORB, os clientes e servidores são formalmente separados; assim, mudanças em um não afetam o outro. O cliente CORBA sabe somente pedir que algo seja realizado pelo servidor CORBA e esse sabe somente realizar a tarefa a ele requisitada. Isso significa que pode-se mudar a forma com a qual um servidor executa uma tarefa, sem afetar a maneira com que o cliente requisita a tarefa. Por exemplo, pode-se evoluir e criar uma nova implementação de um servidor CORBA, sem a necessidade de alterar o cliente; ou pode-se criar novos clientes que façam uso de interfaces já existentes de servidores CORBA sem alterá-los.

a) Requisições

No CORBA, cliente e servidor comunicam-se através de mensagens chamadas requisições. Toda a interação em um sistema CORBA é baseada num cliente que envia uma requisição ou num servidor que responde a uma requisição. O processo de enviar uma requisição é chamado de invocação.

b) IDL

Em qualquer aplicação distribuída, o cliente e o servidor necessitam de algumas informações básicas para comunicarem-se, como por exemplo, o cliente precisa conhecer quais operações e seus respectivos argumentos estão disponíveis para serem requisitados. No CORBA essa informação é incluída na interface. A interface define as características e o comportamento de objetos, incluindo as operações que podem ser requisitadas a esses objetos. Usando a terminologia de orientação a objeto, uma interface é similar a uma classe (Otte, Patrick, Harkey 1996).

Para definir interfaces, o CORBA, utiliza a linguagem de definição de interfaces (Interface Definition Language - IDL). A IDL é uma linguagem puramente declarativa, isto é, ela declara o conjunto dos nomes dos métodos do objeto e seus respectivos parâmetros; portanto, ela é uma linguagem de definição e não de programação. Sua utilização se restringe em definir interfaces e estruturas de dados e não para escrever algoritmos (Otte, Patrick, Roy, 1996).

Clientes e objetos podem ser implementados em qualquer linguagem de programação, pois os clientes, através da utilização das interfaces em IDL, possuem todas as informações necessárias para requisitar uma operação ao objeto. Clientes não precisam conhecer os detalhes da implementação do objeto (Orfali, Harkey, Edwards, 1996).

A IDL suporta mapeamento para várias linguagens de programação, tais como, Java, C, C++, Smalltalk, Cobol e Ada95. . A Figura 3.2 ilustra um exemplo da IDL CORBA. Neste exemplo, um arquivo IDL CORBA descreve uma interface Empregado, que define as operações promoção e demissão e os tipos de dados definidos pelo usuário. A aplicação do cliente que utiliza esse arquivo deve ser capaz de requisitar as operações promoção e demissão e utilizar os tipos de dados especificados. O servidor que satisfará a requisição desse cliente deve ser capaz de realizar o trabalho associado às operações promoção e demissão e manipular os tipos de dados especificados.

```
Interface Empregado
{
  void promoção    ( in char              nova_colocação );
  void demissão    ( in Cod_Demissão      razão,
                     in string            descrição );
};
```

Fig. 3.2 – Exemplo de uma definição de interface.

FONTE: Otte, Patrick, Roy (1996, p. 2-4).

c) Códigos de implementação de objetos

A implementação do código de um objeto pode estar em um ou mais objetos, também denominados objetos servidores. O objeto que utiliza os serviços prestados pelos objetos servidores assumem o papel de objetos clientes. Considerando a IDL, descrita na Figura 3.4, um objeto cliente requisita a operação de promoção num Empregado específico, sendo que a implementação deste serviço encontra-se em um outro objeto servidor.

3.2.2 Arquitetura CORBA

A Figura 3.3 ilustra a arquitetura CORBA.

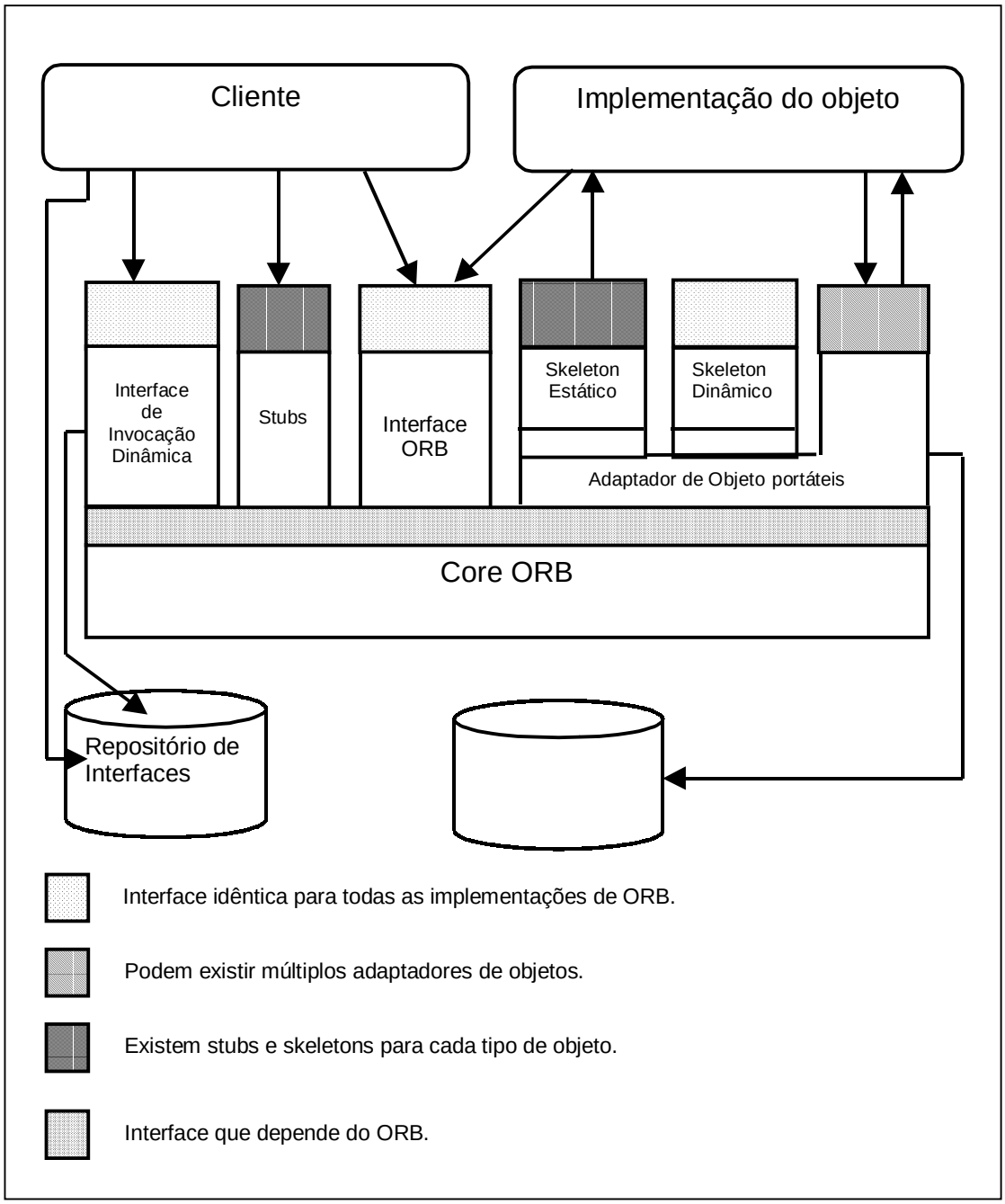


Fig. 3.3 – Arquitetura CORBA.

FONTE: adaptada de CORBA, (1998, p. 2.3).

No topo da Figura 3.3, fora do ORB, estão representados o cliente e a implementação do objeto. O restante da Figura representa elementos pertencentes ao ORB. Abaixo, nesta Figura, está o “core” (núcleo) do ORB, cujos detalhes não são controlados pela especificação CORBA. O “core” do ORB pode ser implementado da maneira que o seu fornecedor desejar (Mowbray, Ruh 1997).

Um cliente para fazer uma invocação ao objeto pode tomar dois caminhos, ou seja, via “stubs”, ou através da Interface de Invocação Dinâmica. Utilizando os “stubs”, um cliente acessa um método de um objeto remoto da mesma maneira que acessaria um outro, residindo no seu próprio espaço de endereçamento. As rotinas dos “stubs” são geradas, automaticamente, no momento da compilação do arquivo que contém a descrição, em IDL, da interface do objeto CORBA a ser invocado, devendo ser “linkado” com a aplicação do cliente. Os “stubs” acessam as referências do objeto e interagem com o ORB para realizar a invocação requisitada. Os “stubs” dependem do ORB; pois, utilizam-se de mecanismos otimizados para interagir com ele e do mapeamento da linguagem de programação do cliente em IDL. Diferentes fornecedores de ORB apresentam diferentes “stubs”.

Utilizando a Interface de Invocação Dinâmica (Dynamic Invocation Interface - DII) um cliente, em tempo de execução, invoca um objeto CORBA, sem o auxílio dos “stubs”. Para isso, é necessário especificar, na invocação dinâmica do objeto CORBA, o método a ser executado e o seu conjunto de parâmetros. Tal recurso é necessário quando não se tem acesso aos arquivos com os “stubs”, criados em tempo de compilação. Essa interface é de especial interesse para certos tipos de aplicações que precisam ter acesso a objetos remotos, em tempo de execução; porém, necessitam do auxílio do Repositório de Interfaces(Orfali, Harkey, 1996).

O Repositório de Interfaces contém a descrição das interfaces dos objetos CORBA nele registrados, isto é, possuem o mesmo tipo de informação contido nos “stubs”. As suas APIs (Application Programming Interface) permitem o acesso, o registro e a atualização dessas informações, auxiliando a construção de invocações dinâmicas através da DII. Tais informações, contidas no repositório, podem ser consideradas como metadados dos

objetos CORBA (Queiroz, Madeira, 1998). A Figura 3.4 mostra a invocação via “stubs” e DII.

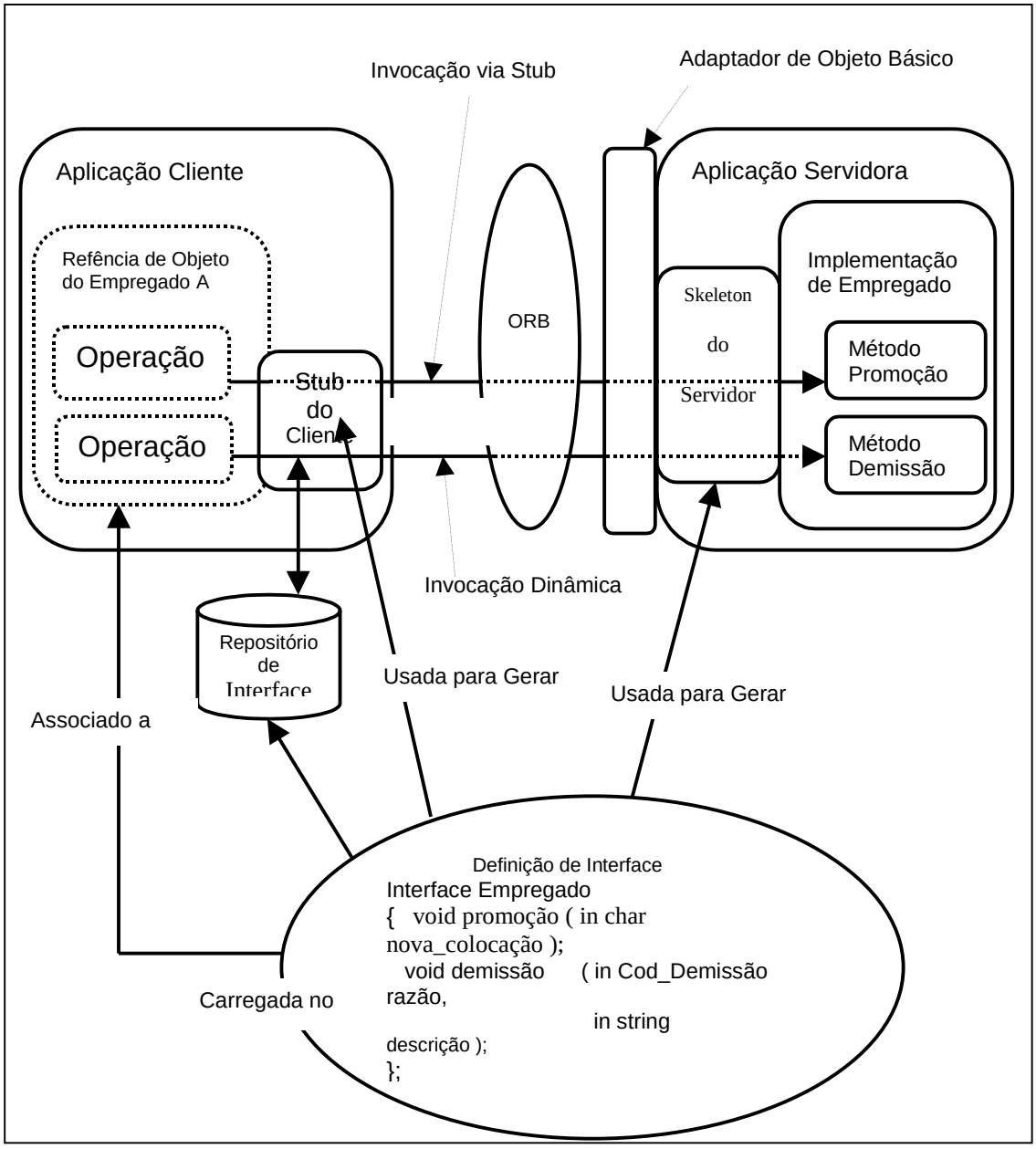


Fig. 3.4– Invocação via “stubs” e DII.

FONTE: Otte, Patrick, Roy(1996, p. 8.8).

Um servidor é composto por um ou mais objetos servidores, que contêm as implementações dos objetos clientes. Quando um objeto cliente envia uma requisição

para o ORB, esse seleciona a implementação que satisfaz a requisição e usa o Adaptador de Objeto (Object Adapter- AO) para direcionar a requisição para a implementação correspondente. O Adaptador de Objeto utiliza os "skeletons" para chamar os métodos na implementação. A maioria dos Adaptadores de Objeto são projetados para cobrir uma faixa larga de implementações, objetivando restringir o projeto de um novo adaptador. Isto ocorrerá somente quando uma implementação requeira um serviço ou uma interface muito diferente. O Adaptador de Objeto Portável ("Portable Object Adapter" – POA) foi especificado de forma a prover um Adaptador de Objeto que possa ser usado em múltiplos ORBs, isto é, um Adaptador de Objeto que, com um mínimo de alteração, possa interagir com diferentes ORBs. O capítulo 9 da especificação "The Common Objet Request Broker: Architecture and Specification" (CORBA, 1998) descreve o POA de forma detalhada (projeto, modelo abstrato, interfaces, etc.).

"Skeletons" estáticos são responsáveis por prover uma conexão entre os Adaptadores de Objeto e os métodos que executam as operações num objeto. São similares aos "stubs" do cliente, sendo gerados, automaticamente, no momento da compilação do arquivo que contém a descrição em IDL de cada método do objeto CORBA.

Quando os objetos CORBA não possuem os correspondentes "skeletons" estáticos, os "skeletons" dinâmicos serão responsáveis por oferecer um mecanismo de acesso a esses objetos. Tal recurso inspeciona os valores dos parâmetros da mensagem recebida para determinar o objeto destinatário e o correspondente método, ele pode utilizar o Repositório de Interfaces para obter informações (CORBA, 1998; Orfali, Harkey, 1996).

Um Adaptador de Objeto Básico (Basic Object Adapter – BOA), mostrado na Figura 3.4, é responsável por (CORBA, 1998; Orfali, Harkey, Edwards, 1996):

- Gerar e interpretar referências de objetos.
- Invocar métodos.
- Ativar e desativar objetos.
- Mapear as referências dos objetos às suas correspondentes implementações.

- Registrar implementações.

3.2.3 Interoperabilidade CORBA

As especificações CORBA possibilitam a comunicação ORB a ORB, isto é, a interoperabilidade entre ORBs (interORBability). Um ORB, como já descrito, provê mecanismos, pelos quais objetos, de forma transparente, enviam requisições e recebem respostas. Um ORB permite interoperabilidade entre aplicações executando em diferentes máquinas em ambientes distribuídos heterogêneos. A interoperabilidade entre ORBs estende esta definição para o caso em que o cliente e o objeto servidor estejam em diferentes ORBs, isto é, de forma transparente, objetos enviam requisições e recebem respostas. O CORBA especifica uma abordagem detalhada e flexível para dar suporte a objetos distribuídos, através de redes, gerenciados por ORBs heterogêneos. Esta abordagem permite que seus elementos possam ser combinados de várias maneiras a satisfazer uma ampla faixa de necessidades (Stone, Curtis, Bradley, 1998).

3.2.4 "Object Management Architecture" – OMA

O nível mais alto das especificações CORBA é a Arquitetura de Gerenciamento de Objeto (Object Management Architecture - OMA). Ela constitui-se de (Otte, Patrick, Roy, 1996; Mowbray, Ruh, 1997):

- Serviços CORBA: São objetos que implementam um conjunto de funções padronizadas para a criação e o controle de acesso aos objetos, rastreamento de objetos e referências de objetos etc. Esses serviços são essencialmente um conjunto de funções, criadas para facilitar o desenvolvimento das aplicações; assim, o desenvolvedor pode chamar essas funções ao invés de criar as suas próprias. Como exemplo pode-se citar o serviço de identificação de objetos por nomes utilizados para localizar objetos, serviços de eventos utilizados para comunicação entre objetos, serviço de ciclo de vida utilizado para controle da criação de objetos, serviço de persistência utilizado para armazenagem de objetos etc.
- Facilidades CORBA: São coleções de objetos que provêm um conjunto de funções de alto nível, de propósito geral, para serem utilizadas em diferentes aplicações, tais como, facilidades para gerenciar a composição de documentos, acessar banco de

dados, imprimir arquivos, sincronizar a temporização em um ambiente distribuído Existe também as facilidades para se trabalhar com agentes móveis, Mobile Agent Facility (MAF). Esta facilidade agrega recursos para criação, comunicação e migração de objetos dentro de um sistema distribuído.(CORBA-MAF-2000).

- Domínios CORBA: Suportam tarefas de domínios específicos associados a segmentos de mercado, como manufatura, finanças e telecomunicações.
- Objetos aplicativos: Provêm um conjunto de objetos que executam tarefas específicas para usuários finais. São essencialmente aplicações orientadas a objeto desenvolvidas para prover capacidades de negócios (bussiness capabilities). A OMG não padroniza esses objetos.

A Figura 3.5 ilustra a arquitetura OMA, onde o ORB é o barramento comum entre os aplicativos, os serviços CORBA (CORBAservice), as facilidades CORBA (CORBAfacilities) e os domínios CORBA (CORBAdomains).

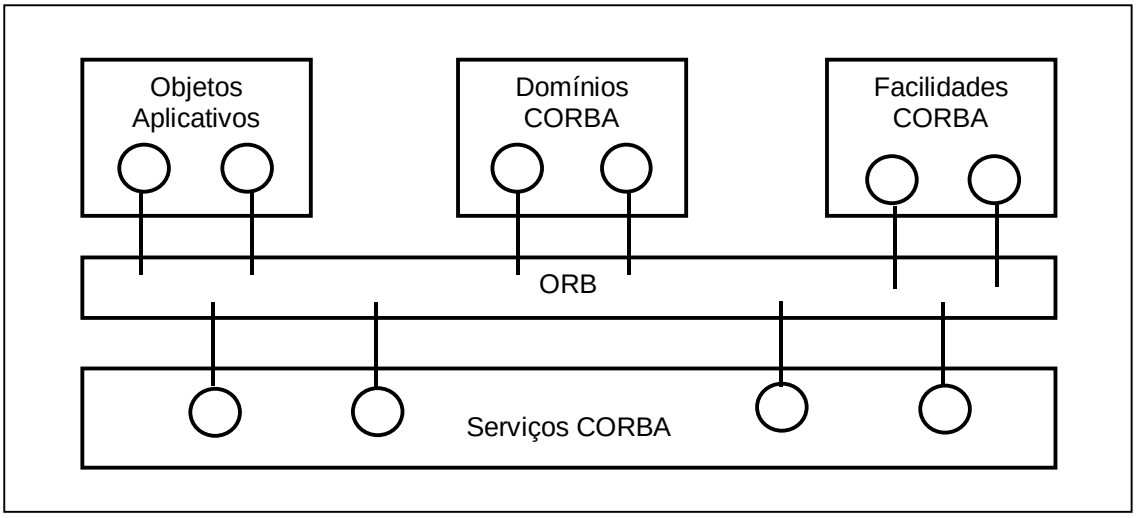


Fig. 3.5 – Arquitetura de Gerenciamento de Objeto (OMA).

FONTE: Mowbray, Ruh (1997, p. 9).

3.2.5 Serviços CORBA

Os serviços CORBA são fundamentais e globalmente aceitos. São úteis a todos os tipos de aplicação e independentes do domínio da aplicação. Dentre os principais podem ser citados (Mowbray, Ruh 1997):

3.2.5.1 Serviço de Persistência de Objeto

O serviço de persistência provê interfaces comuns aos mecanismos responsáveis pelo armazenamento e gerenciamento do estado persistente dos objetos, como arquivos, banco de dados relacional, banco de dados orientado a objeto etc.

O serviço de persistência permite ao objeto manter-se persistente, além da aplicação que o criou ou do cliente que o utilizou. O ciclo de vida de um objeto pode ser relativamente curto ou indefinido; assim, esse serviço armazena o estado do objeto e o restaura quando necessário.

3.2.5.2 Serviço de identificação dos objetos

O serviço de identificação dos objetos (name service) é um serviço genérico utilizado para localizar um objeto, isto é, se o cliente possui o nome do objeto, ele pode, através desse serviço, recuperar a referência ou o endereço de memória desse objeto (Mowbray, Ruh 1997).

As operações chaves do serviço de nomeação são de “ligar” (bind) e “resolver” (resolve). A operação de “ligar” é usada para adicionar um nome de objeto e sua referência no diretório do serviço. Na operação de “resolver”, o cliente entra com um nome de objeto particular e recebe a referência ou endereço desse objeto, como valor de retorno, ou uma exceção, caso o nome não se encontre no diretório.

A associação nome/objeto é chamada nome de ligação (name binding). O nome de ligação é sempre definido em relação a um nome de contexto (name context), que é um objeto contendo um conjunto de nomes de ligação. “Resolver” um nome é determinar o objeto associado a esse nome, num dado contexto, referenciado por um nome de contexto. “Ligar” um nome é criar um nome de ligação nesse contexto. O nome é sempre resolvido em relação a um contexto; não existem nomes absolutos. A Figura 3.6 ilustra um exemplo de alguns objetos de contexto e os nomes gerenciados por eles. Os nós no gráfico de nomes (naming graph) indicam os objetos nomes de contexto.

Cada objeto nome de contexto, no gráfico de nomes, define diferentes escopos e subescopos no espaço de nomes. Cada folha corresponde ao nome da instância do objeto. Os nomes estão indicados nas setas do gráfico. O caminho no gráfico de nomes serve para, dado um determinado nome de objeto fornecido por um cliente, obter a sua referência como valor de retorno, ou uma exceção (CORBAservices, 1998). Por exemplo, o objeto de nome c2 está indicado, no gráfico de nomes, pelo caminho dado pelo contexto sys e pelo contexto bin.

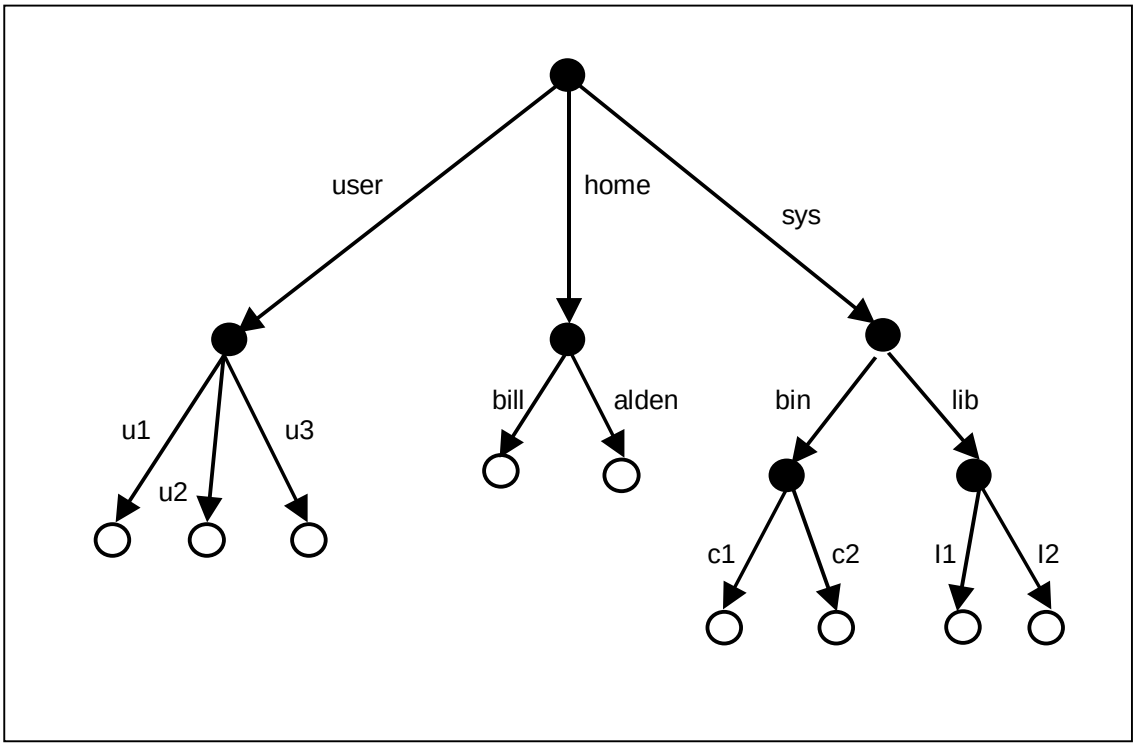


Fig. 3.6 – Exemplo de um gráfico de nomes.

FONTE: CORBAservices, (1998, p. 3.2).

3.2.5.3 Serviço de Segurança

O serviço de segurança possui características que endereçam questões de segurança tais como:

- Identificação e autenticação: Estes serviços verificam se usuários e objetos são quem eles dizem que são e que direitos possuem.

- Autorização e controle de acesso: São serviços que decidem se um objeto pode ser acessado ou não, normalmente usando sua identificação e/ou um atributo de privilégio e um atributo de controle do objeto-alvo.
- Auditoria de segurança: É um mecanismo de auditoria capaz de identificar o usuário corretamente, mesmo depois de uma cadeia de chamados através de vários objetos.
- Segurança de comunicação: Ela permite estabelecer uma comunicação confiável entre cliente e objeto e define também uma proteção de integridade e proteção de confiabilidade das mensagens transmitidas entre os objetos.
- Administração: Administra a segurança da informação.

3.2.6 Facilidades CORBA

As facilidades CORBA são, por definição, coleções de objetos que proporcionam serviços a serem utilizados diretamente pelas aplicações, englobando funcionalidades de mais alto nível do que as disponibilizadas pelos serviços CORBA. Podem ser definidas como sendo especializações dos serviços CORBA. Como exemplo de facilidades, têm-se correio eletrônico (e-mail) e documentos compostos “compound documents” (CORBAfacilities, 1998).

3.3 RMI

Sistemas distribuídos necessitam que aplicações sejam capazes de se comunicar, executando em vários espaços de endereçamento, normalmente em diferentes computadores. Para um mecanismo de comunicação básico, a plataforma Java suporta o conceito de “socket”.

O endereçamento de equipamentos na Internet é baseado em um identificador que independe da tecnologia de rede envolvida, conhecido como endereço IP (Internet Protocol), cuja a notação é o do exemplo a seguir: “128.32.96.4”. O endereço IP permite apenas a localização de um dado equipamento na Internet. Para a localização de uma aplicação qualquer, executando nesse equipamento, existe um outro identificador conhecido como “port”. Essa localização é realizada através da associação do “port”

com um dos protocolos de transporte TCP (Transmission Control Protocol) ou UDP (User Datagram Protocol). Essa tríade, composta por endereço IP, número de "port" e protocolo de transporte, é conhecida como "**socket**" (Cyclades, 1996).

Apesar de o "socket" ser flexível o suficiente para comunicação em geral, ele requer tanto do cliente, quanto do servidor, o conhecimento dos protocolos para troca de mensagens, isto é, a codificação e decodificação dessas mensagens. O projeto desses protocolos é sempre difícil e susceptível a erros. O Java suporta o conceito de invocação de método remoto (Remote Method Invocation – RMI), que permite uma abstração de mais alto nível ao desenvolvedor, no processo de comunicação entre cliente e servidor. O RMI permite manipular objetos distribuídos utilizando Java. Exemplificando, assim como o CORBA especificou um padrão de comunicação entre cliente e servidor, onde esses podem ser escritos em qualquer linguagem de programação, o padrão RMI foi projetado para ser especialmente integrado à linguagem Java, isto é, clientes escritos em Java se comunicam com objetos escritos em Java através do RMI.

Basicamente o RMI é um RPC (Remote Procedure Call), que, ao invés de realizar chamadas a procedimento remotos, realiza chamadas aos métodos de um objeto remoto. Um objeto remoto é aquele que se localiza em uma JVM (Java Virtual Machine), diferente daquela em que se encontra o cliente, ou seja, em outro computador. Um objeto remoto é descrito por uma ou mais interfaces escritas na própria linguagem de programação Java. Por ser tratar de um objeto remoto, estas interfaces são chamadas de interfaces remotas. A invocação de um método de um objeto remoto tem a mesma sintaxe que a invocação de um método de um objeto local. O cliente, quando invoca um objeto, não sabe se esse objeto é local ou remoto em relação a ele.

O RMI providencia a:

- Localização (binding) de um método de um objeto remoto.
- Comunicação, através de uma conexão de rede confiável, permitindo a transmissão de requisições e respostas, funcionando como um canal entre cliente e servidor.

- Interface com usuário: converte a chamada de um método em um bloco de dados (mensagem), que pode trafegar através da conexão e remonta a mesma chamada de método, no outro extremo (remoto), a partir dessa mensagem. São necessários códigos nos pontos finais da conexão para deixar transparente o uso do RMI pelo usuário. No lado cliente, um “stub” tem a mesma assinatura (nome, argumentos e tipo de retorno) que o método remoto, e sua função é empacotar os argumentos em uma mensagem para ser enviada através do canal de comunicação. O complemento do “stub”, localizado no lado do servidor, é chamado de “skeleton”. Sua função consiste em converter as mensagens que chegam pelo canal de comunicação em argumentos e chamar a implementação do método requisitado. Quando o método termina a execução, o valor de retorno é convertido em uma mensagem, que é remetida ao “stub” do cliente (java,1999; Sun,1999).

3.4 CONSIDERAÇÕES FINAIS

Tanto o CORBA, quanto o DCOM e o RMI, auxiliam a resolução de problemas surgidos quando se definem e se implementam objetos distribuídos num ambiente. A meta dos sistemas distribuídos é permitir que códigos de programas e dados, armazenados em diferentes plataformas, trabalhem em conjunto. O CORBA e o DCOM provêem suporte similar para implementação de objetos acessíveis remotamente pelas aplicações. Esta capacidade simplifica a construção dos sistemas distribuídos.

As organizações devem utilizar padrões como guias no desenvolvimento de sistemas, porque tanto produtos podem ser comprados, como aplicações podem ser desenvolvidas com a confiança de que o hardware e o software trabalharão juntos por muitos anos. Os benefícios da padronização dos sistemas, em ambientes distribuídos, evidenciam-se na redução da complexidade do gerenciamento, no aumento do grau de interoperabilidade e usabilidade, assim como na redução dos custos de administração e suporte, pois estas se tornam mais especializadas (Nimer, 1998).

Ao adotar um padrão, uma organização deve considerar a sua disponibilidade, maturidade, influência, estilo, precisão e propósito das suas especificações, assim como os produtos disponíveis, ferramentas de suporte ao desenvolvimento, aceitação pelos desenvolvedores e referências de sucesso de implementações compatíveis a esse padrão.

Deve-se levar em conta também a plataforma de hardware (estações de trabalho, PCs etc.), o sistema operacional (UNIX, Windows etc.) e a rede disponível na organização, assim como o nível de preparo da equipe (conhecimento da tecnologia de distribuição, padrões, conceitos de orientação a objeto, linguagens de programação orientadas a objeto, etc.).

Embora enfoquem diferentes aspectos, tanto os produtos que seguem a especificação CORBA, quanto o DCOM, são produtos cujo propósito é atender à complexidade da distribuição de objetos. O CORBA foi especificado para suprir os requisitos necessários à construção de um sistema distribuído. Surgiu como uma solução aos problemas enfrentados pelas organizações ao tentarem integrar uma mescla de sistemas, "mainframes", UNIX com várias arquiteturas de hardware, sistemas operacionais proprietários de minicomputadores, PCs Macintosh e IBM etc. A meta foi combinar objetos e tecnologia de sistemas distribuídos dentro de uma arquitetura "interoperável" que pudesse resolver problemas de integração num empreendimento (enterprise). Dessa forma, o foco principal do CORBA é definir uma estrutura que melhor solucione problemas de ambientes heterogêneos (Tallman, 1998).

CAPÍTULO 4

TECNOLOGIA DE AGENTES

Fechei vossas palavras na memória, a chave, vós mesmo a guardareis. (Willian Shakespeare)

Os agentes se tornaram um interessante tópico de pesquisa e estudo, pois figuram em diversas disciplinas da Ciência da Computação, incluindo arquiteturas orientadas a objeto e arquiteturas de objetos distribuídos, engenharia de software, sistemas de aprendizado adaptativos, inteligência artificial, sistemas especialistas, algoritmos genéticos, processamento distribuído, algoritmos distribuídos e segurança, para apenas enumerar alguns campos. O presente capítulo pretende explorar os principais conceitos relacionados com a tecnologia de agentes empregados neste trabalho de pesquisa; entretanto, maiores detalhes podem ser obtidos na dissertação de mestrado de Silva (Silva,2000).

O termo agente, entretanto, assim como objeto, não possui uma definição unanimemente aceita, em parte devido à multiplicidade de enfoques sob os quais é estudado. Além disso, a área é criticada dentro da própria Ciência da Computação, onde algumas correntes nem mesmo consideram os agentes como uma nova tecnologia, alegando que qualquer coisa que possa ser feita com agentes pode também ser feita em qualquer linguagem.

Os principais pontos que tornam os agentes alvos de críticas são (Hermans, 1996):

- Outras subáreas da IA (sistemas especialistas, redes neurais) não teriam obtido tanto sucesso quanto alguns esperavam e o novo paradigma dos agentes seria ***a válvula de escape***.
- Tudo com o rótulo ***agente*** vende (isto também seria verdadeiro na pesquisa).
- Como alguns agentes de software disponíveis no mercado possuem uma arquitetura que não é nem muito complexa, nem sofisticada, pergunta-se o que os qualificaria como ***inteligentes***.

Os pesquisadores da área, por sua vez, refutam tais críticas com argumentos como os seguintes:

- O que distingue as arquiteturas baseadas em agentes de outras arquiteturas é que elas oferecem soluções razoáveis para certa classe de problemas a um custo aceitável.
- Os agentes tornam possível superar as diferenças entre os diversos tipos de redes e plataformas.
- A característica de distribuição, cada vez mais presente nos ambientes computacionais, é muito mais facilmente tratada por meio de agentes.

Estes prós e contras, quanto à nova tecnologia, não representam evidentemente a lista completa e devem ser tomados como ilustração. O que se quer mostrar é a necessidade de uma definição tão clara e precisa quanto possível para agente.

4.1 - O CONCEITO DE AGENTE – UMA DEFINIÇÃO?

Durante um debate ou uma simples conversa entre pesquisadores, levantam-se questões a respeito de agentes autônomos. A partir do momento em que uma breve explanação sobre o assunto deixa divagações duvidosas "no ar" como *mas agentes se parecem justamente com um programa... como diferenciá-los?*, torna-se evidente a necessidade de uma explicação mais satisfatória. Logo, de imediato, busca-se defender a noção de um agente, despida do perfil único o de um programa, propondo-se uma nova conceituação.

Portanto, o que é um agente? Relacionam-se, abaixo, várias definições, experimentalmente vinculadas a pesquisadores empenhados no estudo e desenvolvimento de agentes. O exame e comparação de algumas destas definições orientam a elaboração do novo conceito (Franklin, Graesser, 1996).

O Agente AIMA – Artificial Intelligence: a Modern Approach – "Um agente é qualquer coisa que perceba seu ambiente através de "sensors" – sensores – e aja sobre este ambiente através de "effectors" – efetivadores". Esta definição depende crucialmente do que se considera como um ambiente e do que significam os termos

"sentir" e "agir". Desejando chegar a um contraste útil entre agente e programa, deve-se restringir, pelo menos, algumas noções referentes a esta dependência.

O Agente Hayes-Roth - "Agentes Inteligentes realizam continuamente três funções: percepção de condições dinâmicas no ambiente; ação para afetar condições no ambiente; e raciocínio para interpretar percepções, solucionar problemas, redigir inferências e determinar ações." Surge o raciocínio durante o processo de selecionar ações.

O Agente IBM (http://activist.glp.ibm.com:81/WhitePaper/pct2.html) "Agentes inteligentes são entidades-software que manipulam conjuntos de operações em benefício de um usuário ou de outro programa com um certo grau de independência ou autonomia, e de alguma forma, empregar conhecimento ou representação das metas e anseios do usuário.". Esta definição vislumbra um agente inteligente, atuando por concessão de um indivíduo ou de um outro agente.

4.2 - A ESSÊNCIA DE UM AGENTE

Evita-se, normalmente, prescrever argumentos a respeito de como uma palavra deveria ser utilizada. Russell e Norvig, a este respeito, colocam: "A noção de um agente converge para uma ferramenta de análise de sistemas, não uma caracterização absoluta que divida o mundo em agentes e não-agentes." (Russell, Norvig, 1995). Visto que agentes "vivem" no mundo real e os conceitos deste mundo produzem categorias desfocadas, tenta-se capturar a essência de ser um agente e definir a ampla classe de agentes.

As definições da seção anterior parecem derivar de uma ou duas utilizações comuns da palavra "agente". – ***alguém que age ou pode agir*** e ***- alguém que age no lugar de outro com permissão.***

Pode-se tomar como exemplos de agentes, baseados na seção 4.2, os humanos, alguns robôs móveis e autônomos que vivem no mundo real, os "agentes-software" que residem em sistemas operacionais, base de dados, redes de computadores, e, finalmente, os agentes artificiais, que estão numa tela ou memória de computador. O que esses agentes compartilham que constitui a essência de ser um agente?

Destaca-se que cada um deles faz parte de e/ou se situa em algum ambiente. Sentem esse ambiente e agem, com autonomia, sobre o mesmo. Atuam de modo que suas ações atuais possam afetar seus sensores/sentidos e seu ambiente, de maneira contínua, durante algum período de tempo. Tais requisitos constituem a essência de ser agente e podem ser formalizados como a seguir:

"Um agente autônomo é um sistema situado dentro e como parte de um ambiente, sente e age sobre este, ao longo do tempo, em busca de cumprir sua própria agenda e de maneira que afete o próprio sentido no futuro." (Franklin, Graesser, 1996).

4.3 - CLASSIFICAÇÃO DE AGENTES

A Tabela 4.1 apresenta uma possível classificação através das propriedades atribuídas a um agente:

TABELA 4.1- CLASSIFICAÇÃO, POR PROPRIEDADES, DOS AGENTES

Propriedades	**Outros Nomes**	**Significado**
Reativo	Sensível e atuante	Responde, em tempo hábil, a mudanças no ambiente
Autônomo		Exercita controle sobre suas ações
Orientado a metas	Atividade a um propósito	Não age simplesmente em resposta ao ambiente
Tempo contínuo		Processo de execução contínua
Comunicativo	Hábil socialmente	Comunica-se com outros agentes, inclusive, talvez, com pessoas
Aprendizado	Adaptativo	Altera o comportamento diante de experiência prévia
Mobilidade	Mensageiros	Apto para se locomover de uma máquina para outra
Flexível		Ações não seguem um roteiro
Caráter		Personalidade e estado emocional confiáveis

FONTE: Franklin, Graesser, 1996.

Os agentes podem ser, então, classificados de acordo com o subconjunto das propriedades que satisfazem, ou seja, de acordo com as tarefas que cumprem, pelo alcance e sensibilidade de seus sensores, pelo alcance e efetividade de suas ações, pela quantidade de estado interno que possuem ou ainda pelo ambiente no qual o agente atua e "vive".

4.4 - TENDÊNCIAS ATUAIS PARA A UTILIZAÇÃO DE AGENTES

A indústria de agentes ainda se apresenta em um estado embrionário. Assim, a abrangência da tecnologia e dos sistemas baseados em agentes ainda se encontra em crescimento. Atualmente, diversas classes de agentes têm sido desenvolvidas, entre as quais se destacam aquelas evidenciadas no "Green Paper" (OMG, 2000) produzido pelo "Agent Working Group – OMG (Object Management Group)" sobre a Tecnologia de Agentes.

1) Agentes para Gerenciamento de Sistemas e Redes: As companhias de telecomunicações lideram o "ranking" dos setores mais ativos na área e, sem sombra alguma de dúvidas, surgiram como grupo mais comprometido com o paradigma de agentes. O objetivo principal dessas companhias visa a utilização dos agentes para assistência às complexas tarefas de gerenciamento de seus sistemas e rede, tais como, mecanismos de antecipação às falhas, análises de problemas e síntese de informações.

2) Agentes "Interest matching": Provavelmente, trata-se da classe de agentes mais utilizada e, por trabalharem às ocultas, esses agentes atuam sem o conhecimento da maior parte dos usuários que se utilizam dos seus serviços. Os agentes de "combinação de interesses" – "interest matching" – são adotados por "sites Web" comerciais para oferecer recomendações. Por exemplo, se determinado internauta aprecia obras como "*Frank Sinatra's Greatest Hits"*, este mesmo consumidor em potencial poderia desejar um trabalho musical como *"Tony Bennett's Songs for a Summer Day"*. Esses agentes observam os padrões de interesse que podem ser utilizados na elaboração de sugestões/recomendações. Destaca-se a presença de tais agentes no "site" Amazom.com e em diversos outros "sites" de áudio e vídeo.

3) Agentes Assistentes para Usuário: Estes agentes, às vezes apresentados como "conselheiros-cartoon", operam ao nível de interface de usuário (User Interface), proporcionando informações ou avisos/conselhos para os usuários. Companhias como Microsoft, Lotus e Apple têm mostrado grande interesse nessa área. Como exemplo mais conhecido e publicamente difundido, na forma de figuras animadas de "help" dos produtos Microsoft Office, esses agentes utilizam "bayesean networks" para análise e predição de tópicos, sobre os quais os usuários possam solicitar auxílio.

4.5 - TIPOS DE APLICAÇÕES DE AGENTES

Os tipos de aplicações que empregam agentes ainda são limitados. À medida que os conceitos sejam progressivamente assimilados e que se perceba o surgimento de novas ferramentas, a abordagem baseada em agentes será naturalmente adotada nas aplicações da Tecnologia da Informação. Os principais destaques, verificados (OMG, 2000) no âmbito de aplicações da Tecnologia de Agentes, podem ser agrupados como apresentado a seguir:

1) Controle de Processos: Edifícios Inteligentes (por exemplo, aquecimento/ refrigeração e segurança inteligentes - "smart"); gerenciamento de aparelhagem (por exemplo, em refinarias); robôs.

2) Agentes Pessoais (PDA – Personal Digital Agents): "Filtros" de e-mail e notícias; gerenciador de Agenda Pessoal; secretárias automáticas pessoais.

3) Tarefas de Gerenciamento de Informação

a) **Pesquisa/Busca de Informação:** A quantidade gigantesca de informação disponível nas "entranhas" de uma intranet corporativa impõe barreiras à capacidade da maior parte dos usuários quanto à recuperação efetiva de informação útil. Agentes de busca possuem conhecimento sobre diversas fontes de informação. Tal conhecimento abrange os tipos de informação disponíveis em cada domínio, métodos para acessar determinadas informações e noções sobre a segurança e precisão das fontes de informação. Esses agentes utilizam tal habilidade na execução de tarefas de busca específicas.

b) **Filtragem de Informação:** Outra tarefa comum para agentes. Agentes de "filtragem" lidam com um conhecido problema de sobrecarga, limitando ou selecionando o montante exagerado de informações que chega para um determinado usuário. A idéia básica se resume em desenvolver um substituto on-line com inteligência suficiente sobre as necessidades – quanto ao tipo de informação do usuário. Tendo, portanto, a capacidade de selecionar apenas documentos de interesse. Essa classe de agentes, em geral, funciona como porteiros eletrônicos, evitando uma "inundação" desnecessária de "lixo informativo".

c) **<u>Monitoramento de Informação:</u>** Muitas tarefas dependem da notificação imediata ou em tempo hábil sobre alterações em diferentes fontes de dados. Um operador responsável pelo planejamento de logística (planner), por exemplo, elabora um plano de transporte para um equipamento de uma certa localização para outra, mas o andamento deste plano pode ser atrapalhado em virtude de uma abrupta alteração climática (uma tempestade), durante uma parada para reabastecimento. O "planner" gostaria de ter conhecimento, o mais rápido possível, a respeito de qualquer evento que, como o imprevisto citado, pudesse afetar seu plano. Os agentes são úteis no monitoramento de fontes de dados distribuídas e, como entidades-software, desenvolvem um comportamento paciente e repetitivo para constantemente monitorar as alterações em fontes de dados. Alternativamente, agentes móveis podem ser despachados para localizações remotas e/ou inacessíveis inspecionando fontes de dados às quais um usuário, normalmente, não tem acesso.

4.6 AGENTES MÓVEIS: O FUTURO DA COMPUTAÇÃO DISTRIBUÍDA

Os agentes móveis, também conhecidos com *agentes mensageiros*, compreendem a definição de objetos que possuem comportamento, estado e localização. Tais agentes são autônomos, visto que, uma vez invocados, decidem com autonomia a respeito de quais localizações serão visitadas e de quais instruções devem ser executadas. Este comportamento é definido implicitamente através do código do agente, ou alternativamente, especificado por um itinerário (modificável em tempo de execução) – (Wong, Paciorek, Moore, 1999).

Os agentes são caracterizados como móveis desde que possam migrar entre localidades que disponibilizam basicamente o ambiente para a execução desses agentes representando uma abstração da rede de computadores e do sistema operacional. O ciclo de vida de um agente é fundamentalmente determinado por um conjunto de eventos relacionados: Os eventos de criação e destruição (disposal) podem ser comparados ao escopo dos "constructor" e "destructor" definidos para um objeto. O evento de despacho (disparo para migração) sinaliza que o agente se encontra em fase de partida para uma nova localização. O evento de chegada indica que o agente atingiu, com sucesso, a nova

localização. O evento de comunicação notifica ao agente a necessidade de manipulação de mensagens recebidas de outros agentes.

4.7 - AGENTE ESTACIONÁRIO OU FIXO

A Mobilidade é uma propriedade ortogonal dos agentes. Isto é, todos agentes não são necessariamente móveis. Um agente pode apenas fixar-se e comunicar-se com o ambiente por meios convencionais, tais como várias formas de RPC (Remote Procedure Calling) e "messaging". Denominam-se *agentes estacionários* aqueles que não podem se mover ou não querem.

Definição de um agente estacionário : Um agente estacionário executa apenas em sistemas nos quais iniciou sua execução. Se ele necessita de informação, que não se encontra nesse sistema, ou precisa de interagir com um agente em um sistema diferente, tipicamente usa um mecanismo de comunicação, tal como RPC.

Em contraste, um agente móvel não está confinado ao sistema onde iniciou sua execução. O agente móvel dispõe de liberdade para "viajar" entre os "hosts" de uma rede. Criado em um ambiente de execução, pode levar/transportar seu código e estado para outro ambiente na rede, no qual recomeça/retoma sua execução. Pelo termo "estado", tipicamente se consideram os valores-atributos do agente que ajudam a determinar como proceder; assim, que recomeça sua execução no ambiente de destino. Pelo termo "código" assume-se, dentro do contexto orientado a objetos, o código da classe necessário para execução do agente(Hohl, Klar, Baumann, 1997).

4.8 - OS BENEFÍCIOS POTENCIAIS DA TECNOLOGIA DE AGENTES MÓVEIS

A motivação e o interesse pelos agentes móveis não devem se fundamentar na tecnologia *per se*, e sim nos benefícios alcançados no desenvolvimento de sistemas distribuídos. Desta forma, enumeram-se as possíveis vantagens para tal abordagem na Tabela 4.2 apresentada a seguir (White, T.; Bieszczad, A.; Pagurek, B, 1998).

TABELA 4.2- BENEFÍCIOS POTENCIAIS DA UTILIZAÇÃO DE AGENTES MÓVEIS.

Benefícios Potenciais da Tecnologia de Agentes Móveis	
Redução no tráfego de rede	Os sistemas distribuídos freqüentemente dependem dos protocolos de comunicação, que envolvem múltiplas interações no cumprimento de uma determinada tarefa. Ainda mais quando medidas de segurança são vitais. O resultado é o enorme tráfego na rede. Os agentes móveis permitem que se "empacote" e se despache uma transação para um "host-destino", onde as interações se desenrolem localmente. A utilidade dos agentes móveis se percebe quando chegam a reduzir o fluxo de dados brutos (raw data) na rede. Quando armazenados, em grandes volumes, em "hosts" remotos, os dados deveriam ser processados localmente, evitando-se a transferência através da rede. O lema é simples: levar a computação aos dados, ao invés de levar os dados para a computação**.**
Superação (overcoming) à latência das redes de computadores	Sistemas críticos de tempo-real, tais como robôs em processos de manufatura necessitam responder a mudanças no ambiente em tempo hábil. Controlar tais sistemas sobre uma rede industrial de tamanho substancial envolve latências significantes e inaceitáveis. Agentes móveis oferecem uma solução, visto que podem ser destacados de um controle central para agir localmente e diretamente executar as orientações do controlador.
Execuções assíncrona e autônoma	Em geral, dispositivos móveis dependem de conexões de rede frágeis e caras. Ou seja, tarefas que requerem uma conexão contínua entre um dispositivo móvel e uma rede fixa tornam-se provavelmente inviáveis em Termos técnicos e econômicos. Tarefas podem ser embutidas em agentes móveis, as quais, então, são despachadas pela rede. Após o despacho, os agentes independem do processo de criação e podem operar com autonomia e assincronia. O dispositivo móvel pode se reconectar, mais tarde, para "recolher" o agente.
Adaptação dinâmica	Agente móveis possuem a habilidade de sentir seu ambiente de execução e reagir às mudanças com autonomia. Múltiplos agentes móveis apresentam a capacidade de auto-distribuição entre os "hosts" de uma rede, de maneira tal a manter a configuração ótima para satisfazer requisitos ou solucionar problemas particulares.
Suporte, por natureza, heterogêneo	A computação em rede é fundamentalmente heterogênea de ambas as perspectivas de hardware e software. Como os agentes móveis independem, geralmente, de máquina e camada de transporte, considerando apenas seu ambiente de execução, fornecem condições ótimas para integração sólida de sistemas. O "framework" de mobilidade desobriga o vínculo de amarração dos agentes móveis a um host determinado. Os agentes podem atuar em qualquer local em que esteja instalado um framework. Os custos de execução de uma máquina virtual Java (JVM), em um dispositivo, apresentam taxas decrescentes.

(CONTINUA)

TABELA 4.2 - CONCLUSÃO

Robustez e tolerância às falhas	Reagindo dinamicamente a situações e eventos desfavoráveis, os agentes móveis facilitam a concepção de sistemas distribuídos robustos e tolerantes às falhas. Se um sistema apresenta sinais de mal funcionamento, os agentes móveis podem ser utilizados para aumentar a disponibilidade de certos serviços nas áreas relacionadas. Por exemplo, a densidade de agentes de detecção de falhas e agentes de reparos pode ser incrementada. Algum tipo de gerenciamento de agentes é requerido, a fim de garantir que o sistema baseado em agentes cumpra seu propósto. Em outro cenário possível, se um nó de uma rede está na iminência de ser "derrubado" (shut-down), todos os agentes em execução nessa máquina, serão advertidos e, em tempo hábil, despachados para outro nó, dando continuidade às suas operações.
Interação com Sistemas de tempo-real	A instalação de um agente móvel próximo a um sistema de tempo-real pode prevenir/evitar atrasos (delay) provenientes do congestionamento de rede. Em sistemas de gerenciamento de rede (Network Management Systems - NMS), agentes network-manager (NM) atuam *in loco* (próximos aos componentes de hardware / software), de modo que esta vantagem pode não ser tão evidente como as outras.
Atualizações de software / Extensão (on-line) de serviços	Um agente móvel pode ser atualizado virtualmente, de acordo com a necessidade. Em contraste, a funcionalidade de swapping de servidores é complicada; especialmente se existe o propósito de se manter um nível apropriado da qualidade de serviço (QoS). Os agentes móveis podem ser utilizados para estender as capacidades de aplicativos, por exemplo, disponibilizando serviços. Isso permite o desenvolvimento de sistemas extremamente flexíveis.
Paradigma de desenvolvimen- to conveniente	A criação de sistemas distribuídos e flexíveis pode ser relativamente fácil. A grande dificuldade se relaciona com a exigência de um "framework" para mobilidade. Ambientes RAD (rappid application development) de alto nível são necessários à medida que a área (agentes móveis) amadurece. É muito provável que as ferramentas consagradas de programação orientada a objetos evoluirão para ambientes de desenvolvimento orientado a agentes, os quais incluirão funcionalidades específicas que facilitarão a mobilidade do agente.

FONTE (White, T.; Bieszczad, A.; Pagurek, B,1998).

CAPÍTULO 5

EVOLUÇÃO DO SISTEMA PARA CONTROLE DE SATÉLITES

Propostas genéricas não decidem casos concretos. (Oliver Wendell Holmes)

A grande extensão do País e a existência de imensas áreas com baixa densidade populacional, especialmente na Região Amazônica, são características predominantes para justificar o uso de satélites como instrumento de integração do território nacional, através de suas redes de comunicação, serviços de previsão de tempo e acompanhamento dos processos de uso do solo. Estas condições fisiográficas, aliadas à prática adquirida no estudo e na utilização de técnicas espaciais, foram motivos suficientes para se iniciar um programa de desenvolvimento de tecnologia espacial no Brasil.

O Instituto Nacional de Pesquisas Espacias (INPE), como uma das principais organizações envolvidas na evolução tecnológica espacial brasileira, assumiu a responsabilidade do lançamento e controle dos primeiros satélites brasileiros. O programa espacial brasileiro compreende o lançamento de quatro satélites, sendo os dois primeiros utilizados para coleta de dados e os outros, para sensoriamento remoto.

Os satélites de coleta de dados são considerados satélites de órbita baixa (aprox. 750 km), ficam em média doze minutos visíveis pelas estações de rastreio e gastam em média 90 minutos para dar uma volta em torno da terra. Os satélites geoestacionários diferem dos satélites de coleta de dados, principalmente pela altura (36000 km) e o tempo de visibilidade pelas estações de rastreio, no caso 24 horas, já que eles acompanham o movimento da terra.

O ciclo de vida de um satélite compreende etapas que vão desde a fase de lançamento até a fase de rotina. A fase de lançamento é considera a mais crítica, devido a problemas que possam ocorrer com o próprio veículo lançador. A próxima fase (aquisição) é coberta de expectativa pelos engenheiros de satélites, já que são captados os primeiros sinais do satélite. Em seguida, a fase de aceitação objetiva testar as funcionalidades de todos os subsistemas internos do satélite. Depois de um teste geral,

o controle de um satélite entra na fase de rotina. A fase de rotina é intercalada periodicamente pela fase de manobras, que consiste basicamente em alterar a atitude de um satélite (INPE, 1993).

Para gerir todas as peculiaridades inerentes ao controle de um satélite, o INPE criou uma infra-estrutura robusta, apoiada por duas estações de rastreamento e aplicativos para controle de satélites.

5.1- O SISTEMA DE RASTREIO E CONTROLE DE SATÉLITES

O sistema de rastreio e controle de satélites do INPE é constituído pelo Centro de Controle de Satélites (CCS - São José dos Campos), por duas estações terrenas remotas, uma em Cuiabá (MT) e outra em Alcântara (MA), por uma rede de comunicação de dados que interliga estas unidades (RECDAS) e, finalmente, por um software aplicativo (SICS-Sistema de Controle de Satélites), desenvolvido especialmente para controlar os satélites desenvolvidos pelo INPE (Figura 5.1).

a) O Centro de Controle de Satélites

Sua finalidade é assegurar o funcionamento perfeito do satélite, desde sua injeção em órbita, até o fim de sua vida útil. A partir do CCS, são programadas e controladas as atividades das estações terrenas. O sistema computacional do CCS se compõe de dois computadores ALPHA-DIGITAL, cujo software aplicativo (SICS) desempenha as funções listadas a seguir:

- Receber, em tempo real, das estações terrenas, os dados de telemetria, contendo informações sobre a atitude, temperaturas e parâmetros funcionais dos equipamentos de bordo do satélite, processá-los e arquivá-los. Esta função permite às equipes de controle de solo monitorar continuamente a orientação do satélite no espaço (atitude) e o seu estado de funcionamento.
- Receber das estações e arquivar os dados de localização do satélite (medidas de distância ou angulares).
- Gerar e transmitir às estações terrenas telecomandos que, quando irradiados pelas estações ao satélite, são recebidos e executados por seus sistemas de bordo. Isso

permite que se atue a partir do solo no satélite para a reconfiguração do estado de funcionamento de seus equipamentos, ou execução de manobras de controle de atitude ou órbita.

- Monitorar o estado de funcionamento dos equipamentos residentes nas estações terrenas.

As atividades do CCS são desenvolvidas em instalações apropriadas, para cada fase da vida do satélite em órbita. Assim, nas fases críticas (lançamento ou emergência), as atividades de controle são executadas a partir da sala de controle principal. Em fase de rotinas, as atividades são executadas a partir de uma sala de controle dedicado ao satélite em questão.

Na Fase de Lançamento de Órbitas Iniciais (FLOI), que considera, além do caso nominal os principais casos de contingências, as atividades de controle seguem procedimentos desenvolvidos previamente ao seu lançamento. Esses procedimentos constituem o chamado Plano de Operações de Vôo para o FLOI. Na fase de rotina, devido à repetibilidade das operações de controle, o plano de vôo é gerado periodicamente, de maneira automática, com o auxílio de um programa computacional, desenvolvido para essa finalidade.

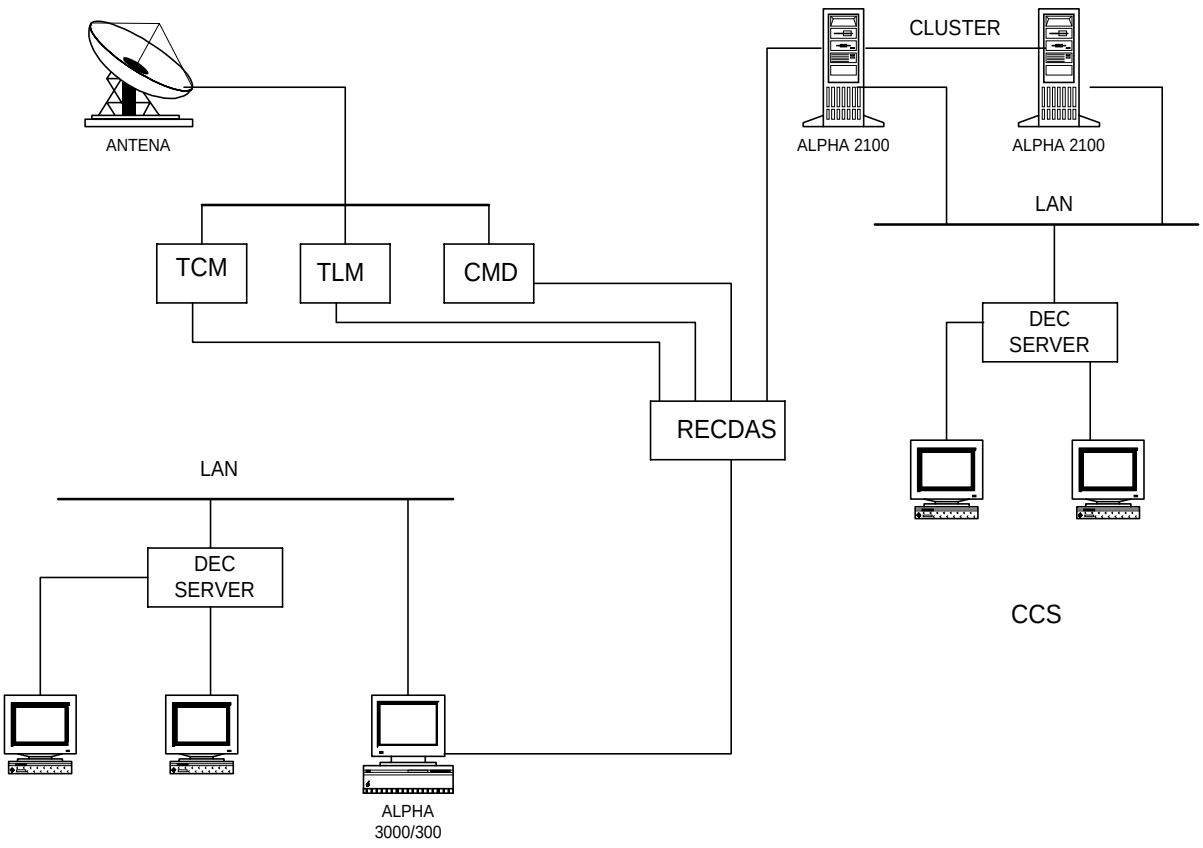


Fig. 5.1- Arquitetura simplificada do SICS.

FONTE: INPE,1989

b) Estações terrenas

As estações terrenas apresentam-se como elo de ligação entre o Centro de Controle e o satélite em órbita. Suas atividades básicas são:

- Adquirir o sinal do satélite e segui-lo durante sua passagem sobre a estação.
- Extrair do sinal recebido os dados do estado dos equipamentos de bordo, datá-los e encaminhá-los ao CCS.
- Irradiar para o satélite os telecomandos recebidos do CCS, no horário determinado.

- Efetuar as medidas de localização (distância do satélite até a estação e ângulos de azimute e de elevação do satélite em relação à estação), datá-las e encaminhá-las ao CCS. Esses dados são processados no CCS para determinar a órbita do satélite.

c) Rede de Comunicação de Dados (RECDAS)

Esta rede interliga o CCS às estações terrenas e apresenta as seguintes características:

- É uma rede privada de comunicação de dados que implementa o protocolo X.25 de acesso.
- Constituída por três nós, um em cada local, e por um centro de gerenciamento de redes localizado no CCS.
- Implementa a topologia em estrela, sendo o nó de São José dos Campos o centro da rede.
- Baseada na rede pública de comutação por pacotes (RENPAC).

d) Software Aplicativo -Versão Inicial

O desenvolvimento de software para controle de satélites no INPE se intensificou na década de 80 com o surgimento da MECB (Missão Espacial Completa Brasileira). O SICS (Sistema de Controle de Satélites) foi o primeiro sistema desenvolvido em FORTRAN 77 para atender a essas necessidades. Implantado, inicialmente em computadores VAX da DIGITAL e posteriormente migrado para computadores ALPHA da própria DIGITAL. O SICS é composto pelo Software Operacional do Satélite do Centro de Controle de Satélites (CCS), denominado Software de Controle de Satélites (SCS), que fica localizado em São José dos Campos - SP, e pelo Software Operacional de Estação Terrena (CET), que fica localizado, juntamente com outros equipamentos, em Cuiabá (MT) e em Alcântara (MA), conforme a Figura 5.1.

Descrição Funcional

O SCS faz parte do software aplicativo do Centro de Controle de Satélites e realiza as tarefas de tempo real para controle de satélites, as tarefas de comunicação com as entidades externas ao CCS e as tarefas de suporte ao processamento “off-line”. Ele

também efetua a monitoração e controle de satélites e a monitoração e controle das Estações Terrenas (ET).

O CET faz parte do software aplicativo do computador das Estações Terrenas de Alcântara e Cuiabá e tem como objetivo dar apoio nas funções de supervisão dos equipamentos das Estações, nas funções de controle de antena e nas funções de "back-up" parcial do Centro de Controle de Satélite em operações de contingências. Os principais equipamentos da Estação Terrena relacionados à comunicação com o satélite, excetuando os computadores, são mostrados na Figura 5.1.

ANTENA: Responsável pela recepção e transmissão de dados para o satélite;

TCM: Equipamento responsável pelo envio de telecomandos para o satélite;

TLM: Equipamento responsável pelo recebimento de telemetrias do satélite;

CMD: Equipamento responsável pelas medidas de distância e calibração.

Principais funções do SICS

- Prover meios para a transmissão de telecomandos, cancelamento de telecomandos a serem transmitidos, execução da validação e da verificação de comandos, através de telemetria de tempo real e gravação dos comandos enviados em um histórico.

- Prover meios para a visualização dos dados de telemetria em tempo real e a partir de um histórico; assinalar, no caso de visualização de tempo real, os dados de telemetria fora dos limites, identificados através do processamento da telemetria; visualizar todos os dados de telemetria fora dos limites de um mesmo quadro e recuperar, no caso de tempo real, os últimos eventos recebidos de visualização.

- Gerenciar a antena, permitindo a execução de uma estratégia de aquisição e rastreio do satélite durante a passagem. A estratégia de aquisição selecionada deve ser monitorada e interrompida, caso necessário.

- Calcular os dados de apontamento da antena para uma dada estratégia e enviar para o operador da antena.

- Enviar telecomando que liga o transmissor de bordo do satélite.

- Prover meios para o acompanhamento visual da órbita do satélite em tela mural do CCS.

- Prover meios para manutenção, armazenamento e recuperação de dados em arquivos históricos.

- Prover meios para a visualização, em tempo real, dos dados de supervisão dos equipamentos de uma determinada estação terrena, tanto localmente, como remotamente, no CCS. Gravar os dados de supervisão num arquivo histórico de supervisão da estação e reportar problemas críticos detectados nos equipamentos.

- Prover meios para auxiliar na operação de uma Estação Terrena e realizar funções auxiliares de operação do Segmento Solo: comunicação operador/operador, envio de dados de previsão de passagem do satélite (no CCS), reconhecer alarmes, visualizar a configuração do Segmento Solo, para que o operador tenha conhecimento das conexões ativas, ativar e desativar procedimentos de varredura da antena (na Estação Terrena), receber o estado de operação da antena, interromper um processo de aquisição, testar uma determinada estratégia de aquisição. A maioria dessas funções devem estar disponíveis tanto no CCS como nas Estações Terrenas.

- Prover meios para a solicitação e o armazenamento de medidas de distância do satélite e de medida de calibração do Conjunto de Medidas de Distância (CMD) da Estação Terrena: armazenar as medidas de distância e de calibração, que foram consideradas válidas num histórico.

- Prover meios para inicialização e atualização do relógio dos computadores do CCS e das Estações Terrenas com o horário universal (GMT).

- Prover meios para a troca de mensagens entre os componentes do SCS e demais hospedeiros da Rede de Comunicação de Dados de Satélites (RECDAS) ou das Estações de Rastreio e Controle Estrangeiras.

- Prover meios para a transferência de arquivos de dados entre o CCS e as Estações Terrenas e/ou o armazenamento de arquivos de previsão de passagem em disco para o histórico no CCS.

- Supervisionar o estado dos equipamentos de uma Estação Terrena; configurar os equipamentos da Estação para passagem, calibração e testes.

- Prover meios para o armazenamento dos dados de telemetria e o processamento desses dados em tempo real e dos dados de testes: prover meios para recuperação de alarmes, armazenamento de mensagens de telemetria, em tempo real, e do computador de bordo válidas, validação dos quadros de telemetria de tempo real e das mensagens de telemetria de testes.

Para implementar as funcionalidades listadas anteriormente, dividiu-se o SICS em subsistemas, cada um dos quais com funções bem definidas (INPE,1989):

PRE - Software de Preparação de Arquivos e Tabelas.

ALA - Software Gerenciador de Eventos.

GRF - Software de Representação Gráfica.

HIS - Software de Manutenção de Arquivos Históricos.

SCO - Software de Comunicação.

CAS - Software de Controle de Acesso.

REL - Software de Inicialização e Atualização do Relógio.

STA - Software de Transferência de Arquivos.

CMD - Software de Telecomandos.

TMS - Software de Processamento de Telemetria.

DTM - Software de Visualização de Telemetria.

RAN - Software de Processamento de Medidas de Distância.

GAN - Software Gerenciador da Antena.

OPS - Software de Operação do segmento Solo.

SUP - Software Supervisor da Estação Terrena.

MON - Software de Monitoração da Estação Terrena.

SIMULADOR – Software responsável pela simulação de um satélite.

Todos esses subsistemas apresentam-se disponíveis nas Estações (Cuiabá e Alcântara), bem como no CCS, com exceção dos subsistemas GAN e SUP, que estão disponíveis apenas nas estações. A interação dos principais subsistemas pode ser detalhada na Figura 5.2 e comentada a seguir.

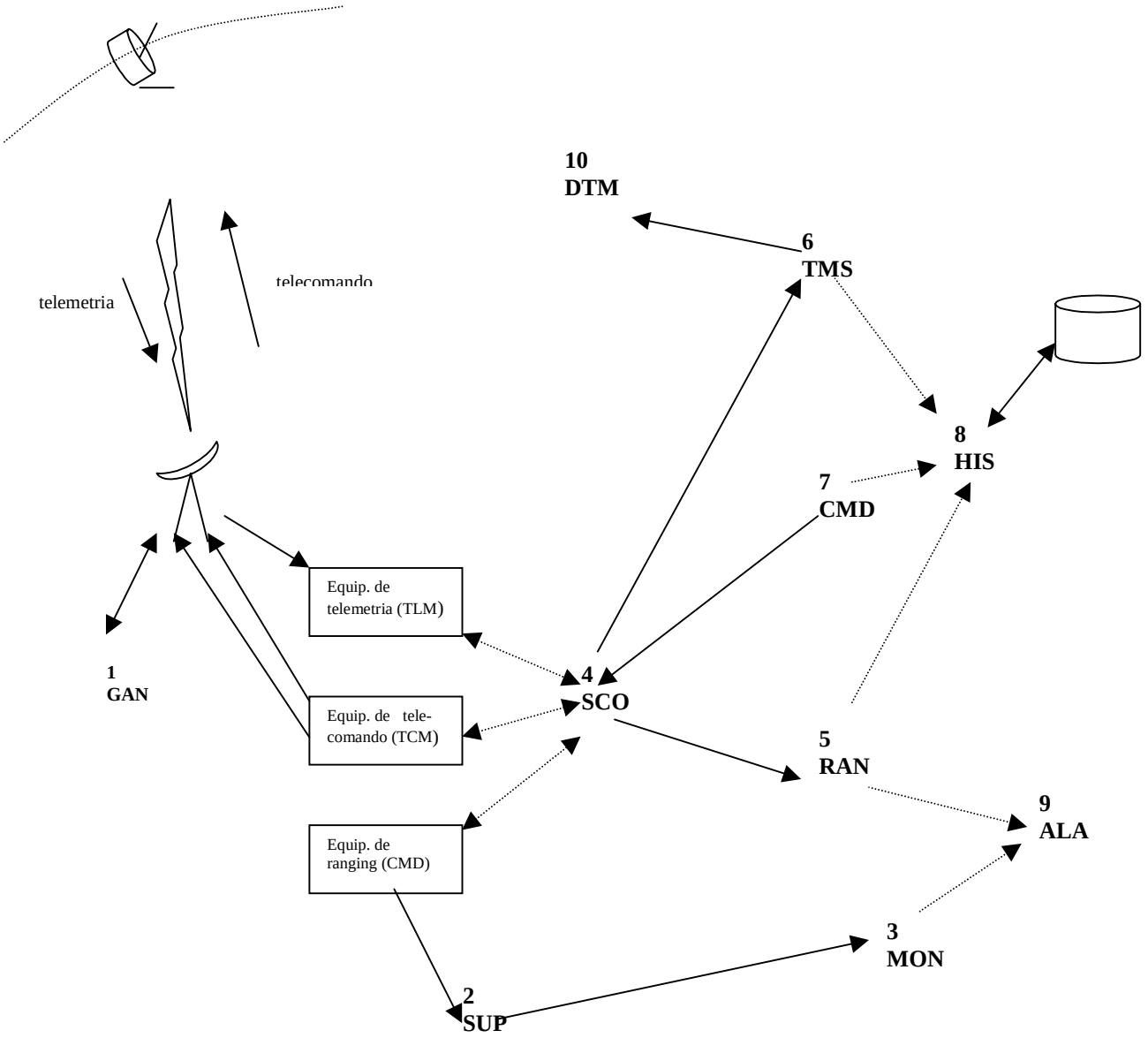


Fig. 5.2– Representação da interação entre os subsistemas.

FONTE: adaptada de INPE, (1989).

Observações:

1 – Todo o processo de recepção de dados (telemetria) e o envio de telecomandos para o satélite só é possível quando o satélite passa sobre as estações. O GAN, neste caso, assume a responsabilidade de:

- Executar uma estratégia de aquisição para tentar localizar o satélite no espaço.

- Apontar a antena para a posição da órbita do satélite.

- Recuperar as medidas angulares da antena.

- Irradiar os telecomandos para o satélite e receber as telemetrias que descrevem o funcionamento interno de um satélite.

2 -Nas estações terrenas, o SUP é o responsável pela supervisão dos equipamentos. O objetivo básico deste subsistema consiste em informar aos operadores as possíveis falhas de hardware nas estações. Ele faz a monitoração dos equipamentos de telemetria, telecomando, e "ranging" em intervalos predefinidos.

3 – O MON atua como um mediador entre os parâmetros supervisionados pelo SUP e a visualização dos mesmos para os usuários. Sua atividade básica se restringe em visualizar os dados de supervisão dos equipamentos de uma estação terrena em tempo real e a partir do histórico bem como divulgar via ALA para os operadores os equipamentos que apresentam problemas críticos.

4 – O SCO disponibiliza o meio para a troca de mensagens entre os subsistemas do SICS (TMS, RAN e CMD) com os seus respectivos equipamentos das estações (equipamento de telemetria, ranging e telecomando). O protocolo de comunicação desses equipamentos é X-25; portanto, o SCO disponibiliza um conjunto de funções para envio e transmissão de dados sobre este protocolo.

5 - O cálculo da medida de distância entre o satélite e a terra fica a cargo do RAN. Ele se conecta com o equipamento de "ranging", via SCO. Esses dados são utilizados para se fazer previsões das próximas órbitas.

6 - O TMS responsabiliza-se pelo processamento de telemetrias recebidas do satélite. A função básica desse subsistema se traduz em analisar as telemetrias recebidas do

satélite e informar ao usuário aquelas que se apresentam com valores fora do limite permitido.

7 - O CMD atua no envio de telecomandos para o satélite. As funcionalidades implementadas neste subsistema se restringem em:

- conectar com o equipamento de telecomando via SCO;
- inserir um comando básico ou uma seqüência de telecomandos a serem transmitidos para o satélite;
- autorizar e enviar um telecomando.

8 – O HIS praticamente faz interface com todos os outros subsistemas. Atribui-se ao HIS a capacidade de gerir todos os dados recebidos do satélite (telemetrias, medidas de distância do satélite etc.).

9 – O ALA assume o papel de notificar os operadores de satélites as eventuais falhas que ocorrem nos outros subsistemas. Suas funcionalidades consistem em reportar estas informações para o operador responsável pela operação do subsistema; receber as mensagens de alarmes geradas pelos outros subsistemas e encaminhá-las para os respectivos usuários de cada subsistema; manter um "log" de todas as mensagens geradas pelo sistema no banco de dados.

10 - O DTM disponibiliza para os usuários as interfaces para a visualização das telemetrias processadas pelo TMS. Através dessas interfaces, pode-se visualizar as telemetrias de um satélite em tempo real e fora de passagem.

5.2 - VERSÃO ATUAL DO SICS

A versão atual do SICS foi desenvolvida para ser utilizada no controle do primeiro satélite brasileiro de sensoriamento remoto (CBERS). O SICS atual tem uma arquitetura cliente-servidor e apresenta-se disponível em plataformas PCs. Para a implementação, utilizou-se o ambiente de desenvolvimento Visual C++ da Microsoft e a Metodologia Orientada a Objetos (Cardoso, Gonçalvez, Ambrósio, 1998) . A Figura 5.3 ilustra a arquitetura da nova versão.

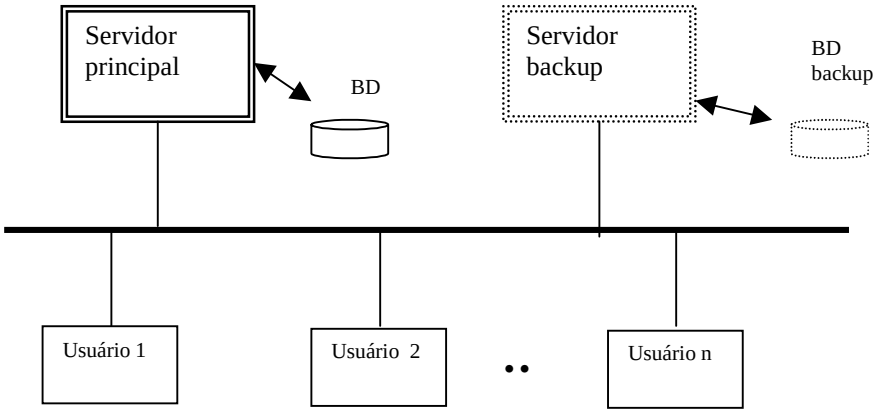


Fig. 5.3 – Arquitetura da versão atual do SICS.

FONTE: adaptada de Gonçalvez, Cardoso, Ambrósio(1998).

Por motivos de tempo e disponibilidade de pessoal, a nova versão do SICS implementou somente os subsistemas TMS, CMD e HIS. Para a operação do satélite CBERS utiliza-se a nova versão do SICS e parte da versão anterior que, até o momento, se encontra implementada nos computadores da DIGITAL.

De acordo com a versão atual, o servidor principal disponibiliza, para os usuários do sistema, os serviços de:

- processamento de telemetria(TMS);
- envio de telecomando(CMD) e
- armazenamento dos dados (telemetria e telecomandos) no banco de dados(HIS).

Existem várias categorias de usuários para controles de satélites, dentre elas pode-se ressaltar:

a) Controladores de satélites: Apresentam-se como responsáveis pela execução do plano de vôo de um satélite.

b) Engenheiros de satélites: Responsabilizam-se pelo acompanhamento dos subsistemas internos de um satélite, tais como bateria, carga útil etc.

Caso haja falha no servidor principal, o servidor de “backup” assume o controle do satélite e passa a acessar o banco de dados de “backup”, que representa uma cópia idêntica do banco de dados do servidor principal(Gonçalves, Cardoso, Ambrósio,1998).

5.3 - LIMITAÇÕES DAS VERSÕES DO SICS

Após uma longa fase de sistemas centralizados, baseados em mainframes (SICS- versão inicial), a quebra do processamento em "client-server" foi utilizada no software de controle de satélites com diversas vantagens (SICS-versão atual), tais como a distribuição do processamento, parte alocada no servidor e parte alocada no cliente; a descentralização do processamento; etc. Entretanto, pode-se ainda ressaltar algumas limitações no software utilizado no controle de satélites do INPE, conforme os itens apresentados a seguir:

a) Disponibilidade de serviços

Na versão inicial e atual do SICS, os serviços disponíveis pelo sistema foram implementados nos subsistemas. Falhas na comunicação, ou mesmo no subsistema, inviabilizam a execução de determinada tarefa. Como exemplo, uma falha no software que processa telemetria (TMS) impossibilita o processamento e a visualização das telemetrias recebidas do satélite.

b) Tolerância a falhas

A queda de um servidor em uma arquitetura centralizada ou cliente-servidor compromete todos os subsistemas ali residentes e conseqüentemente toda a operação de controle de satélites. A utilização de servidores "backups", como mecanismo de redundância utilizado na versão do SICS atual, não supera falhas ocorridas, por exemplo, em uma passagem real do satélite. Isso porque o tempo consumido para posicionar o servidor de "backup" no contexto em que se encontrava o servidor principal, pode ultrapassar o intervalo de uma passagem.

c) Interoperabilidade

Como já foi mencionado, somente os subsistemas TMS, CMD e HIS foram implementados na versão atual do SICS. Isso reforça a necessidade da utilização dos serviços prestados pela versão inicial e implementados pelos subsistemas RAN, OPS, GAN etc. A integração não foi totalmente possível entre a versão inicial e a versão atual, dado a diversos fatores, a Tabela 5.1 elucida alguns destes.

TABELA 5.1- CARATERÍSTICAS DE IMPLEMENTAÇÃO ENTRE AS VERSÕES DO SICS

Fator	**SICS inicial**	**SICS atual**
Plataforma	ALPHA-DIGITAL	PC
Sistema operacional	OPEN-VMS	Windows-NT
Sistema de armazenamento	Arquivos indexados	Banco de dados relacional
Linguagem	FORTRAN-77	C++
Metodologia	Estruturada	Orientada a Objetos
Arquitetura	Centralizada	Cliente-servidor

d) A centralização

Os ambientes da versão inicial e da versão atual do SICS apresentam-se totalmente centralizados. Atualmente uma falha no servidor pode comprometer toda a missão. O tempo de reparo de uma falha em um servidor pode ultrapassar o limite de uma passagem do satélite, que é de aproximadamente 10 minutos. Deve-se ressaltar também que a falha em um dos subsistemas, como por exemplo o subsistema TMS, implica a indisponibilidade desse serviço para todos os usuários.

e) Localização entre usuários e recursos computacionais

O acesso ao SICS, tanto na versão inicial quanto na versão atual, está restrito a controladores de satélites fisicamente localizados no Centro de Controle de Satélites. A versão inicial utiliza terminais predefinidos, como forma de acesso ao servidor principal. A versão atual utiliza um conjunto de PCs que assumem o papel de clientes, interagindo com um servidor responsável pelo gerenciamento de toda a base de dados. Em ambas as versões, exige um conhecimento prévio do usuário, de quais são os serviços oferecidos pelo sistema, bem como quais servidores que atendem a estas solicitações.

f) Flexibilidade para a atender às diferentes situações de controle

Como já foi mencionado anteriormente, a operação de controle de um satélite compreende diferentes fases, como a de lançamento, de rotina e de manobras. O número de usuários oscila de acordo com cada uma das fases. Como por exemplo, na fase de lançamento, praticamente todos os usuários envolvidos direta ou indiretamente com o missão, desejam visualizar as telemetrias recebidas do satélite. Entretanto, na fase de regime ou rotina, o controle do satélite se restringe somente aos operadores. As fases críticas (lançamento e manobra) exigem uma intervenção por parte do administrador de sistemas, que atua em um primeiro instante na busca pela disponibilização de máquinas para os usuários esporádicos e, em seguida, na instalação do software de controle.

Aplicações centralizadas ou cliente-servidor não estão preparadas para se adaptar e se moldar, de acordo com a demanda por solicitações de serviços dos usuários. Talvez o máximo que se consiga é restringir o número de usuários que acessam o sistema ao mesmo tempo.

Enfim, tanto uma aplicação centralizada, como uma aplicação cliente servidor, se apresentam de forma estática para seus usuários. Ou seja, existe um conjunto fixo de servidores, bem como um número limitado de usuários. O aumento/diminuição do número de usuários requer uma intervenção por parte do administrador de sistemas. O ideal seria ter uma aplicação flexível e dinâmica, que superasse as possíveis variações no ciclo de vida de um satélite, sem a necessidade da atuação do administrador de sistema. Assim, o objetivo básico deste trabalho de pesquisa se concentra em apresentar uma arquitetura flexível e dinâmica que pode ser aplicada ao software de controle de satélites. Maiores detalhes desta arquitetura, bem como as possíveis soluções para as limitações atuais do software de controle de satélites são apresentados no capítulo seguinte.

CAPÍTULO 6

A ARQUITETURA PROPOSTA

Um filósofo de estatura imponente não pensa em um vazio. Mesmo suas idéias mais abstratas são, em alguma medida, condicionadas pelo que é ou não conhecido na época em que ele vive. (Alfred North Whitehead)

Neste capítulo são apresentadas as idéias gerais de uma arquitetura flexível e dinâmica para objetos distribuídos, proposta ao Sistema de Controle de Satélites do INPE, a qual, oportuna e posteriormente, será referenciada neste trabalho, como arquitetura proposta ou arquitetura SICSD.

Atualmente, a tecnologia de Objetos Distribuídos divide aplicações "client-server", em componentes autogerenciáveis ou objetos que podem interoperar em ambientes de redes, com diferentes plataformas, introduzindo vantagens adicionais, dentre elas pode-se citar:

- **O aumento da disponibilidade de serviços**

A tecnologia de objetos distribuídos provê mecanismos que asseguram a disponibilidade dos objetos e, conseqüentemente, o aumento da disponibilidade de serviços, independente de falhas nos computadores. Esta característica, agregada a ambientes distribuídos, pode ser a solução para as limitações do SICS, apresentadas na seção 5.3, na letra "a".

- **A tolerância a falhas**

A tecnologia de objetos distribuídos implementa, com uma certa transparência, a localização física de um objeto. Caso ocorra uma falha no nó, onde ele se encontre instanciado, uma nova conexão pode ser estabelecida, com outro objeto que presta o mesmo serviço. A distribuição de objetos em vários nós da rede, com a capacidade de enviar telecomandos para o satélite, serve para ilustrar os mecanismos de redundância que podem ser utilizados em uma arquitetura distribuída para superar eventuais falhas.

Essa capacidade intrínseca a sistemas distribuídos pode ser a solução para a limitação ilustrada na seção 5.3, letra “b”.

- **O aumento da concorrência e desempenho**

A capacidade de instanciar cópias de um mesmo objeto, em mais de um nó da rede, apresenta-se como um fator que deve ser levado em consideração para explorar o recurso de concorrência em sistemas distribuídos, ou seja, dois ou mais usuários podem solicitar o mesmo serviço ao sistema, mas sendo atendidos por objetos instanciados em diferentes nós.

- **A interoperabilidade**

O desenvolvimento da tecnologia de distribuição levou ao uso da tecnologia “middleware”. Pode-se considerar, de forma geral, que essa tecnologia propõe a disponibilidade de um conjunto de facilidades comumente necessárias para integrar componentes (objetos ou não) num sistema distribuído. Existe, com efeito, um novo paradigma em computação, cujo foco é a interoperabilidade, entendendo-se por interoperabilidade a possibilidade de um programa, em um sistema, acessar programas e dados em outros sistemas. A tecnologia de distribuição de objetos delega a oportunidade ao desenvolvedor de software de distribuir e globalizar, de forma transparente, objetos ou programas que estejam implementados em ambientes heterogêneos. A integração das versões iniciais e atuais do SICS podem ser alcançadas com a introdução de um ”middleware” entre elas. Portanto, a tecnologia de distribuição de objetos pode ser um catalisador nesse processo, auxiliando na busca pela solução da limitação apresentada na seção 5.3, na letra “c”.

- **A descentralização**

A carga de uma aplicação distribuída não se restringe a um único computador e sim em vários. A descentralização proporciona, em paralelo, uma outra contribuição, que consiste em permitir que existam réplicas de um mesmo objeto, instanciadas em computadores distintos, aumentando a disponibilidade de serviços prestados pelo sistema. Essa capacidade inerente a objetos distribuídos pode ser a solução para limitação descrita na seção 5.3, na letra “d”.

- **Superar a separação entre usuários e recursos computacionais**

A transparência de localização de objetos constitui uma característica peculiar de sistemas distribuídos, ou seja, o usuário simplesmente solicita um serviço ao sistema, a localização de quem irá atender este serviço fica sob responsabilidade do sistema. Como por exemplo, na aplicação para controle de satélites, o usuário não precisa se preocupar com a localização física do subsistema que processa telemetria, nem a localização da base de dados para recuperar as telemetrias de órbitas anteriores. A característica da transparência de localização de objetos constitui um fator preponderante, que pode auxiliar na solução para limitação descrita na seção 5.3, na letra "e".

Tendo visto as vantagens adicionais provenientes de um sistema distribuído, o objetivo deste trabalho concentra-se em propor uma arquitetura distribuída aplicada ao software de controle de satélites. Além das vantagens provenientes de um sistema distribuído, a arquitetura proposta explora outros recursos, tais como:

- **A segurança**

A política de segurança, aplicada nas versões do SICS, se restringe ao controle de acesso de usuários através de senhas. Atualmente, com a explosão da INTERNET, a implementação de mecanismos de segurança tem sido considerada em diversos sistemas computacionais. Contudo, no contexto de sistemas distribuídos, problemas de segurança, decorrentes da própria característica de distribuição, devem ser analisados e tratados adequadamente. Além das políticas de identificação e autenticação dos usuários, definidas nas versões do SICS, a arquitetura proposta sugere implementar outras, tais como:

a) Criptografia: Para garantir a segurança no recebimento e envio de dados para o satélite.

b) Auditoria: O serviço de auditoria consiste em inspecionar o sistema para averiguar se ele está sendo utilizado de forma segura, ou se está sendo invadido por usuários não autorizados.

c) Criação de domínios de objetos: Um domínio de objeto compreende um conjunto de objetos regidos por uma mesma política de segurança.

- **Flexibilidade**

Como já foi mencionado na seção 5.3 do capítulo anterior, as aplicações centralizadas, ou cliente-servidor, não são suficientemente flexíveis para se adaptarem de acordo com a demanda de solicitações de serviços de seus usuários. Entretanto, as aplicações desenvolvidas, utilizando objetos distribuídos, também não estão preparadas para atender às oscilações que podem ocorrer durante a operação de um satélite. O objetivo deste trabalho concentra-se em capacitar a aplicação para se autoajustar de acordo com as solicitações de serviços. A replicação automática de um objeto, que presta determinado serviço, pode ser um exemplo na tentativa de sintonizar a demanda com a oferta por um determinado serviço.

A demanda pela utilização de um aplicativo para controle de satélites sofre variações durante todo seu ciclo de vida. A fase de rotina é a fase predominante durante todo o ciclo. Nessa fase, um grupo restrito de usuários, no caso os controladores de satélites, assumem a responsabilidade pelo controle e operação. Entretanto, a inércia dessa fase é quebrada pelas operações de manobra, também conhecidas como “fase de manobras”. Essas operações exigem atenção especial dos usuários de dinâmica orbital, bem como os usuários responsáveis pelos painéis solares, já que objetivam mudar a atitude de um satélite.

A flexibilidade proposta neste trabalho de pesquisa engloba a capacidade de a aplicação de controle de satélites disponibilizar um número maior de objetos, quando existe um incremento na demanda, bem como reduzir o número de objetos quando existe um decremento.

- **A busca por uma arquitetura aberta**

O rápido avanço da tecnologia vem possibilitando a construção de sistemas de computação, compostos por um grande número de processadores, ligados através de redes de alta velocidade. Esses sistemas distribuídos apresentam inúmeras vantagens sobre sistemas centralizados como escalabilidade, confiabilidade, desempenho, entre outros. O processamento distribuído aberto, o qual procura englobar a heterogeneidade de meios, proporcionando a interoperabilidade e portabilidade de aplicações entre

sistemas geograficamente separados, vem sendo estudado. A ISO (International Standard Organization), através de seu Modelo de Referência para Processamento Distribuído Aberto – Object Distributed Process(ODP) e o consórcio da OMG (Object Management Group), não tem medido esforços na tentativa de desenvolvimento desse padrão.

- **Balanceamento de carga**

O balanceamento de carga em uma arquitetura centralizada é nulo; já numa arquitetura cliente-servidor, pode-se explorar parte do processamento no cliente e parte no servidor. Por definição, em uma arquitetura distribuída, pode-se tirar maior proveito do recurso de balanceamento de carga. A arquitetura proposta define mecanismos para tentar otimizar os recursos computacionais existentes. Dentre os mecanismos podem-se citar a migração de um objeto de uma máquina saturada para uma máquina ociosa; a replicação de um objeto muito solicitado em vários nós de uma rede etc.

O balanceamento de carga tem sido um dos principais temas de discussão no desenvolvimento de sistemas distribuídos. Alguns autores como Barth (Barth, et al, 1998), Thißen(Thißen, Neukirchen, 2000) e Daval (Daval, Lacroix, Guyennet, 2000), defendem uma *distribuição estática* dos objetos. Ou seja, define-se a priori, a alocação dos objetos de acordo com a disponibilidade dos recursos computacionais. O aumento na demanda por determinado serviço implica em uma busca no sistema para identificar os objetos que atendam este serviço. O balanceamento de carga, se caracteriza, pelo fato de a solicitação de serviço ser atendida pelo objeto instanciado no nó com a maior capacidade de processamento. O balanceamento estático, por não permitir a migração, não explora em profundidade, os recursos computacionais existentes. Pois, em um determinado instante, objetos ociosos podem estar alocados em máquinas ociosas com alto poder de processamento; e objetos com alta carga de trabalho, podem estar instanciados em nós saturados.

Outros autores, como Lee (Lee, 1999), Gopal (Gopal, Vajapeyam, 1998) e Schnekenburger (Schnekenburger, Stal, 1999), sugerem o *balanceamento dinâmico,* ou seja, os objetos podem migrar de um nó para outro, de acordo com a demanda por serviços. Entretanto, não levam em consideração a possibilidade de replicar objetos. A

capacidade de replicar objetos, em um ambiente dinâmico, constitui um fator tão importante quanto migrá-los. As réplicas, além de aumentar a disponibilidade do sistema, garante a continuidade das operações em caso de falhas.

As características de flexibilidade e dinamismo propostos na arquitetura estão fundamentadas na capacidade de migrar e replicar os objetos da aplicação de controle de satélites. Maiores detalhes dessa arquitetura são apresentados nos próximos itens.

6.1 - MODELAGEM DA APLICAÇÃO

A estrutura computacional existente hoje para controle de satélites no INPE é heterogênea. A primeira versão do SICS está instalada em estações de trabalho da DIGITAL utilizando o sistema operacional VMS. A segunda versão foi desenvolvida utilizando PCs e o sistema operacional Windows NT. A escolha do sistema operacional Windows NT foi um decisão política. A versão anterior do SICS estava instalada em sistemas "mainframe" da DIGITAL, os quais tinham alto custo de manutenção. Decidi-se na época adotar PCs como plataforma de desenvolvimento para baixar os custos. A escolha do sistema operacional foi um reflexo da parceria entre o Brasil e a China para desenvolvimento do CBERS. Os chineses desenvolveram todo o aplicativo utilizando Windows NT e para manter a compatibilidade de operação do satélite entre os dois países, adotou-se também o mesmo sistema operacional.

A versão proposta, nesse trabalho de pesquisa, prevê a implementação de um protótipo dessa proposta, em plataformas abertas(utilização de PCs com o sistema operacional LINUX); entretanto, necessita de fazer interface com os atuais sistemas legado para controle de satélites.

Analisando as três possíveis soluções, citadas anteriormente no capítulo 3 para a implementação do sistema distribuído, chega-se as seguintes decisões:

- Para a implementação do protótipo, da arquitetura distribuída proposta, não deve-se utilizar DCOM, já que é um produto proprietário Microsoft.
- A utilização da arquitetura RMI também foi descartada, já que é uma forma de distribuição exclusiva da linguagem java. A arquitetura proposta prevê a interface com os sistemas legados existentes para controle de satélites.

- A arquitetura proposta deve ser desenvolvida utilizando arquiteturas que implementem o padrão CORBA. Primeiro, para permitir a interface da arquitetura proposta com os sistemas legados para controle de satélites, segundo, para permitir a implantação deste sistema em plataformas abertas. Portanto, utilizou-se um "Midlleware" que segue o padrão CORBA (Orfali, Harkey, 1997; Montez, Oliveira, Fraga, 1998).

A arquitetura proposta modela a aplicação para controle de satélites em objetos e os distribui dentro de um domínio de rede predefinido. A rotina de carga do sistema define a localização inicial desses objetos. A partir desse momento, eles podem migrar ou serem replicados, de uma máquina para outra, de acordo com a demanda por solicitação de serviço, ou seja, apresentam um comportamento dinâmico.

A arquitetura proposta não mantém a divisão do SICS em subsistemas. Esta divisão, projetada nas versões iniciais do SICS, foi uma divisão funcional, como por exemplo o subsistema TMS responsabiliza-se pelo processamento de telemetria, o CMD, pelo envio de telecomandos para o satélite etc.

Os objetos da Figura 6.1 são exemplos baseados no atual sistema de controle de satélites e servem para ilustrar o funcionamento da arquitetura proposta; mas estudos futuros poderão indicar uma forma diferente de dividir o sistema e determinar um conjunto de diferentes objetos.

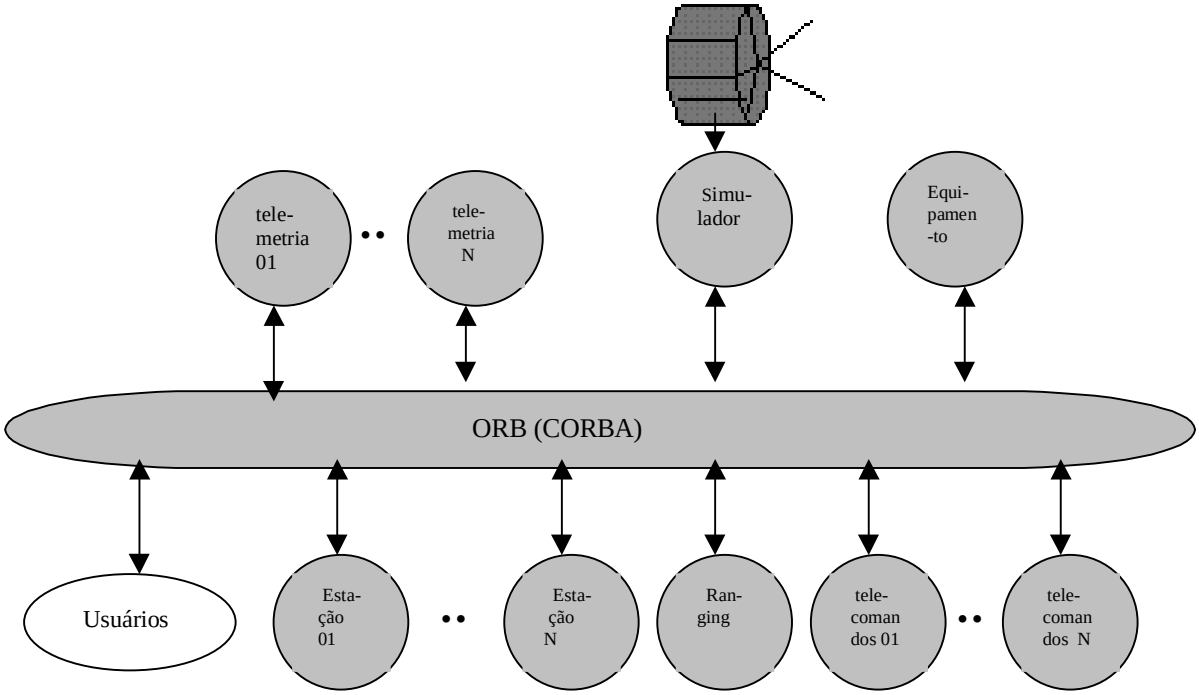


Fig. 6.1- Arquitetura SICSD.

Na arquitetura SICSD, os objetos da aplicação se comunicam através de um "middleware", que implementa a especificação CORBA, podendo existir mais de uma cópia de um objeto instanciado em nós diferentes da rede, conforme detalhamento a seguir.

- ***Telemetria***: Este objeto encapsula o estado interno do satélite, como por exemplo, a voltagem de uma determinado circuito, o posicionamento de uma determinada chave (ligado ou desligado), como também os métodos inerentes ao objeto, tais como processar telemetria, visualizar etc. Esse objeto incorpora os serviços prestados pelo subsistema TMS (processar telemetria), bem como os serviços prestados pelo subsistema DTM(visualizar telemetria), implementados nas versões anteriores do SICS.

- ***Telecomando***: Este objeto encapsula os telecomandos que podem ser enviados para o satélite, bem como os métodos pertinentes ao objeto, tais como enviar telecomando, visualizar telecomandos etc. Este objeto agrega as funcionalidades do subsistema CMD, implementado nas versões anteriores do SICS.

- ***Estação***: Este objeto contém a descrição das estações utilizadas para recepção dos dados dos satélites, que são controlados pelo INPE. Atualmente existem duas estações: Cuiabá e Alcântara. Como a arquitetura proposta modela em objetos a aplicação para controle de satélites, atribui-se a este objeto os parâmetros pertencentes a uma estação, tais como latitude, longitude etc.
- ***Ranging***: Este objeto encapsula as medidas de distância do satélite em relação a terra, bem como os métodos responsáveis por esse cálculo. Este objeto substitui as funcionalidades implementadas no subsistema RAN, nas versões anteriores do SICS.
- ***Equipamento***: Contém a descrição dos equipamentos instalados para a recepção e transmissão de dados para o satélite. Esse objeto encapsula os atributos dos equipamentos utilizados para controle de satélites, tais como frequência, potência etc.
- ***Simulador***: Este objeto simula os possíveis estados de um satélite e disponibiliza esses dados para todos os outros objetos através de uma interface. Entende-se por "estado" de um satélite as possíveis condições internas que ele pode se encontrar em um determinado instante. A capacidade de simular a falha em um determinado circuito elétrico, ou mesmo o posicionamento de uma chave (ligado ou desligado), devem ser atribuições do objeto simulador. Como já foi mencionado, através das telemetrias tem-se um panorama do funcionamento interno de um satélite e com o auxílio dos telecomandos pode-se alterar o estado de um satélite (ligar ou desligar uma determinada chave). Portanto, além do estado de um satélite, esse objeto encapsula os métodos para visualizar estes estados (recuperar telemetria) ou modificá-los (enviar telecomando).

Os usuários interagem com os objetos distribuídos através de um "middleware". O "Broker" atende às solicitações de serviços dos usuários e se incumbe de localizar o objeto capaz de atendê-las.

A replicação de um objeto em mais de um nó da rede está relacionado com o número de usuários que utilizam os serviços disponíveis desse objeto, bem como a necessidade da disponibilidade de um determinado serviço em caso de falha. Por exemplo, o objeto telemetria apresenta-se como o mais solicitado, porque existem vários usuários interessados em visualizar o estado interno do satélite. Os usuários responsáveis pelos painéis solares desejam visualizar as telemetrias que mostram a posição do satélite em relação ao sol. Os usuários responsáveis pela bateria estão interessados em acompanhar a voltagem e a corrente gerada pela mesma. Outros objetos, como por exemplo o telecomando, também devem ser replicados para promover a disponibilidade de serviços; entretanto, somente um usuário a cada instante pode utilizar seus serviços. Ou seja, não é permitido o envio de telecomandos para o satélite', de pontos diferentes da rede, ao mesmo tempo.

A arquitetura SICSD compreende os objetos da aplicação para controle de satélites (telemetria, "ranging", telecomandos etc.) e os serviços de Agentes, Persistência, Segurança e Balanceamento. A Figura 6.2 apresenta uma visão da arquitetura SICSD com os seus serviços disponíveis.

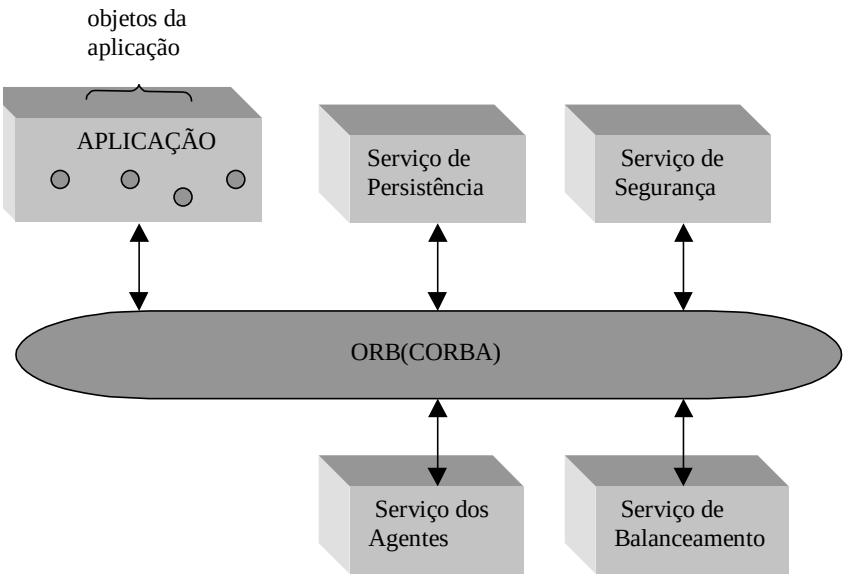


Fig. 6.2 Uma visão dos serviços da arquitetura SICSD.

A opção pela divisão da arquitetura SICSD em serviços surgiu como forma de criar uma infra-estrutura capaz de suportar uma *arquitetura flexível e dinâmica*. A definição destes quatro serviços foram suficientes para desenvolver a arquitetura proposta. Isto não descarta a possibilidade da incorporação de novos serviços, caso seja necessário agregar novas funcionalidades.

A arquitetura proposta incorpora todas as funcionalidades presentes nas versões anteriores do SICS e agrega novas capacidades. Fazendo um paralelo entre a arquitetura proposta e as versões anteriores do SICS, pode-se notar algumas semelhanças e/ou diferenças:

- A arquitetura proposta modela a aplicação para controle de satélites em objetos, enquanto as versões anteriores modelaram a aplicação, de forma estruturada, dividindo-a em subsistemas com funcionalidades bem definidas. Como exemplo, na versão anterior, existe um processo para enviar telecomando. Na arquitetura proposta, este processo está encapsulado no objeto telecomando.
- O subsistema HIS, nas versões anteriores do SICS, apresenta-se como o responsável pelo gerenciamento do sistema de armazenamento de dados. A arquitetura proposta sugere o serviço de persistência para manter persistente os objetos da aplicação.
- O serviço dos agentes da arquitetura proposta tem funcionalidades similares ao gerenciador de eventos das versões anteriores do SICS (ALA). Ambos assumem o compromisso de monitorar eventuais falhas do sistema.
- O serviço de segurança e balanceamento incrementam as funcionalidades de uma aplicação para controle de satélite. O primeiro cria uma infra-estrutura segura para os objetos da aplicação. Já o segundo procura explorar, de uma forma otimizada, os recursos computacionais disponíveis. Os detalhes desses serviços estão descritos a seguir.

O requisito segurança tem sido considerado em diversos sistemas computacionais. Contudo, no contexto de sistemas distribuídos, problemas de segurança, decorrentes da própria característica de distribuição, devem ser analisados e tratados adequadamente. A funcionalidade presente em sistemas distribuídos, oferecida pela interconexão de

elementos computacionais, através de uma rede de comunicação e pela abertura das interfaces de componentes do sistema, faz surgir problemas, tais como a manutenção de privacidade e integridade de dados que trafegam pela rede e a garantia de confiança no acesso aos componentes de software. Portanto, optou-se por criar um serviço de segurança capaz de implementar políticas, que estabeleçam níveis e critérios de segurança adequados ao funcionamento do sistema.

O serviço de segurança responsabiliza-se por garantir o acesso ao sistema somente de pessoas previamente autorizadas, bem como garantir que os usuários somente tenham acesso às funções previamente definidas, de acordo com o seu perfil. Como por exemplo, as funções disponíveis pelo sistema, para um engenheiro de satélite, são diferentes das funções disponíveis para um controlador de satélites.

A arquitetura SICSD permite que objetos migrem de uma máquina para outra; portanto, mecanismos de segurança devem ser utilizados para garantir que somente objetos autorizados possam migrar para máquinas remotas. Os detalhes deste serviço são tratados no Capítulo 10.

A arquitetura SICSD prevê a distribuição dos objetos da aplicação de controle de satélites, em um domínio de rede pré-definido. Portanto, mecanismos de monitoração do estado destes objetos, bem como o status da rede como um todo, devem ser suportados. Atualmente, os agentes apresentam-se como o recurso de software capaz de suprir as necessidades anteriormente apresentadas. Ou seja, os agentes caracterizam-se como entidades de software, capazes de sentir o ambiente onde eles estão inseridos e agir com uma certa autonomia sobre o mesmo.

Partindo dessas premissas, a arquitetura proposta conta com um serviço dos agentes capaz de manter atualizada e distribuída todas as informações necessárias para se desenvolver uma aplicação dinâmica e flexível para controle de satélites. Dentre o conjunto de informações gerenciadas pelo serviço dos agentes, pode-se ressaltar:

- a localização física de cada objeto instanciado;
- o estado de um objeto, podendo ser “ready”, se o objeto mostra-se disponível para receber solicitações de serviços do sistema, ou “failed”, caso contrário;

- a disponibilidade de CPU de um determinado nó etc.

De acordo com a arquitetura proposta, os agentes devem ser instanciados em cada nó da rede, com o objetivo de monitorar todos os objetos ali residentes. O conjunto de informações geridas pelo serviço dos agentes são armazenadas na *base de configurações*. Os detalhes deste serviço são tratados no Capítulo 7.

Com base nas informações geridas em cada nó, pelo serviço dos agentes, tem-se o perfil do ambiente predefinido para a execução da arquitetura SICSD. Não basta, no entanto, conhecer os status dos objetos distribuídos, a disponibilidade de CPU em um determinado nó etc. É preciso agir, tentando corrigir disparidades em relação à distribuição de carga de CPU do sistema, bem como à disponibilização de serviços que tem uma crescente demanda.

Atribui-se a responsabilidade de resolver o problema citado ao serviço de balanceamento, capaz de tomar decisões que podem resultar na migração ou na replicação dos objetos da aplicação para controle de satélites. Esse serviço responsabiliza-se por garantir os recursos de mobilidade, a flexibilidade e o dinamismo da arquitetura proposta. O recurso de mobilidade agrega-se à arquitetura graças à atuação do serviço de balanceamento que atua como o responsável por mover objetos de máquinas saturadas para máquinas ociosas. A característica dinâmica da arquitetura proposta se deve ao fato de o serviço de balanceamento migrar objetos de uma máquina para outra, conforme as solicitações dos usuários e do estado da rede de computadores disponível para o controle de satélites. A capacidade de replicar objetos que estão sendo muito solicitados em máquinas remotas torna o sistema flexível, já que o sistema se molda de acordo com a demanda de solicitações de serviços de seus usuários.

O processo de migração normalmente ocorre com o objetivo de fazer um balanceamento de carga do sistema. O processo de cópia de um objeto normalmente ocorre com o objetivo de disponibilizar serviços para o sistema. Os detalhes desse serviço são tratados no Capítulo 8.

Finalmente, surge a necessidade de manter os objetos persistentes da aplicação para controle de satélites em um sistema de armazenamento de dados. Cria-se, então, o

serviço de persistência, capaz de armazenar/recuperar os objetos persistentes da aplicação, como por exemplo, as telemetrias recebidas do satélite, os telecomandos enviados etc. Esse serviço recebe solicitações dos objetos persistentes para armazená-los e localiza o banco de dados que deve ser armazenado esse objeto. Ele implementa a transparência da localização física de armazenamento de um objeto, ou seja, como a arquitetura proposta é dinâmica, a localização física de um objeto pode ser diferente da localização do banco de dados onde ele deve ser armazenado. Esse serviço disponibiliza para a aplicação os serviços de acesso ao banco de dados, tais como armazenamento, recuperação e exclusão. Os objetos persistentes da aplicação simplesmente acionam esses serviços. Os detalhes desse serviço são tratados no Capítulo 9.

6.2 - A CARGA DO SISTEMA

Considerando as políticas de segurança que qualquer sistema de controle de satélites, deve-se estabelecer previamente um conjunto de nós para a implantação da arquitetura SICSD. Desta forma, apesar de o sistema ser distribuído, existe uma relação de nós onde os objetos pertencentes ao sistema podem ser instanciados. A rotina de carga do sistema começa por um nó, que assume o papel de gerente, podendo este ser qualquer um dos nós disponíveis para a carga. O nó gerente acessa uma base de dados, que contém a relação dos outros nós pertencentes ao sistema, bem como suas respectivas rotinas de carga. Finalmente, dispara-se, em paralelo, as rotinas de carga dos nós remotos (Figura 6.3).

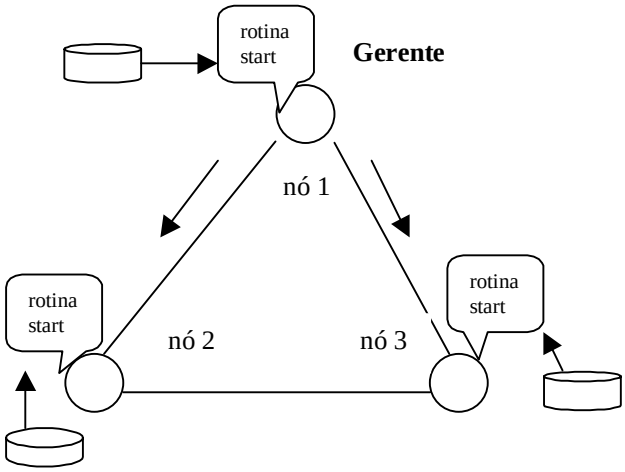


Fig. 6.3-Carga do sistema proposto.

Os objetos instanciados pela rotina de carga estão listados a seguir:

a) Carga dos objetos da aplicação para aquele nó;

b) Carga do serviço responsável pela persistência;

c) Carga do serviço responsável pela segurança;

d) Carga do serviço responsável pelo balanceamento e

e) Carga do serviço dos Agentes.

6.3 - INTERAÇÃO DO USUÁRIO COM O SISTEMA

O serviço de balanceamento responsabiliza-se pela interface entre os usuários e a arquitetura proposta. Sua tarefa básica concentra-se em identificar o usuário e disponibilizar funções de acordo com o seu perfil. Como por exemplo, um controlador de satélites tem um conjunto de funções diferentes de um engenheiro de satélites. A responsabilidade do primeiro converge para a monitoração das telemetrias recebidas do satélite. Já o segundo dirige sua atenção para diagnosticar o funcionamento interno dos subsistemas de um do satélite, tais como o funcionamento da bateria, do subsistema de carga útil etc.

O processo de atendimento da solicitação de um serviço segue os seguintes critérios:

Primeiro verifica se no nó onde o usuário está conectado existem objetos da aplicação que atendam à solicitação desejada. Em caso afirmativo, estabelece-se a conexão no próprio nó, Caso contrário, faz-se uma busca na *base de configuração* e recuperam-se aqueles objetos que atendam ao serviço solicitado. Estabelece-se a conexão com o objeto instanciado no nó que apresentar a maior disponibilidade de CPU.

A atribuição dada ao serviço de balanceamento, para fazer a interface entre a arquitetura proposta e o usuário, deve-se basicamente pelo fato de o serviço de balanceamento ter a capacidade de identificar qual o objeto se encontra mais apto para atender à solicitação de serviço de um usuário. Ou seja, diante das informações geridas pelo serviço dos agentes e armazenadas na base de configuração, o serviço de balanceamento identifica o objeto que esteja instanciado no nó, com a menor carga de trabalho, para atender determinada solicitação de serviço.

6.4 - FUNCIONAMENTO DO SISTEMA

Como já foi mencionado, implementou-se a arquitetura SICSD em um domínio de rede predefinido. Cada um dos nós pertencentes a esse domínio contém uma rotina de carga, responsável por instanciar os objetos residentes nesse nó. A primeira ação realizada pela rotina de carga de cada nó consiste em instanciar os objetos dos Serviços de Persistência, Agentes, Segurança, Balanceamento e, finalmente, os objetos da aplicação para controle de satélites. A dinâmica de interação em cada nó, entre os serviços e os objetos da aplicação, pode ser mostrada na Figura 6.4 e comentada a seguir.

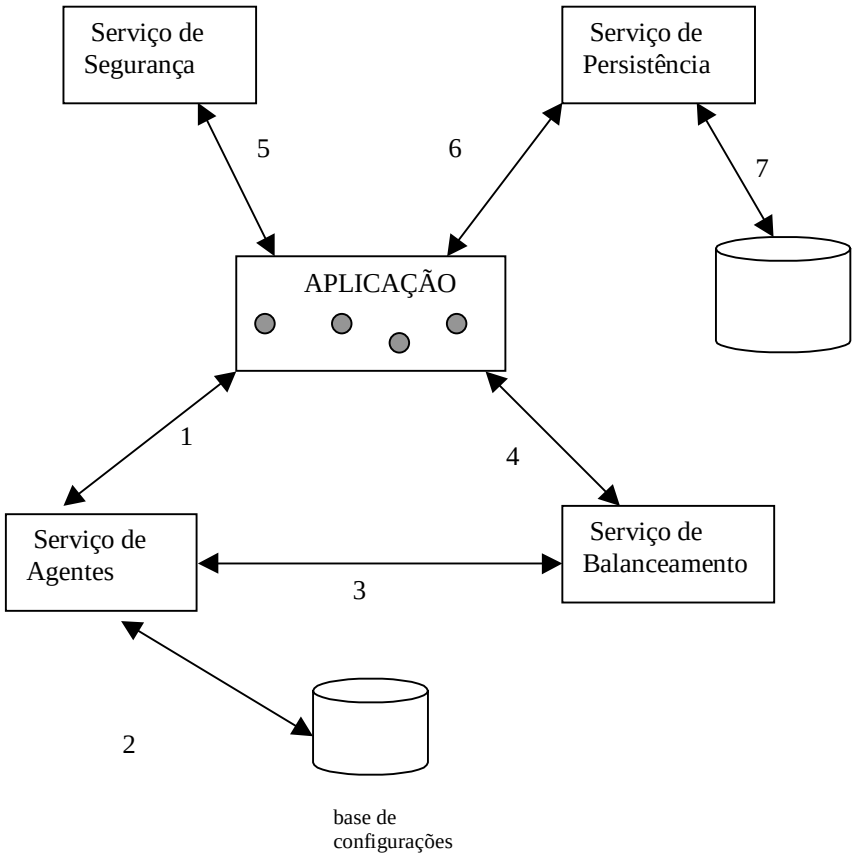


Fig. 6.4 - A dinâmica de interação em cada nó entre os serviços e os objetos.

Comentários:

1 - Após a carga dos objetos da aplicação, os agentes inicializam também as suas atividades, que consistem, em um primeiro instante, na monitoração dos objetos da aplicação.

2 – As informações monitoradas pelos agentes são mantidas na base de configurações. Vários parâmetros são monitorados, dentre eles podem ser citados:

- o estado de um objeto, podendo ser “ready”, se o objeto apresenta-se disponível para receber solicitações de serviços dos usuários, ou “failed”, caso ele não esteja disponível;
- a disponibilidade de CPU de um determinado nó;

- a disponibilidade de I/O de um determinado nó etc.

3 - Todo processo de manutenção na base de configurações deve ser analisado pelo serviço de balanceamento. Ou seja, se o serviço dos agentes detecta que um objeto da aplicação recebeu uma nova solicitação de serviço, essa informação deve ser atualizada na base de configurações e o serviço de balanceamento analisa se essa manutenção alterou o balanceamento de carga do sistema.

4 - A atuação do serviço de balanceamento consiste basicamente em levantar os pontos críticos do ambiente configurado para controle de satélites, ou seja, identificar os nós que contêm uma sobrecarga de trabalho. Dois motivos podem ser a causa desse problema. Primeiro existe um número elevado de objetos instanciados naquele nó; segundo, os objetos instanciados naquele nó estão recebendo várias solicitações de serviços de outros nós da rede.

O serviço de balanceamento resolve o primeiro problema migrando objetos de máquinas saturadas para máquinas ociosas e o segundo, replicando os objetos de um nó saturado em nós ociosos.

5 - O serviço de segurança analisa todo acesso de objetos da aplicação. Essa prevenção tem como objetivo garantir que somente objetos/usuários autorizados possam tirar proveito dos serviços prestados pelos objetos.

6 - O serviço de persistência tem como objetivo armazenar/recuperar os objetos persistentes da aplicação. Os métodos de armazenamento dos objetos persistentes da aplicação não acessam diretamente o sistema de armazenamento de dados; simplesmente eles acionam, através de mensagens, o serviço de persistência para fazer essa operação.

7 - O serviço de persistência assume o papel de mediador entre a aplicação e o sistema de armazenamento de dados.

Com a difusão da INTERNET, o número de aplicações distribuídas, utilizando padrões como o CORBA ou DCOM, em crescimento. Apesar de serem distribuídas, as aplicações ainda mantêm uma arquitetura de software cliente-servidor, ou seja, existe mais de um servidor estático em algum ponto da rede, atendendo a solicitações de

serviços de vários clientes. O objetivo deste trabalho concentra-se em apresentar um protótipo de uma aplicação distribuída, aplicada ao software de controle de satélites, onde os servidores serão dinâmicos, ou seja, poderão migrar de um nó para outro, de acordo com as solicitações de serviços.

Um ambiente distribuído flexível e dinâmico difere de um ambiente distribuído estático pela capacidade do sistema de se ajustar à demanda por solicitações de serviços. O aumento da demanda propicia a criação de novas réplicas do objeto, que atendem essa solicitação. Em contrapartida, a queda na demanda pode disparar o processo de destruição dos objetos ociosos no sistema.

A união da utilização do conceito de distribuição com o conceito de agentes permitirá construir uma aplicação dinâmica e flexível, capaz de se adaptar às necessidades do usuário, com a utilização balanceada dos recursos da rede. Se a demanda por determinado serviço aumenta, os agentes são capazes de detectar o objeto que atenda esse serviço, bem como o número de usuários solicitantes. Essa informação é disseminada para os outros nós, com o auxílio dos próprios agentes. O objeto que atenda essa demanda poderá migrar ou ser replicado. Dessa forma o sistema se auto-ajustará para atender às solicitações de seus usuários.

A falha de um computador ou, por exemplo, de um objeto, que envia telecomandos para o satélite, fica transparente para o usuário. A arquitetura proposta se incumbe de localizar outro objeto, capaz de continuar a responder à solicitação do usuário. A arquitetura proposta colocará disponível réplicas do objeto, com a responsabilidade de enviar comandos para o satélite.

Enfim, o objetivo deste trabalho de pesquisa é criar um ambiente capaz de se adaptar dinamicamente às solicitações dos controladores e demais usuários do sistema, melhorando um conjunto de características, como desempenho, flexibilidade, confiabilidade e utilização dos recursos computacionais disponíveis.

CAPÍTULO 7

O SERVIÇO DOS AGENTES

A inteligência... é a capacidade de criar objetos artificiais, especialmente ferramentas para fazer ferramentas. (Henri Bergson)

O serviço dos agentes surge como uma prestação de serviços para o serviço de balanceamento, ou seja, a partir das informações coletadas pelos agentes, o serviço de balanceamento atua no sistema, tentando otimizar os recursos computacionais existentes. Dentre os principais requisitos atendidos pelo serviço dos agentes podem ser destacados:

1) Supervisionar os objetos em diversas máquinas, obter informações a respeito de cada objeto/máquina, enviar mensagens para cada objeto/máquina.
2) Permitir uma visão global, quanto às informações sobre os objetos, a partir de qualquer máquina.
3) Servir de apoio à decisão do serviço de balanceamento.

A implementação do serviço dos agentes em cada nó conta com o apoio dos agentes fixos e móveis. O agente fixo, denominado agente supervisor, atua dentro do contexto como um ponto de referência em cada nó local. Uma vez instanciado, o agente supervisor responsabiliza-se pelo controle do armazenamento, controle de acesso, atualizações, disseminação e disponibilização das informações referentes aos objetos instalados naquela máquina. Ou seja, sua missão compreende, em primeira instância, tarefas de supervisão dirigidas ao nó, onde ele está instanciado.

O agente supervisor administra a estrutura, de maneira que as informações geridas por ele não se restrinjam ao contexto limitado de apenas um nó. Tal abordagem permite o acesso às informações distribuídas por todos os nós do sistema. Basicamente, pode-se estabelecer que o agente supervisor implementa uma abstração similar a de um servidor, que disponibiliza uma variedade de serviços embutidos ao seus usuários.

Em resumo, as principais atividades de responsabilidade do agente supervisor são:

a) Controlar e manipular a base de configuração do sistema.

b) Instanciar os agentes mensageiros e agentes monitores.

Os *agentes móveis*, denominados neste trabalho de pesquisa como *agentes mensageiros*, têm como atividade principal disseminar, por todos os nós da rede, as informações coletadas pelos agentes monitores e supervisores. Seu funcionamento assemelha-se ao do agente supervisor, ou seja, de tempos em tempos ele é acionado, recupera as informações gerenciadas pelo agente supervisor e transfere para todos os outros nós da rede, até retornar ao nó de origem. Cada agente móvel tem uma base de dados que contém os nós por onde ele deve percorrer. Instanciados pelo agente supervisor de cada nó, realizam tarefas predeterminadas, movendo-se por todos os nós do sistema e retornando à sua origem, após completar suas missões.

Os *agentes monitores* apresentam características similares ao agente supervisor, ou seja, são fixos em cada nó da rede. Sua tarefa básica compreende monitorar os objetos da aplicação de controle de satélites. Para cada objeto instanciado da aplicação, existe um agente monitor responsável por monitorar o seu estado.

7.1 - ATUALIZAÇÃO DA BASE DE CONFIGURAÇÃO DO SISTEMA

A base de configuração do sistema coleciona diversas tabelas de configuração, concernentes tanto ao âmbito dos objetos como ao estado do sistema (através de algumas variáveis de estado). As informações armazenadas nas tabelas procuram retratar uma visão, a partir de cada "host", generalizada da situação de todos os outros "hosts" do ambiente. Para tanto, definem-se as seguintes tabelas:

a) Tabela de nós;

b) Tabela de objetos; e

c) Tabela de conexões.

Após a carga do sistema, o agente supervisor inicia o processo de monitoração dos objetos localmente instanciados e armazena essas informações na base de configuração. Ele não atua diretamente no processo de monitoração dos objetos da aplicação de controle de satélites instanciados em cada nó. Ele delega esta atividade para o agente

monitor. Para cada objeto da aplicação instanciado, existe um agente monitor com a responsabilidade de monitorar seu "status", podendo ser "ready" ou "failed". Essas informações são repassadas para o agente supervisor e armazenadas na base de configurações. Em seguida, os agentes mensageiros transportam o conteúdo dessa base para os nós remotos. Depois de um determinado período, cada agente supervisor tem conhecimento dos objetos instanciados nos nós remotos. Os passos da atualização da base de configuração são comentados a seguir e ilustrados na Figura 7.1.

1 - Existe um agente monitor fixo para cada objeto instanciado da aplicação.

2 - Os parâmetros monitorados são repassados para o agente supervisor.

3 - O agente supervisor mantém estes parâmetros armazenados na base de configuração.

4- Os agentes mensageiros responsabilizam-se pelo trabalho de distribuição do conteúdo da base de configuração para os outros nós da rede.

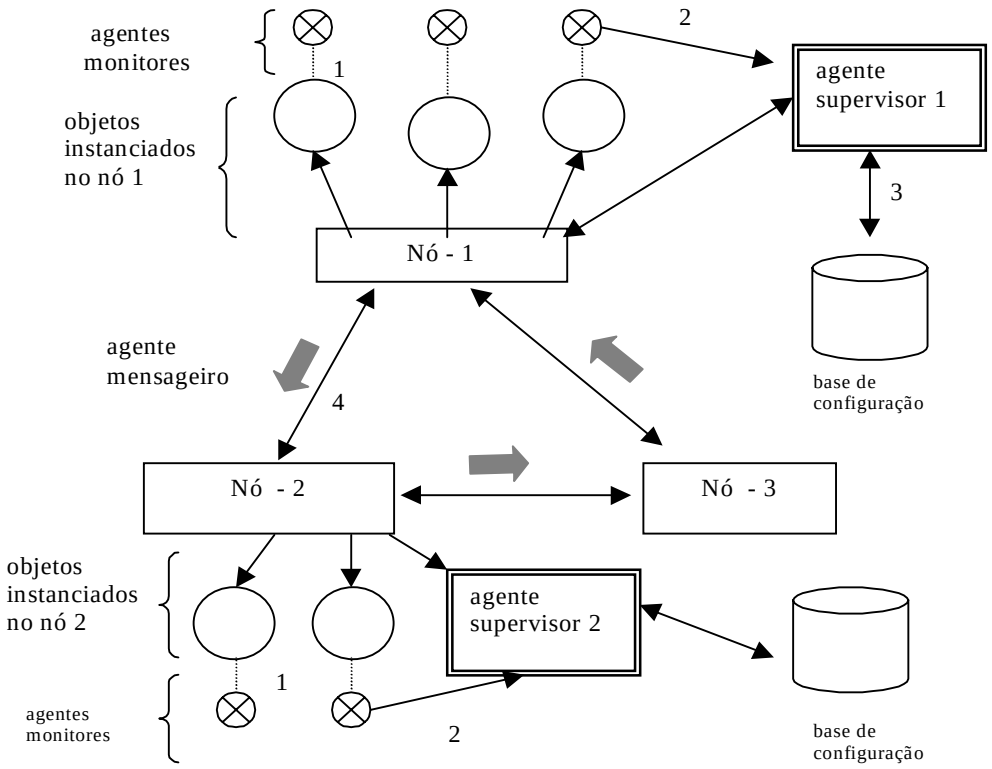


Fig- 7.1 – Interação entre os agentes supervisores, monitores e mensageiros.

O processo de atualização da base de configuração nos nós remotos é realizado pelo agente supervisor, dentro de um intervalo predefinido. Após a atualização da base local,

o agente supervisor aciona o agente mensageiro para atualizar esses dados nos nós remotos.

As características de flexibilidade e dinamismo, propostos na arquitetura distribuída para controle de satélites, são reflexos da exploração do conteúdo da base de configuração pelo serviço de balanceamento.

7.1.1 – Tabela de Nós

A *tabela de nós* contém a relação dos nós definidos para a implantação da arquitetura proposta para controle de satélites. Os campos que compõem essa tabela são listados a seguir (Tabela 7.1).

- nome do nó;
- desempenho;
- disponibilidade de CPU ;
- disponibilidade de I/O ;
- status;
- taxa de disponibilidade de CPU e
- taxa de disponibilidade I/O.

TABELA 7.1 – TABELA DE NÓS

Nome	Desempenho	Disp. De CPU	Disp. de I/O	Status	Taxa de disp. de CPU	taxa de disp. de I/O
Patras	5	0.5	0.4	Ready	2.5	2.0
Pelion	2	0.7	0.2	Suspend	1.4	0.4
Andros	1	0.4	0.1	Ready	0.4	0.1

Dados da tabela:

Nome do nó: Contém o nome atribuído a cada máquina, dentro de um domínio de rede de computadores.

Desempenho: O fator desempenho baseia-se no “benchmark” de cada computador. Os testes de “benchmark” podem ser divididos em dois tipos: Benchmark para componentes isolados e para todo o sistema.

Os “Benchmarks” para componentes isolados medem o desempenho específico de um determinado componente, como por exemplo o tempo de acesso ao Winchester, o tempo de acesso à memória RAM (Random Access Memory) etc.(INTEL,2000; ZDNET,2000). O “benchmark” para todo o sistema mede o desempenho do sistema como um todo. Isto envolve tempo de acesso ao Winchester, memória etc.

Buscando medir o desempenho do sistema como um todo, optou-se pelo “benchmark” “Winstone 99”. Esse aplicativo pode ser utilizado para medir o desempenho de aplicações que executam nos sistemas operacionais da Microsoft (Windows 95, 98 e NT). Optou-se por esse “benchmark” para que se possa ter uma idéia global do desempenho de cada máquina utilizada na arquitetura SICSD.

A aplicação para controle de satélites apresenta-se heterogênea, ou seja, existem objetos que utilizam somente os mecanismos de entrada e saída de dados e outros que acessam somente a CPU. Portanto, se fosse utilizado um “benchmark”, que medisse o desempenho somente do processador, poderia não retratar a realidade da aplicação.

A atribuição do fator desempenho para cada nó seguiu-se o seguinte critério:

A máquina com menor desempenho recebeu peso um, as demais foram calculadas proporcionalmente a essa máquina. Por exemplo, a máquina que teve o desempenho duas vezes maior que a de menor desempenho teve peso dois. O parâmetro desempenho foi calculado para todas as máquinas disponíveis, para a implantação do protótipo da arquitetura SICSD.

Disponibilidade de CPU: O parâmetro disponibilidade de CPU é monitorado pelo agente supervisor em intervalos de tempo predefinido. Ele apresenta, em porcentagens, a disponibilidade média da CPU em um determinado período. A freqüência de leitura desse parâmetro pode ser configurável (Kunz,1991).

Disponibilidade de I/O: O parâmetro disponibilidade de I/O também é monitorado pelo agente supervisor . Ele apresenta, em porcentagens a disponibilidade média do

processador de entrada e saída do disco, em um determinado período, para cada nó do sistema.

Status: Este parâmetro contém o estado de um nó em um determinado instante. A freqüência de leitura deste parâmetro é a mesma do parâmetro disponibilidade de CPU.

Um nó pode se encontrar nos seguintes estados:

- "ready": pronto para receber solicitações de serviços;
- "failed": ocorreu uma falha no nó não conhecida, como por exemplo a queda do sistema operacional, falha de conexão em rede etc.
- "suspend": o nó pode encontrar-se suspenso para manutenção.

Taxa de disponibilidade CPU: Este parâmetro é calculado em função do resultado da multiplicação do parâmetro "desempenho" pelo parâmetro "disponibilidade de CPU". A freqüência de cálculo desse parâmetro segue a freqüência de leitura do parâmetro disponibilidade de CPU. Quanto maior o resultado, maior a disponibilidade do processamento de CPU de um nó, como por exemplo, baseando-se na Tabela 7.1, o nó patras tem a taxa igual a 2.5; portanto ele contém uma maior disponibilidade de CPU do que o nó pelion, que tem a taxa igual a 1.4.

Analisando a taxa de disponibilidade de CPU, tem-se uma idéia geral de quais os nós se apresentam com uma carga maior de processamento, porque quanto menor o valor, mais sobrecarregado está o nó. Portanto, o sistema apresenta-se melhor distribuído, ou seja, com as cargas de CPU balanceadas, se a taxa de disponibilidade de CPU para um determinado nó estiver próxima as demais.

Taxa de disponibilidade de I/O: Este parâmetro é calculado em função do resultado da multiplicação do parâmetro "desempenho" pelo parâmetro "disponibilidade de I/O". A freqüência de cálculo desse parâmetro segue a freqüência de leitura do parâmetro "disponibilidade de I/O". Quanto maior o resultado, maior a disponibilidade do nó para fazer operações de entrada e saída, como por exemplo, baseando-se na Tabela 7.1, o nó patras tem a taxa igual a 2.0; portanto ele tem maior disponibilidade para fazer operações de I/O do que o nó pelion, que tem a taxa igual a 0.4.

De acordo com a taxa de disponibilidade de I/O, tem-se uma idéia geral de quais os nós têm uma carga maior de processamento de entrada e saída, porque quanto menor o valor, mais sobrecarregado está o nó.

A análise deste parâmetro para todos os nós pode indicar como está o balanceamento de carga de entrada e saída do sistema. O sistema apresenta-se melhor balanceado se o valor calculado para um determinado nó estiver próximo aos demais.

7.1.2 – Tabela de Objetos

A tabela de objetos contém a relação dos objetos instanciados em cada nó, dedicado à implantação da arquitetura SICSD. Esta tabela é constituída pelos seguintes campos (Tabela 7.2):

- Nome do objeto;
- Nome do nó onde o objeto está instanciado;
- O número de conexões existentes para este objeto e
- O "status" do objeto, pode ser:

♦ "Ready", se o objeto estiver pronto para receber conexões.

♦ "Failed", se o objeto não responder a uma solicitação de serviço. O estado "failed" pode ser ocasionado por uma falha no próprio objeto, uma falha na rede ou uma falha no computador onde está instanciado esse objeto.

TABELA 7.2 – TABELA DE OBJETOS

Nome	Nó	Nconexao	Status
Obj1	Patras	5	Ready
Obj2	Pelion	0	Failed
Obj3	Andros	10	Ready
Obj2	Andros	4	Ready

De acordo com a Tabela 7.2, o objeto "obj1" está instanciado no nó Patras, contém 5 conexões ativas e está pronto ("ready") para receber novas conexões.

A monitoração do estado de cada objeto é obtida através da atuação dos agentes monitores. Como já foi mencionado, para cada objeto da aplicação de controle existe um agente monitor. Em termos pragmáticos, esse agente acessa em intervalos de tempos predefinidos o objeto da aplicação com o qual ele relaciona. Caso falhe a tentativa de acesso, o agente monitor altera o status do objeto da aplicação de "ready" para "failed". O parâmetro número de conexões de um objeto é inicializado com zero e atualizado a cada solicitação de serviço recebida pelo objeto. Toda solicitação de serviço feita pelo usuário é analisada pelo serviço de balanceamento. Este se responsabiliza por localizar o objeto que atende à solicitação, bem como por informar o agente supervisor o objeto que está sendo solicitado. O papel do agente supervisor nesse contexto se resume em incrementar o número de conexões recebidas do objeto.

7.1.3- Tabela de Conexões

A *tabela de conexões*, além de conter o número de conexões de um objeto, ela também contém a origem das conexões (Tabela 7.3).

TABELA 7.3 – TABELA DE CONEXÕES

Nó	Objeto	Origem	Num. Conexão
Patras	Obj1	Patras	3
Patras	Obj1	Pelion	2
Andros	Obj2	Patras	3
Andros	Obj3	Pelion	10

De acordo com a Tabela 7.3 o objeto "obj1", instanciado no nó patras, tem cinco conexões, sendo três conexões originadas do próprio nó e duas conexões originadas do nó pelion.

Esta tabela contabiliza o número de solicitações de serviço de um objeto, agrupadas por nós de origem. O agente supervisor responsabiliza-se por gerir o conteúdo da tabela de conexões; entretanto, o serviço de balanceamento, através do seu método "atender à solicitação de usuário" contribui com esse processo à medida que informa ao agente supervisor o nó de origem da conexão, bem como o objeto solicitado.

7.2 - OS AGENTES MONITORES

Os Agentes Monitores de Objetos (AMO) são agentes estacionários instanciados pelo agente supervisor em cada nó da rede, com funcionalidades bem definidas, ou seja, monitorar os objetos da aplicação de controle de satélites(Silva,2000). Assim que um novo objeto da aplicação é instanciado, instancia-se também um agente AMO, que concretiza um vínculo vitalício com o objeto. Sua atividade básica consiste em: (ver Figura7.2)

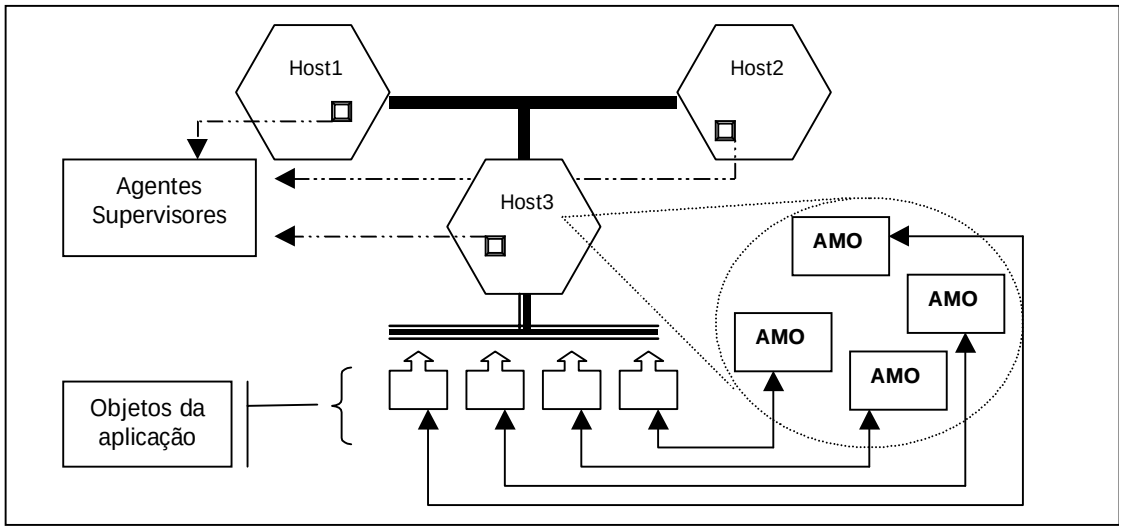


Fig. 7.2: Agentes AMO monitorando objetos da aplicação (proposta SICSD).

Verificar o estado do objeto e transferir essas informações para o agente supervisor. O objeto pode apresentar os estados ativo(ready), em manutenção (suspend) ou inativo, porque não se consegue acessar os serviços desse objeto(failed). Essa funcionalidade atribuída aos agentes monitores foi detalhada na seção 7.1.2.

7.3 - OS AGENTES MENSAGEIROS

Os Agentes Mensageiros (AM) caracterizam-se por agentes móveis que exploram a capacidade de migração entre os nós predefinidos da rede para efetivar suas missões. As funções essenciais desses agentes se relacionam diretamente aos procedimentos para atualização e disseminação das informações contidas nas bases de configuração. Os métodos de definição dessa classe devem permitir abordagens flexíveis para os seguintes casos:

a) Delegação (disparo) do agente: Por “default”, o agente supervisor se encarrega do “disparo” dos agentes mensageiros, definindo estratégias que possibilitem adaptações em face das alterações previstas ou não. Como por exemplo, o agente supervisor define a rota que o agente mensageiro deve seguir para completar sua missão, em função do número de nós ativos na rede.

b) Determinação da freqüência das operações: A freqüência em que o agente migra para os nós, atualizando e disseminando as informações, por “default”, obedece a parâmetros determinados pelo agente supervisor. Convém destacar que a periodicidade pode ser calibrada de acordo com as necessidades do ambiente e com as decisões do agente supervisor.

c) Definição de Itinerários: Os itinerários orientam a seqüência de nós, que devem ser visitadas durante as operações de atualização. Informados a respeito das condições de tráfego das “vias de acesso”, pelas quais passará, o agente mensageiro obtém a descrição dos melhores roteiros a fim de obter um melhor desempenho durante as movimentações, ao mesmo tempo em que evita a sobrecarga dos “caminhos” atualmente congestionados.

d) Transferência de Informação (Replicação): A fim de cumprir sua missão, o agente mensageiro basicamente deve transferir informações para todos os nós, a partir da base de configuração local.

A Figura 7.3 visa demonstrar o esquema básico desempenhado pelo agente mensageiro.

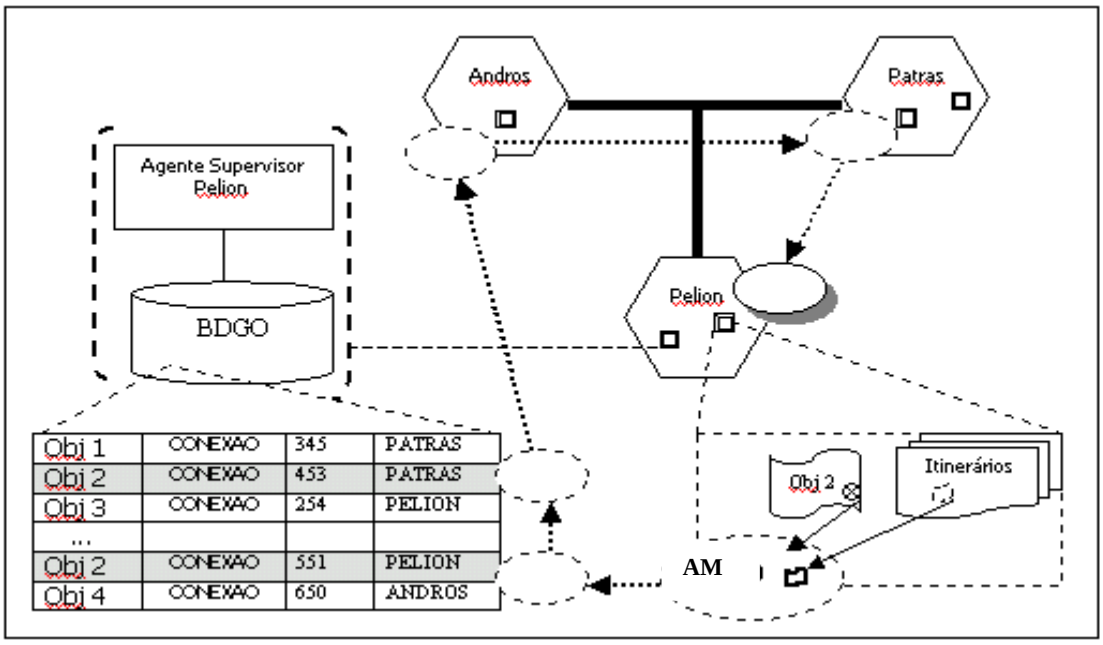


Fig. 7.3 O Agente mensageiro disseminando informações sobre o objeto **Obj2**.

7.4 - IMPORTÂNCIA DO SERVIÇO DOS AGENTES

A tecnologia de agentes complementa e incrementa a funcionalidade da tecnologia de objetos distribuídos. Dentre as principais contribuições na utilização deste paradigma, podem-se ressaltar:

- Eficiência: Em certos casos, os agentes podem consumir menos recursos de rede, já que podem levar a computação aos dados. Deve-se ressaltar também, a possibilidade dos agentes migrarem para onde existam melhores recursos(cpu, memória, etc..);

- Redução de tráfego na rede: Tendo em vista que a maioria dos protocolos de comunicações envolvem várias interações, especialmente quando medidas de segurança estão habilitadas. Com o uso dos agentes, essas interações acontecem localmente.

- Autonomia assíncrona: tarefas podem ser atribuídas a agentes e esses operam assíncrona e independentemente do programa que os enviou.

- Controle de atividades em tempo real: Como os agentes executam localmente, evita-se o problema de latência, intolerável no controle de aplicações de tempo real.

- Robustez e tolerância a falhas: A capacidade de reagirem dinamicamente em situações adversas, tornam os agentes mais propícios a um comportamento tolerante a falhas.

- Desenvolvidos de modo a suportar ambientes heterogêneos, podem atuar como um mecanismo que aumente a interoperabilidade entre esses ambientes.

- Paradigma de desenvolvimento conveniente: O projeto e a implementação de sistemas distribuídos podem se tornar mais fáceis com o uso de agentes móveis, pois esses são inerentemente distribuídos e, assim, possuem uma abstração natural de um sistema distribuído.

CAPÍTULO 8

O SERVIÇO DE BALANCEAMENTO

É do senso comum capturar um método e experimentá-lo. Se ele falhar, admita isso com franqueza e experimente outro. Mas acima de tudo tente algo. (Franklin Delano Roosevelt)

O serviço de balanceamento apresenta-se como o responsável por analisar os dados coletados e armazenados pelo serviço dos agentes. O objetivo principal deste serviço consiste em propor mecanismos capazes de distribuir o processamento de CPU entre os nós predefinidos utilizados na arquitetura SICSD, buscando, dessa forma, otimizar os recursos computacionais existentes.

Toda a atualização da base de configuração deve ser captada pelo serviço de balanceamento. Existem vários motivos que levam à atualização dessa base, dentre eles, podem-se citar:

- a alteração (incremento ou decremento) no número de solicitações de serviços atendidas por um objeto;
- a mudança de status de um objeto de “ready” para “failed”;
- a migração de um objeto de um nó para outro etc.

Após a análise das atualizações, e se for o caso, o processo de balanceamento de carga é disparado e o resultado consiste normalmente na replicação ou na migração de um objeto do nó origem para o nó destino.

As características de *flexibilidade* e *dinamismo,* propostas na arquitetura, são implementadas pela atuação do serviço de balanceamento. A *flexibilidade* deriva da capacidade de a Arquitetura SICSD se adaptar ao número de usuários do sistema. Ou seja, se houver um aumento na demanda por solicitações de serviços, o serviço de balanceamento responsabiliza-se por instanciar novos objetos no sistema que atendam a essa solicitação.

A característica de *dinamismo* proposto na Arquitetura consiste na capacidade dada aos objetos de se moverem de um nó para outro, de acordo com a demanda por solicitação de serviços. Ou seja, se a demanda aumenta, o objeto da aplicação, que atende essa solicitação, pode mover-se para o nó remoto solicitante.

Vários motivos podem disparar o processo de balanceamento de carga, dentre eles podem ser citados os seguintes:

- **Condição 1**: O nó de origem está congestionado, ou seja, vários objetos foram criados e a taxa de disponibilidade de CPU apresenta-se abaixo da média da taxa dos outros nós. Nesse caso, faz-se um levantamento entre os nós remotos para identificar qual o nó remoto tem o maior número de conexões com o nó origem. Finalmente, instancia-se uma cópia do objeto mais solicitado no nó remoto, que contenha o maior número de conexões com o nó origem.

- **Condição 2**: A busca por disponibilidade de serviços. Se existem vários nós ociosos na rede, podem ser instanciadas cópias de objetos de máquinas saturadas em máquinas ociosas.

- **Condição 3:** Uma necessidade operacional, por exemplo, um processo de manutenção em um nó. Nesse caso, há necessidade de uma intervenção do administrador que deve alterar o "status" deste nó de "ready" para "suspend". O serviço de balanceamento assume em seguida o controle desse nó. Sua atuação compreende, em um primeiro instante, suspender todas as conexões existentes com os objetos naquele nó; em seguida, há a criação desses objetos em máquinas remotas mais ociosas.

- **Condição 4:** Queda de um nó. A queda de um nó, involuntariamente, será detectada na primeira tentativa de comunicação de um agente com esse nó. De acordo com a arquitetura SICSD, o agente supervisor tem as informações de todos os objetos instanciados nos nós remotos. Dessa forma, o primeiro nó que identificar a falha no nó vizinho será o responsável por instanciar os objetos desse nó, em outros nós, que estejam disponíveis. As conexões existentes com o nó que entrou no estado "failed" serão perdidas.

- **Condição 5**: Inclusão de um novo nó: A inclusão de um novo nó no sistema conta com a cooperação do administrador de sistemas. Os agentes apresentam-se como os responsáveis por disseminar pela rede, o endereço do novo nó. O processo consiste em disponibilizar os recursos computacionais desse nó inserido na rede para proceder o balanceamento de carga.

- **Condição 6**: Exclusão de um objeto. Os objetos que foram instanciados, entretanto, não estão sendo referenciados por nenhum usuário ou outro objeto, devem ser excluídos. Nesse caso, não apresentam nenhuma conexão ativa. Os objetos que foram criados para atender um "pico" na demanda, por solicitação de serviços, como por exemplo na fase de lançamento de um satélite, podem ser excluídos ao término dessa fase.

- **Condição 7**: Criação de um objeto. O serviço de balanceamento apresenta-se como um mediador entre o usuário e a aplicação de controle de satélites. Esse serviço mostra-se capacitado a criar uma instância de um objeto, caso não encontre nenhuma disponível no domínio de rede definido para a aplicação de controle de satélites. Existe a possibilidade de ocorrer esta condição, já que a condição 6 pode ter excluído um objeto que se encontrava ocioso (número de conexões igual a zero).

- **Condição 8**: Atender à solicitação de um usuário. Toda solicitação de serviço de um usuário para a aplicação de controle de satélites é analisada pelo serviço de balanceamento. Este presta o serviço de identificar o objeto que atenda à solicitação do serviço, bem como o nó onde ele se encontra instanciado.

De acordo com a arquitetura SICSD, instancia-se um serviço de balanceamento em cada nó; portanto, através desse serviço, cada nó tem autonomia para migrar ou replicar seus objetos. A implementação das condições citadas anteriormente estão distribuídas em sete métodos do serviço de balanceamento, assim denominados:

a) balancear carga;

b) remover nó;

c) inserir nó;

d) realizar manutenção;

e) atender à solicitação do usuário;

f) excluir objeto e

g) criar objeto.

O método “balancear carga” implementa as condições 1 e 2. O método “remover nó” auxilia na solução da condição 4. O método “inserir nó” auxilia na solução da condição 5. A solução da condição 3 utiliza os recursos do método “realizar manutenção”. Os métodos “excluir objeto” e “criar objeto” colaboram na solução das condições 6 e 7 respectivamente . O método “atender à solicitação do usuário” implementa a condição 8.

8.1 - MÉTODO PARA BALANCEAR CARGA

O processo de balanceamento de carga consiste em cada nó se auto-analisar e comparar a carga de processamento local com as cargas de processamento remoto. Se os nós remotos têm maior disponibilidade de processamento, ativa-se o processo de balanceamento. A lógica deste método deve implementar a seguinte tabela de decisões:

TABELA 8.1-TABELA DE DECISÕES DO MÉTODO BALANCEAR CARGA

Recursos de CPU	**Número de conexões**	**Ação realizada**
CPU ↓	> 0	Replicar objeto para nó com maior disponibilidade de CPU
CPU ↑	> 0	Não faz nada
CPU ↓	= 0	Migrar objeto para nó com maior disponibilidade de CPU
CPU ↑	= 0	Não faz nada

Legenda:

CPU ↑ : O nó contém uma taxa de disponibilidade de CPU acima da média das taxas de outros nós.

CPU ↓ : O nó contém uma taxa de disponibilidade de CPU abaixo da média das taxas de outros nós.

> 0: O objeto está atendendo a solicitações de serviços, ou seja, o número de conexões é maior do que zero.

A interpretação da primeira linha da tabela deve ser a seguinte:

A taxa de disponibilidade de CPU (CPU ↓) do nó se apresenta abaixo da média das taxas de outros nós; o número de conexões com esse objeto é maior do que zero(`>0).` Neste caso, deve-se replicar este objeto para um nó que expresse maior disponibilidade de CPU. As taxas de disponibilidade de CPU e o número de conexões são recuperados da base de configuração.

Os possíveis caminhos seguidos pelo método são mostrados a seguir e ilustrados na Figura 8.1.

- Acessar a base *de configuração,* com o auxílio do agente supervisor e recuperar a taxa de disponibilidade de CPU para todos os nós.
- Calcular a média das taxas de disponibilidade de CPU .

Se (a taxa de disponibilidade de CPU da máquina local + Δ) se apresentar menor que a média { Δ = valor empírico = 10% da média da CPU}
então
{ Esperar um tempo aleatório (0 <= Δt <= ß) }.
{ Na implementação da simulação desse algoritmo utilizou-se ß = 8 segundos - valor empírico; entretanto, esse valor pode ser configurável }.
- Recuperar novamente a taxa de disponibilidade de CPU para todos os nós.
- Calcular novamente a média das taxas de disponibilidade de CPU.
Se (a taxa de disponibilidade de CPU do nó local + Δ) mantiver abaixo da média

então

{A **primeira tentativa** de balanceamento consiste em replicar o objeto mais solicitado da máquina local para a máquina remota solicitante }.
- Recuperar da base de configurações o nó remoto que contenha o maior número de conexões com o nó local.
- Recuperar o objeto mais solicitado por esse nó remoto.
- **Chamar a rotina replicar_objeto.**

{ **A Segunda tentativa** de balanceamento consiste em migrar os objetos ociosos das máquinas que se apresentam com a disponibilidade de CPU abaixo da média}.
Se (a taxa de disponibilidade de CPU do nó local + Δ) mantiver abaixo da média
então
-Recuperar objetos com número de conexões = 0
-Enquanto houver objetos com conexão = 0 e (a taxa de disponibilidade de CPU do nó local + Δ)< média
Início
- Chamar a rotina **migrar_objeto.**
- Recuperar objetos com número de conexões = 0.
fim_enquanto
fim-se

fim-se

fim-se

rotina replicar_objeto
{ Verificar se o nó solicitante tem disponibilidade de CPU }.
Se a taxa de disponibilidade de CPU do nó solicitante se apresentar
maior que a média das taxas de disponibilidade de CPU
então
- instanciar uma cópia do objeto mais solicitado no nó solicitante.
- Acionar o serviço dos agentes para instanciar um agente monitor desse objeto.

senão
{ Uma outra tentativa de balanceamento consiste em instanciar o objeto da máquina local mais solicitado no nó com a maior taxa de disponibilidade de CPU }.
- Instanciar uma cópia do objeto com o maior número de conexões no nó com a maior taxa de disponibilidade de CPU.
- Acionar o serviço dos agentes para instanciar um agente monitor desse objeto.

fim-se

rotina **migrar_objeto**

{ Esta rotina consiste em instanciar uma cópia do objeto a ser migrado no nó remoto com maior disponibilidade de CPU e, finalmente, excluir esse objeto do nó local}.
- Identificar nó com maior disponibilidade de CPU.
- Instanciar uma cópia do objeto a ser migrado no nó remoto com maior disponibilidade de CPU.
- Acionar o serviço dos agentes para instanciar um agente monitor desse objeto.
- excluir este objeto do nó local.
- Acionar o serviço dos agentes para excluir o agente monitor desse objeto no nó local .

Observação: O algoritmo de balanceamento espera por um tempo aleatório para começar sua execução com o objetivo de evitar que mais de um nó tome decisões semelhantes na busca pelo balanceamento de carga. O algoritmo de balanceamento faz um levantamento da taxa de disponibilidade de CPU de todos os nós integrantes da arquitetura proposta. A finalidade básica deste algoritmo consiste em distribuir a carga de processamento entre os nós; para isto, tenta-se migrar ou replicar objetos de máquinas saturadas para máquinas ociosas. A identificação do nó mais ocioso pode estar sendo realizado ao mesmo tempo pelo algoritmo em nós diferentes da rede, o que poderia provocar a migração ou replicação de objetos de vários nós saturados para um mesmo nó ocioso. Se este fato ocorrer, provavelmente o nó ocioso se tornará

saturado e, da mesma forma, o serviço de balanceamento desse nó tentará reverter esse quadro migrando/replicando os objetos instanciados nesse nó para nós ociosos. Isso poderia provocar oscilações que inviabilizariam o processo de balanceamento do sistema. Se cada nó aguardar um tempo aleatório de espera, pode ser que, ao término do tempo, o sistema se estabilize e não seja necessário executar o balanceamento.

Observa-se que, acrescentou-se um Δ (valor empírico = 10% da média da CPU) na taxa de disponibilidade de CPU do nó local para verificar se disponibilidade de CPU deste nó estava abaixo da média. Este recurso foi utilizado para evitar que um nó com uma taxa de disponibilidade de CPU próximo da média, tente replicar ou migrar objetos. Caso isto ocorra esse nó poderia aumentar sua disponibilidade de CPU e receber novas migrações ou replicações na próxima atuação do serviço de balanceamento. Ou seja, a atuação do serviço de balanceamento não iria ser tão eficaz no processo de distribuição de cargas.

A diferença básica entre a tarefa de replicar e migrar um objeto reside no fato de que a primeira opção mantém uma cópia do objeto a ser replicado no nó local e instancia uma outra cópia no nó remoto. Utiliza-se esta política de balanceamento quando o objeto ainda está sendo referenciado, ou contém um número de conexões maior do que zero no nó local. A segunda tarefa consiste realmente em transferir uma instância de um objeto do nó local para um nó remoto. Aplica-se esta política quando o objeto não está sendo mais referenciado no nó local.

A política de replicação contribui com o balanceamento à medida que disponibiliza na rede uma cópia de um objeto instanciado em uma máquina saturada. As novas solicitações de serviço para esse objeto, provavelmente serão atendidas pela réplica. A migração desponta como a segunda alternativa de balanceamento de carga de um nó saturado. Ela propicia a transferência de objetos de máquinas saturadas e conseqüentemente a redução de carga.

No processo de migração não migra o estado de um objeto. Simplesmente cria-se uma nova instância do objeto migrado no nó destino. Esta política foi adotada porque considerou-se mais importante disponibilizar os serviços encapsulados do objeto

migrado do que simplesmente seu estado. Além disto, armazenar o estado de um objeto que está sendo migrado não é uma atividade trivial.

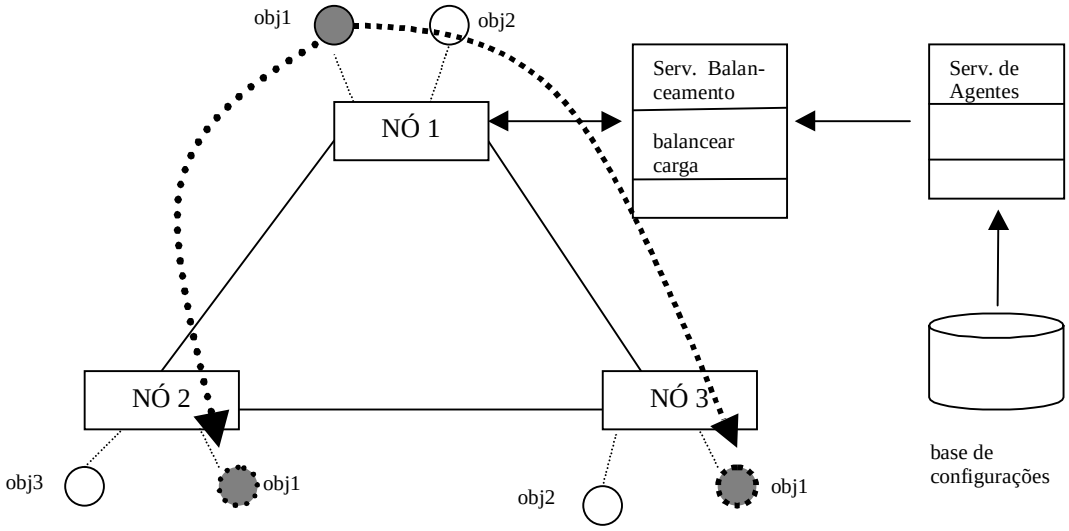


Fig 8.1 – Ilustração do funcionamento do método de balancear carga.

A Figura 8.1 ilustra o seguinte cenário:

1) O serviço de balanceamento instanciado no nó 1 acessa a base de configuração com o auxílio do serviço dos agentes.

2) O nó 1 se apresenta com uma taxa de disponibilidade de CPU abaixo da média; portanto, o método balancear carga deve ser ativado.

3) O objeto denominado "obj1", instanciado no nó 1, contém o maior número de solicitações de serviços originadas do nó 2.

4) O nó 3 apresenta-se como o nó que contém a maior taxa disponibilidade de CPU.

A primeira tentativa de execução do método balancear, instanciado no nó 1, consiste em:

Instanciar uma cópia do objeto "obj1" no nó 2, se a taxa de disponibilidade no nó 2 se apresentar maior do que a média das taxas de disponibilidade de CPU. Isto porque o nó 2 tem o maior número de solicitações de serviços com o objeto "obj1".Caso o nó 2 também esteja saturado, o método balancear examina qual o nó, dentro do domínio da

arquitetura SICSD, possui a maior taxa de disponibilidade de CPU. Uma instância do objeto "obj1" deve ser criada no nó 3, já que nesse exemplo ele expressa a maior taxa de disponibilidade de CPU.

8.2 - MÉTODO PARA REMOVER UM NÓ

1) O Primeiro nó que detectar a falha de um nó alterna o estado deste nó de "ready" para "failed".
2) O nó com maior taxa de disponibilidade de CPU assume o controle do nó "failed" e executa as seguintes atividades:
   2.1) Eliminar o nó da tabela de nós,
   2.2) Enquanto houver objeto no nó "failed",
   início
   Identificar o objeto instanciado naquele nó,
   Identificar o nó com maior disponibilidade de CPU,
   Criar uma cópia do objeto nesse nó,
   Acionar o serviço dos agentes para instanciar um agente monitor desse objeto
   fim-enquanto

8.3 - MÉTODO PARA INSERIR UM NÓ

O processo de inserção de um novo nó no conjunto de nós, predefinido para a arquitetura proposta, conta com a cooperação do administrador do sistema. A intervenção do administrador se resume em executar a rotina de carga deste nó que está sendo inserido no contexto. A partir desse momento, todo o processo de inserção passa a ser automático. Como já foi mencionado anteriormente, o papel da rotina de carga do sistema consiste em instanciar para aquele nó os serviços básicos (serviços dos agentes, segurança, persistência e balanceamento).

A carga do serviço dos agentes engloba instanciar, para cada nó, os agentes supervisor, monitores e mensageiros. A primeira tarefa executada pelos agentes consiste em cadastrar esse novo nó na base de configuração local. Os agentes mensageiros se apresentam como os responsáveis por propagar a existência desse novo nó para os nós remotos. Esse fato se concretiza nos nós remotos na primeira tarefa de atualização das bases de configurações executadas pelos agentes mensageiros desse novo nó.

Em um próximo balanceamento de carga do sistema, esse novo nó inserido se apresenta como um candidato em potencial para receber migrações ou replicações de objetos

oriundos de outros nós, já que em princípio possui uma alta taxa de disponibilidade de CPU. A seqüência lógica desse método é mostrado a seguir e ilustrado na Figura 8.2:

1) O administrador de sistemas ativa a rotina de carga no novo nó.
2) A rotina de carga de um novo nó consiste em carregar os serviços básicos (agentes, persistência, balanceamento e segurança).
3) Este novo nó deve ser cadastrado na base de configuração.
4) O agente mensageiro responsável pela disseminação do conteúdo da base de configuração deve ser ativado.

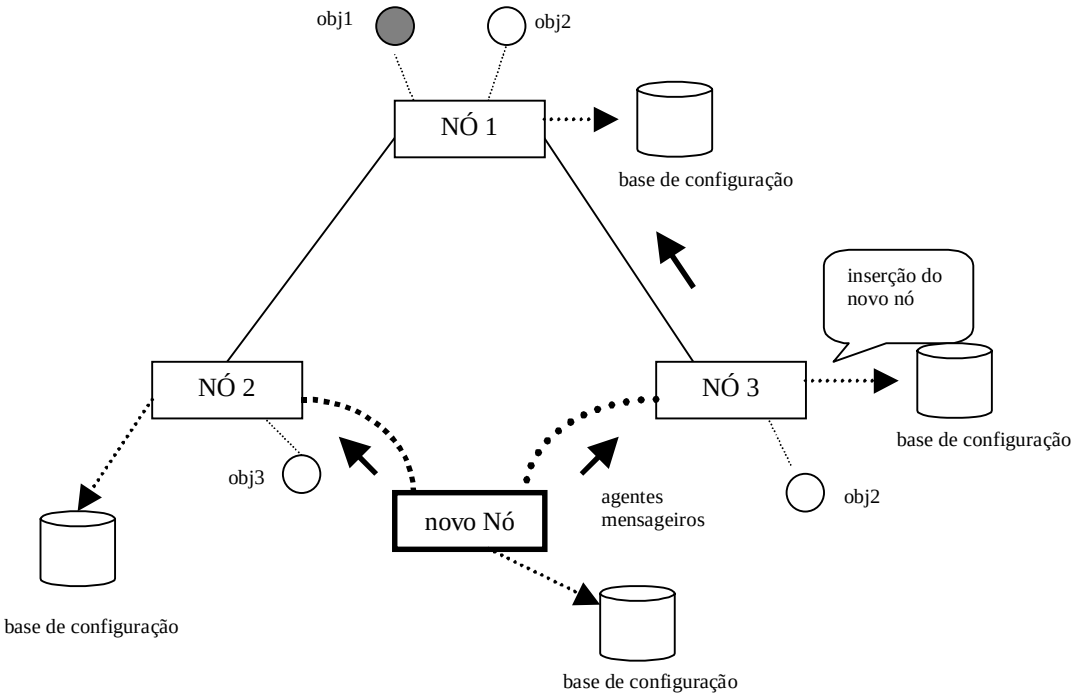


Fig 8.2 – Ilustração da inserção de um novo nó.

A Figura 8.2 elucida o processo de inserção de um ou mais nós. A rotina de carga do nó inserido instancia os agentes mensageiros, que apresentam-se como os responsáveis por distribuir e atualizar o conteúdo da base de configuração dos outros nós remotos.

8.4 - MÉTODO PARA REALIZAR A MANUTENÇÃO NO SISTEMA

O processo de manutenção em um dos nós, pertencentes ao domínio predefinido para a arquitetura SICSD, pressupõe a intervenção do administrador de sistemas. Várias atividades realizadas em um nó podem ser designadas como atividades de manutenção, dentre elas pode-se destacar: a instalação de um software ou a instalação de um novo periférico. O algoritmo desse método engloba as seguintes atividades:

1) Alterar o status do nó de “ready” para “suspend” com o auxílio do agente supervisor.
2) Ativar o método “realizar manutenção” do serviço de balanceamento responsável pelas seguintes tarefas:

2.1) Levantar todos os objetos instanciados nesse nó com o auxílio do agente supervisor.

Enquanto houver objetos instanciados nesse nó
início
Identificar o nó com maior disponibilidade de CPU.
Criar uma cópia do objeto nesse nó.
Acionar o serviço dos agentes para instanciar um agente monitor desse Objeto.
fim-enquanto

2.2) Liberar o nó para efetivar a manutenção após o término da migração dos objetos.

A Figura 8.3 ilustra o processo de desativação do nó 3 para uma possível manutenção. O método de manutenção do serviço de balanceamento se responsabiliza pela instanciação de todos os objetos existentes nesse nó em outros nós remotos, que se apresentem com a maior taxa de disponibilidade de CPU.

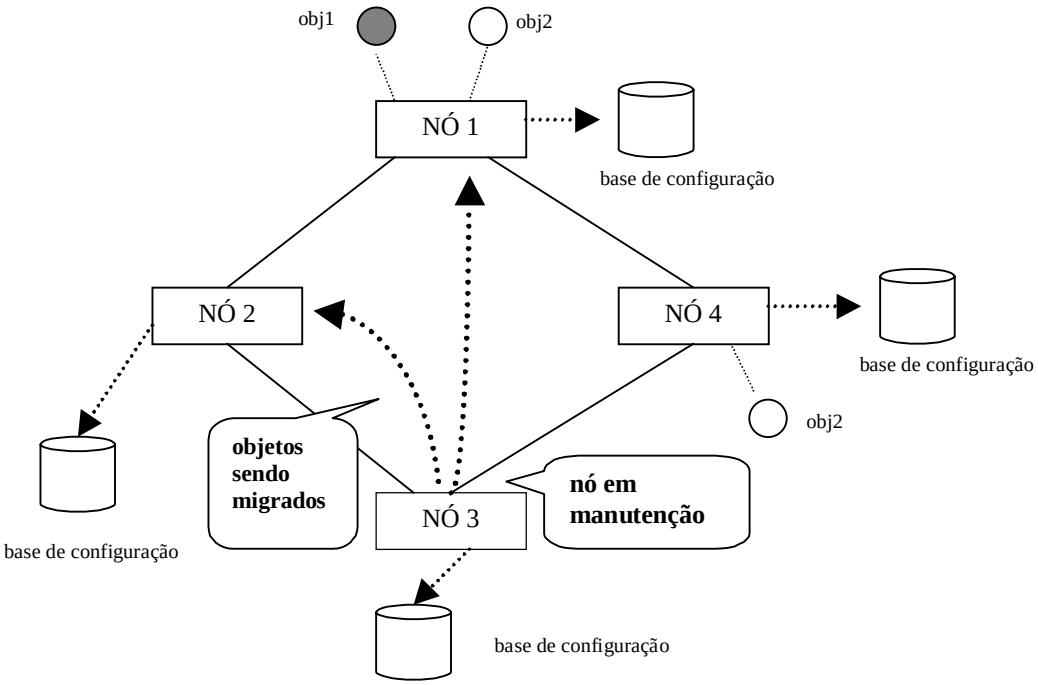


Fig. 8.3 – Ilustração do processo de manutenção em um nó.

8.5 - MÉTODO ATENDER SOLICITAÇÃO DO USUÁRIO

O serviço de balanceamento apresenta-se como o responsável pela interface entre a arquitetura proposta e o usuário. Após a identificação do usuário, o serviço de balanceamento disponibiliza para o mesmo um conjunto de funções, de acordo com o seu perfil (por exemplo, o controlador de satélites compreende um conjunto de funções diferentes de um engenheiro de satélites).

O serviço de balanceamento analisa a solicitação de serviço do usuário e o processo de atendimento segue os seguintes critérios: Primeiro verifica-se, no nó local, se existem objetos que atendam à solicitação desejada. Em seguida verifica se a taxa de disponibilidade de CPU do nó apresenta-se acima da média. Em caso afirmativo, deve ser estabelecida a conexão no próprio nó. Caso contrário, faz-se uma busca na base de configuração (tabela de objetos) e recuperam-se aqueles objetos que atendam ao serviço solicitado. Estabelece-se a conexão com o objeto que esteja instanciado no nó que apresentar maior disponibilidade de CPU. Esse parâmetro se encontra na *tabela de nó* (Tabela 7.1).

Algoritmo do método atender solicitação

Acessar a base de configuração com o auxílio do agente supervisor.

Se objeto existe no nó local e a disponibilidade de CPU do nó local for acima da média
Então
- Conectar-se com o nó local.
- Informar ao agente supervisor o nó local e o objeto solicitado, com o objetivo de atualizar a base de configuração, em relação ao número de conexões com esse objeto.

Senão
Se existir um nó que contenha disponibilidade de CPU e uma instância do objeto desejado
então
Conectar-se com o nó remoto.
Informar ao agente supervisor o nó remoto e o objeto solicitado, para atualizar a base de configuração, em relação ao número de conexões com esse objeto.
Incrementar o número de conexões para esse objeto.
Senão
Chamar o método **criar objeto.**
Fim-se
Fim-se

A Figura 8.4 ilustra o processo de atendimento de serviço para um usuário. O usuário necessita dos serviços do objeto denominado “X”. Nesse caso, não existe instância desse objeto no nó onde o usuário está conectado (nó 1). Portanto, a ação do serviço de balanceamento concentra-se em identificar quais outros nós remotos possuem instâncias desse objeto. No caso, os nós 2 e 4 mostram-se aptos para atender a essa solicitação; entretanto, estabelece-se a conexão com o nó 4, já que ele possui a maior taxa de disponibilidade de CPU.

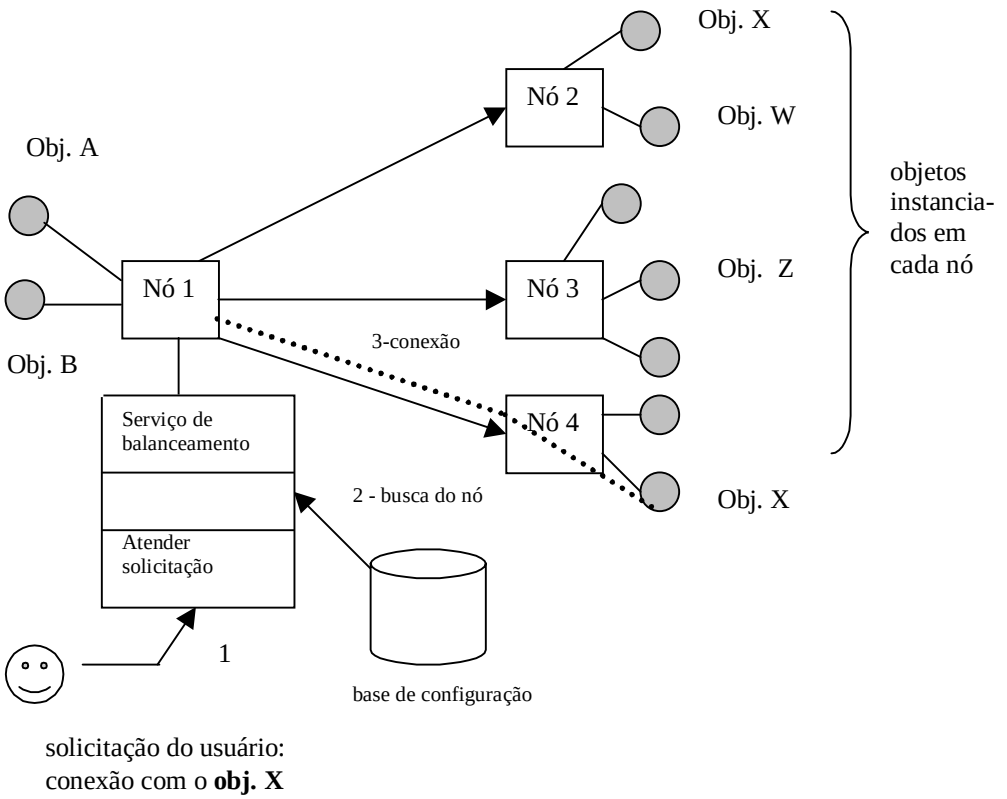


Fig. 8.4 -Processo de solicitação de um serviço do usuário.

8.6 - MÉTODO EXCLUIR OBJETO

A implementação de mecanismos que, excluam objetos que não estão sendo mais referenciados pelo sistema, não é uma atividade trivial em ambientes distribuídos. A Arquitetura DCOM implementa o "*Garbage Collection*", utilizando contadores que indicam o número de vezes que um determinado objeto está sendo referenciado no contexto do sistema distribuído. Ou seja, incrementa-se o contador para toda solicitação de serviço recebido por esse objeto. Ao termino da operação, decrementa-se este contador. Objetos que não contêm nenhuma referência (contador=0) devem ser excluídos. A perda de conexão entre um objeto cliente e um objeto servidor, ocasionada por exemplo por uma falha na rede, pode inviabilizar a contabilização do número de referências do objeto servidor, que, no caso, deveria ser decrementada.

A arquitetura RMI implementa um temporizador para controlar as referências dos objetos. Em intervalos de tempos predefinidos um algoritmo verifica se um determinado

objeto está sendo referenciado. Em caso negativo, o algoritmo assume que não há mais usuários utilizando os serviços prestados por esse objeto; portanto, ele pode ser excluído. A limitação desse algoritmo também se relaciona com eventuais falhas na rede. Como por exemplo, uma falha na rede, no momento do pedido de desconexão do objeto cliente com um objeto servidor, pode inviabilizar o decremento do número de referências do objeto servidor.

O algoritmo assume que um objeto servidor deve ser excluído se a comunicação entre esse objeto e um objeto cliente for interrompida por um tempo superior ao intervalo de tempo definido no algoritmo.

Conclui-se, então, que o desenvolvimento de algoritmos "garbage collection" não se caracteriza por uma atividade trivial. Talvez, por essa complexidade, a OMG sugeriu que a aplicação assumisse o controle do ciclo de vida (criação e destruição) das instâncias de seus objetos. (Pedrick, et al, 1998).

A arquitetura SICSD, através do serviço de balanceamento, presta o serviço "carbage collection" desde que o serviço dos agentes mantenha atualizado, na *base de configurações*, o número de conexões ativas de um objeto. A contabilização do número de conexões ativas de um objeto, na arquitetura SICSD, apresenta-se similar ao proposto na arquitetura DCOM: incrementa-se o contador a cada solicitação de serviço e decrementa-se a cada desconexão.

O método *excluir objeto* é acionado em cada nó, em um intervalo de tempo predefinido, e sua função básica consiste em "destruir" os objetos que apresentarem o valor nulo ou zero no atributo número de conexões. A diferença do método proposto *excluir objeto*, em relação ao método "Garbage collection" da arquitetura DCOM, reside no processo de decrementar o número de referências de um objeto. No processo de desconexão de um serviço, atualiza-se primeiro a base de configurações local, onde está instanciado o objeto, e posteriormente a base de configuração remota. Uma falha na rede, justamente no momento de decrementar o número de referências do objeto servidor remoto, é coberta pela atualização da base local. Atribui-se aos agentes mensageiros a responsabilidade da disseminação do conteúdo dessa base local para os outros nós remotos.

Algoritmo do método excluir objeto:

Identificar todos os objetos instanciados em um nó, com o auxílio do agente supervisor.

```
Enquanto houver objetos instanciados neste nó
    início
     Se o número de conexões = 0
      então
          excluir objeto.
         Acionar o serviço dos agentes para excluir o agente monitor desse objeto,
     fim-se
  fim-enquanto
```

8.7 - MÉTODO CRIAR OBJETO

Como já foi citado anteriormente, atribui-se a responsabilidade de criar os objetos da aplicação à rotina de carga do sistema. Entretanto, no período entre as passagens de um satélite, os objetos da aplicação para controle de satélites podem se tornar ociosos e, conseqüentemente, ser excluídos pelo método "excluir objetos". O serviço de balanceamento, através de seu método "criar objeto", apresenta-se capacitado a instanciar novamente um objeto. O método "atender solicitação do usuário", apresentado na seção 8.5, pode utilizar os serviços prestados pelo método "criar objeto", caso o serviço de balanceamento não encontre o objeto que atenda à solicitação do usuário.

Algoritmo do método criar objeto:

```
Se a taxa de disponibilidade de CPU do nó local está acima da média
 Então
      Criar uma instância do objeto no nó local.
      Acionar o serviço dos agentes para instanciar um agente monitor desse
      Objeto.
 Senão
      Criar uma instância do objeto no nó que apresentar maior taxa
      disponibilidade de CPU.
     Acionar o serviço dos agentes para instanciar um agente monitor desse
     Objeto.
fim-se
```

8.8 - COMPARAÇÃO ENTRE O BALANCEAMENTO DE CARGA DA PROPOSTO NA ARQUITETURA CORBA E O PROPOSTO NA ARQUITETURA SICSD

a) Arquitetura CORBA

O balanceamento de carga, na maioria dos aplicativos que implementam a especificação CORBA, caracteriza-se por uma distribuição seqüencial de carga. Ou seja, se o sistema distribuído contém mais de um objeto, para atender a uma mesma solicitação de um usuário, é estabelecida a conexão de forma seqüencial. Primeiro com o objeto 1 depois com o objeto 2 etc. Ao alcançar o último objeto, retorna-se para o primeiro. Esse processo de conexão não leva em consideração a carga de processamento no nó onde o objeto está instanciado. A Figura 8.5 elucida as solicitações de serviço de um usuário, considerando que no ambiente distribuído exista mais de um objeto para atender à mesma solicitação.

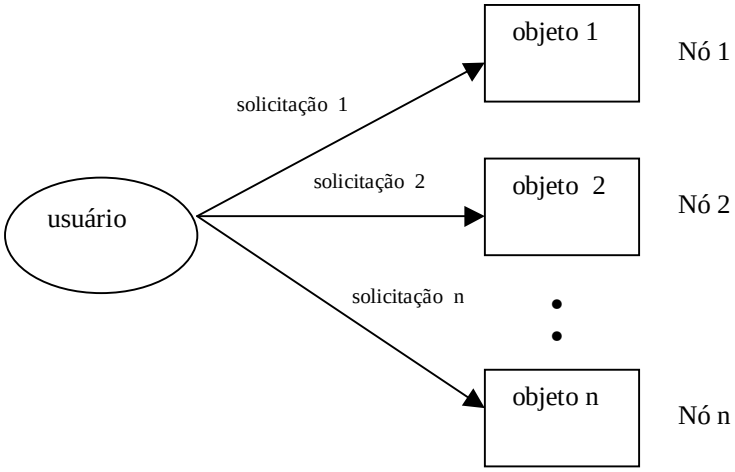


Fig. 8.5 – O processo de balanceamento de carga sugerido na especificação CORBA.
FONTE: adaptada do Visibroker, (1998, p. 14-3).

b) Arquitetura SICSD

A Arquitetura Proposta considera duas formas distintas, realizadas pelo serviço de balanceamento, na busca por uma boa distribuição de carga do sistema. A primeira forma anunciada na seção 8.1, concentra-se basicamente em migrar ou replicar objetos de máquinas saturadas para máquinas ociosas. A segunda objetiva distribuir a carga de processamento, desde o processo de atendimento da solicitação de serviço de um usuário. Ou seja, de acordo com a arquitetura proposta, o processo de atendimento das solicitações de serviços dos usuários utiliza-se de um algoritmo capaz de localizar qual o objeto mais apto para atender a essa solicitação. A seção 8.5 apresentou os detalhes desse algoritmo.

8.9 - COMPARAÇÃO ENTRE O SERVIÇO DE NOMES DA ESPECIFICAÇÃO CORBA E O SERVIÇO DE LOCALIZAÇÃO DE UM OBJETO PELO SERVIÇO DE BALANCEAMENTO

A base de configuração, mais especificamente a tabela de objetos, mantém um controle de todos os objetos instanciados em cada nó. Serviço semelhante (name service) já se encontra disponível nas arquiteturas que implementam a especificação CORBA. O ganho proposto neste trabalho de pesquisa não se restringe apenas em identificar a localização física de um objeto, mas identificar outras informações, que são agregadas, tais como:

- o status do objeto: “ready”, “failed”or “suspend”;
- o número de conexões ativas que esse objeto está atendendo em um determinado instante etc.

A arquitetura proposta mantém um controle de todos os objetos instanciados em cada nó, através da base de configuração (*tabela de objetos)*. A arquitetura CORBA também tem um controle de todos os objetos instanciados, em um ambiente distribuído, através do serviço de nomes “name service”.

O serviço de nomeação CORBA apresenta-se como um serviço genérico de diretório análogo a uma lista telefônica, isto é, se o cliente possui o nome do objeto, ele pode, através desse serviço, recuperar a sua referência (Mowbray, Ruh, 1997). As operações chaves do serviço de nomeação são de “ligar” (“bind”) e “resolver” (“resolve”). Utiliza-

se a operação de “ligar” para adicionar um nome de objeto e sua referência no diretório do serviço. Na operação de “resolver”, o cliente entra com um nome de objeto particular e recebe uma referência de objeto, como valor de retorno, ou uma exceção, caso o nome não se encontre no diretório.

As diferenças entre o serviço de resolução de nomes propostos pela arquitetura CORBA e o serviço de localização de um objeto proposto pelo serviço de balanceamento são conforma apresentado a seguir:

- A especificação CORBA recupera o primeiro objeto disponível que atenda determinada solicitação de serviço; o serviço de balanceamento proposto seleciona o objeto que se encontrar instanciado em um nó, com a maior taxa de disponibilidade de CPU. Teoricamente esse objeto apresenta-se mais apto a prestar serviços do que os outros objetos que estejam instanciados em nós com uma menor taxa.
- Na especificação CORBA, se o sistema distribuído contém mais de uma instância de um objeto para atender a uma mesma solicitação de serviço, é estabelecida a conexão de forma seqüencial. Primeiro, com o objeto 1, depois com o objeto 2, etc. Na arquitetura proposta, o serviço de balanceamento seleciona o objeto instanciado em um nó, com a maior taxa de disponibilidade de CPU.

8.10 - A IMPORTÂNCIA DO SERVIÇO DE BALANCEAMENTO

A heterogeneidade das máquinas que compõem um ambiente distribuído constitui um parâmetro que sempre deve ser levado em consideração, quando se deseja otimizar o balanceamento de carga de um sistema. Partindo desta premissa, o serviço de balanceamento proposto apresenta mecanismos de como explorar os recursos computacionais existentes. Esses mecanismos estão implementados dentro dos principais métodos do serviço de balanceamento (balancear, inserir nó, remover nó e realizar manutenção).

O resultado esperado da execução de qualquer um dos métodos do serviço de balanceamento consiste em otimizar a distribuição de cargas do sistema. O método *balancear* assume a responsabilidade de migrar ou replicar objetos de máquinas

saturadas para máquinas ociosas. O método *inserir nó* tem como conseqüência imediata incrementar recursos computacionais no ambiente predefinido para a arquitetura SICSD. O nó inserido passa a ser um candidato em potencial a receber a migração de objetos de máquinas saturadas. Apesar de eliminar recursos computacionais, no processo de exclusão de um nó, faz-se necessário o balanceamento de carga, já que esse método dispara um conjunto de ações, tentando criar nos nós remotos os objetos que estavam instanciados no nó excluído. O método *realizar manutenção* tem funções semelhantes ao método excluir nó, a diferença reside no fato de que aciona-se o método excluir nó de forma imprevisível, já o método realizar manutenção é previsível e acionado pelo administrador de sistemas. Entretanto, em ambos os casos, as funcionalidades dos métodos se traduzem em buscar o balanceamento de carga, migrando os objetos do nó que está sendo excluído para máquinas ociosas.

CAPÍTULO 9

O SERVIÇO DE PERSISTÊNCIA

Quantas maças não caíram na cabeça de Newton antes de ele ter percebido a dica. (Robert Frost)

O serviço de persistência disponibiliza, para os objetos persistentes da aplicação, os serviços de acesso ao banco de dados (armazenamento, recuperação e exclusão). Os objetos persistentes da aplicação não precisam possuir a implementação destes serviços; pois, simplesmente eles acionam, através de uma mensagem, o serviço de persistência. Este apresenta-se como o responsável por determinar onde deve ser armazenado esse objeto.

Como a arquitetura proposta apresenta-se dinâmica, os objetos persistentes nem sempre permanecem instanciados no nó onde eles devem ser armazenados. Desta forma, o objetivo do serviço de persistência consiste em tornar transparente o acesso e a localização do banco de dados e armazenar o objeto no nó que tenha a maior taxa de disponibilidade de I/O.

9.1 – SERVIÇO DE PERSISTÊNCIA PROPOSTO NA ESPECIFICAÇÃO CORBA

O Serviço de Persistência para Objetos (POS), da especificação CORBA, caracteriza-se por um serviço que permite armazenar objetos persistentes em um arquivo, em um Banco de Dados Relacional (BDR) ou em um Banco de Dados Orientado a Objeto(BDOO). Ele apresenta-se como um mediador entre o sistema de armazenamento e os objetos persistentes da aplicação (CORBAPersistence-1999) Figura 9.1.

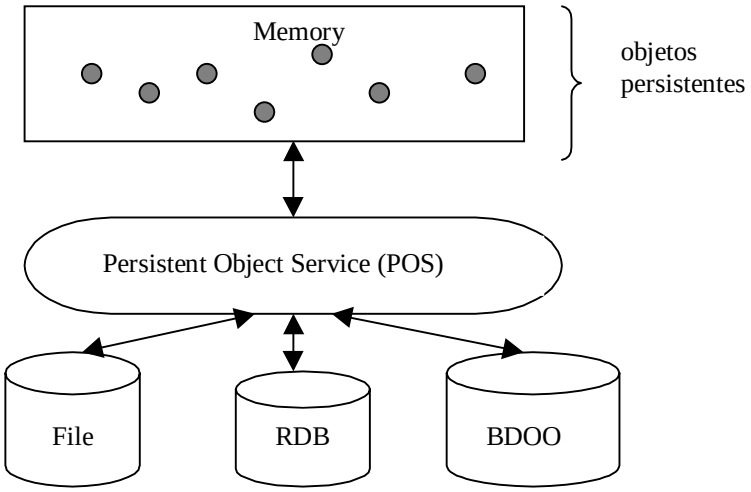


Fig. 9.1 - Serviço de Persistência para Objetos.
FONTE: Orfali (1996, p. 141).

Os elementos do Serviço de Persistência para Objetos (POS)

Os elementos do Serviço de Persistência da especificação CORBA são comentados a seguir e ilustrados na Figura 9.2.

a) **Pesistent Objects (PO):** Caracterizam-se pelos objetos persistentes da aplicação que devem ser armazenados. Cada objeto persistente da aplicação deve conter um identificador de persistência (PID), que descreve a localização da base de armazenamento do objeto, podendo ser um arquivo, BDR ou um BDOO.

b) **Persistent Object Manager (POM):** Responsável por tornar transparente para os objetos persistentes da aplicação a base de armazenamento utilizado. Ele funciona como um roteador, direcionando as chamadas de armazenamento dos objetos para suas respectivas bases de armazenamento. Como por exemplo, se a base de armazenamento utilizado for um arquivo, o serviço POM aciona o protocolo DA (Direct Attribute). A identificação do sistema de armazenamento se encontra encapsulada no PID.

c) **Persistent Data Services (PDS):** São as interfaces específicas para cada base de armazenamento. Esse serviço é o que realmente executa a tarefa de mover dados de um objeto para a base de armazenamento.

Esta camada implementa três protocolos:

- **Dynamic Data Object (DDO):** Este protocolo define a estrutura que contém os atributos de um objeto. O acesso aos dados armazenados pode ser feito diretamente por este protocolo. Sua funcionalidade concentra-se em mapear os objetos persistentes da aplicação para tabelas em um banco de dados relacional.
- **Direct Attribute (DA):** Este protocolo permite o mapeamento direto de um objeto para um arquivo. Cada objeto persistente contém uma interface com a lista de atributos que devem ser persistentes. Este protocolo utiliza uma extensão da linguagem IDL, denominada DDL (Data Definition Language), para definir os objetos persistentes da aplicação. Ele implementa, de uma forma simplificada, um Banco de Dados Orientado a Objeto (Orfali, Harkey, Edwards, 1996). Os objetos de uma mesma classe devem ser armazenados em um mesmo arquivo.
- **ODMG-93**: Este protocolo traduz o resultado do trabalho dos maiores fabricantes de banco de dados orientado a objetos - Object Database Management Group-ODMG-93. Ele implementa o acesso aos objetos armazenados, através de RPC (Remote Procedure Call), e conta com o auxílio da Linguagem de Definição de Objetos (ODL) e da linguagem de Consulta a Objetos (OQL). A linguagem ODL descreve os metadados de um BDOO, isto é, ela apresenta funcionalidades semelhantes à linguagem de definição de dados (DDL), utilizada em um banco de dados relacional. A manutenção dos objetos persistentes (armazenamento, recuperação e exclusão), em um BDOO, empregam as funcionalidades da linguagem OQL (Orfali, Harkey, Edwards, 1996).

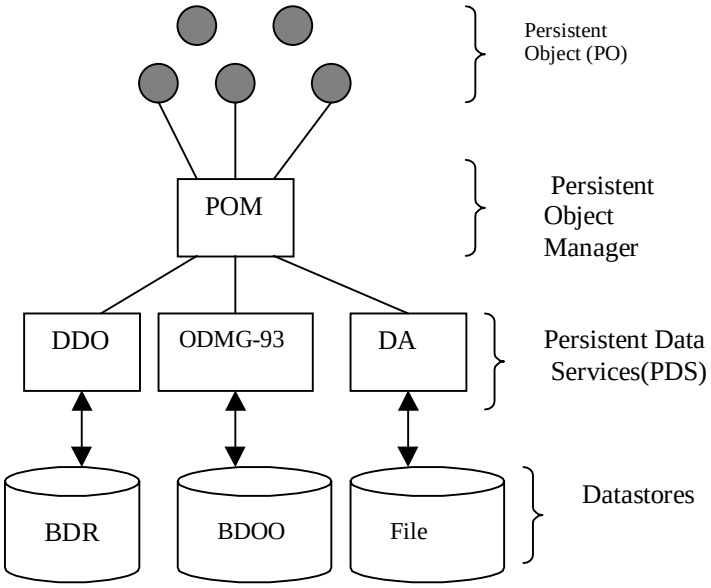


Fig. 9.2 - Serviço de Persistência de Objetos (POS).
FONTE: Orfali (1996, p. 145).

O Serviço de Persistência para Objetos (POS) não foi adotado pela maioria das empresas que implementam a especificação CORBA (IONA, IBM, Visionnaire etc.), conseqüentemente, este serviço foi **cancelado.**

Em agosto de 1999, as principais empresas desenvolvedoras de sistemas de banco de dados (ORACLE, IBM, SUN etc.) propuseram uma nova especificação para o serviço de persistência PSS (Persistent State Service, versão 2.0).

9.2 – PERSISTENT STATE SERVICE –PSS

A segunda versão do serviço de persistência surgiu principalmente para tornar mais transparente o acesso ao sistema de armazenamento para os objetos persistentes da aplicação. O PSS continua utilizando o serviço disponível de armazenamento da empresa (arquivos, BDR ou BDOO) (CORBAPersistence-1999).

De acordo com este novo serviço, a definição do *esquema* do banco de dados (a estrutura do banco de dados, os tipos de dados) pode ser efetivada, utilizando a Linguagem de Definição do Estado da Persistência (PSDL – Persistent State Definition Language). Esta linguagem apresenta-se como uma extensão da linguagem IDL,

proposta pela especificação CORBA. Sua função básica consiste em definir os atributos que serão armazenados no sistema de armazenamento para cada classe da aplicação, que contenha objetos persistentes. Duas etapas são necessárias para empregar esta linguagem no processo de armazenamento dos objetos persistentes. A primeira consiste em criar para cada classe persistente da aplicação um arquivo com extensão PSDL, contendo a descrição dos atributos que devem ser armazenados no banco de dados. A segunda caracteriza-se pela compilação e geração dos códigos necessários para armazenar/recuperar os objetos persistentes da aplicação. Um exemplo do conteúdo deste arquivo é mostrado a seguir.

ARQUIVO JAVA

Arquivo Pessoa.java

```
Class Pessoa {
        String nome;
        String CPF;
  }
```

ARQUIVO PSDL

Arquivo Pessoa.psdl

```
abstract storetype  Pessoa {
        String nome;
        String CPF;
  }
```

A nova versão do Serviço de Persistência foi definida em agosto de 1999, mas até o momento, os principais “brokers” que implementam a especificação CORBA ainda não implementaram este novo serviço. Além disso, alguns pontos podem ser levantados nesta nova versão, tais como:

- Existe um arquivo PSDL para cada classe da aplicação que contenha objetos persistentes. Alterações na classe da aplicação, como por exemplo a inclusão de um novo atributo, sem alterar o arquivo PSDL correspondente, pode comprometer a integridade de armazenamento dos dados.
- O processo de compilação do arquivo PSDL gera, de forma automática, o código empregado no armazenamento dos objetos persistentes da aplicação.

Com o objetivo de simplificar e facilitar o trabalho do desenvolvedor de software de armazenar/recuperar os objetos persistentes da aplicação, sugere-se criar um serviço de persistência capaz de:

- Suprimir a responsabilidade do desenvolvedor de mapear, para um arquivo PSDL, cada classe da aplicação que contenha objetos persistentes.

- Eliminar a necessidade da geração de código para os objetos persistentes, mesmo que haja alteração nos atributos de um objeto.

9.3 – O SERVIÇO DE PERSISTÊNCIA PROPOSTO

O Serviço de Persistência proposto disponibiliza para a aplicação o acesso ao sistema de armazenamento de dados, podendo este ser um arquivo, um BDR ou um BDOO. Os objetos da aplicação solicitam a esse serviço o acesso ao sistema de armazenamento de dados. O serviço faz o mapeamento deste objeto para um registro, se o sistema de armazenamento for um arquivo; para uma tupla se o sistema de armazenamento for um BDR e não tem transformações, se for um BDOO (Figura 9.3).

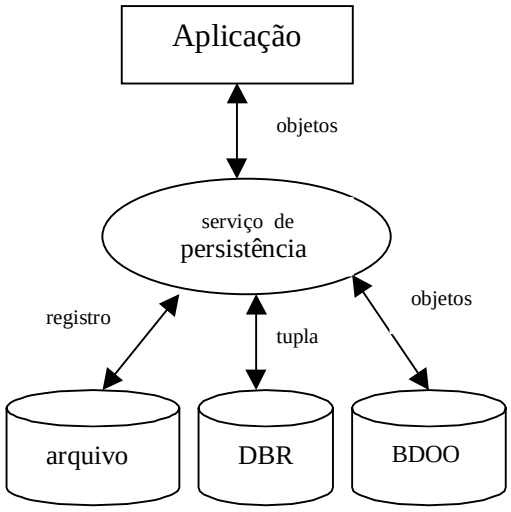


Fig. 9.3- Serviço de persistência proposto.

De acordo com o serviço de persistência proposto, os objetos persistentes da aplicação não necessitam conter código para as operações de armazenamento ou recuperação. No processo de armazenamento, o objeto envia uma mensagem para o serviço de persistência, ele se incumbe de gerar o PID (Identificador de Persistência) para este objeto. Já no processo de recuperação, informa-se apenas o PID do objeto desejado. O Serviço de Persistência gerencia onde devem ser armazenados/recuperados os objetos (Ferreira,1996).

Instanciado pela rotina de carga, esse serviço mantém-se fixo em cada nó da rede, aguardando as solicitações dos objetos para acessarem o sistema de armazenamento. Esse serviço contém uma base de dados, denominada “base de armazenamento”, previamente configurada pelo Administrador de Banco de Dados, contendo a relação dos objetos persistentes da aplicação e o nome da base de dados, onde eles devem ser armazenados. Uma mesma base de dados pode estar replicada em mais de um nó, como por exemplo, a base de dados do objeto persistente “telemetria” pode estar criada tanto no nó1 quanto no nó2.

O Serviço de Persistência atualiza uma dessas bases em um determinado nó e atribui ao Sistema Gerenciador de Banco de Dados Distribuído (SGBDD) a responsabilidade de manter a atualização dessas cópias de armazenamento em outros nós.

A intervenção do administrador de banco de dados faz-se necessária no processo da definição da localização da base de dados de cada objeto persistente por dois motivos. O primeiro caracteriza-se por definir quais serão os servidores de dados para a arquitetura SICSD. Ou seja, apesar de a arquitetura ser distribuída, os nós que se apresentam como servidores de dados, devem ser melhor configurados. Normalmente, possuem maior capacidade de processamento, memória e armazenamento em disco. O segundo motivo objetiva mensurar qual a disponibilidade desejável da base de dados de um determinado objeto persistente. Como por exemplo, pode-se determinar que a base de dados do objeto persistente telemetria deve estar disponível em pelo menos três nós servidores de dados, já que esse objeto, por definição, apresenta-se como o mais solicitado em uma aplicação para controle de satélites. Entretanto, a base de dados de outros objetos, como por exemplo o telecomando, deve estar replicada em mais de um nó, para garantir somente a disponibilidade, já que somente um usuário pode enviar telecomandos para o satélite ao mesmo tempo. A Tabela 9.1 elucida um exemplo da base de armazenamento.

TABELA 9.1 –BASE DE ARMAZENAMENTO DO SERVIÇO DE PERSISTÊNCIA

Objeto	**Base de dados**	**Nó de armazenamento**
Telemetria	tab_telemetria	Patras
Telecomando	tab_telecomandos	Patras
Ranging	tab_ranging	Pelion
Telemetria	tab_telemetria	Andros
Telecomando	tab_telecomandos	Andros
Telemetria	tab_telemetria	Pelion

Comentários de alguns campos da Tabela 9.1:

- O objeto “telemetria” dever ser armazenado na base de dados, cujo nome é tab_telemetria. Esta base se encontra nos nós Patras, Andros e Pelion. O Serviço de Persistência atualiza somente uma dessas bases, por exemplo, no nó Patras. O SGBDD apresenta-se como o responsável por fazer a atualização nos nós Andros e Pelion.
- O objeto “telecomando” deve ser armazenado na base denominada tab_telecomando.

9.4 – FUNCIONAMENTO DO SERVIÇO DE PERSISTÊNCIA PROPOSTO

O Serviço de Persistência apresenta-se como o mediador entre a aplicação e o sistema de armazenamento de dados. Ele recebe mensagens dos objetos da aplicação para acessar o banco de dados, recupera do repositório de interfaces a descrição da interface deste objeto e finalmente interage com o serviço dos agentes para identificar o nó que contenha a melhor taxa de disponibilidade de I/O para armazenar o objeto. No processo de recuperação de um objeto, decorrem os mesmos passos do processo de armazenamento, a diferença reside no fato de que a recuperação requer a intervenção de um usuário, ou outro objeto, para que o processo se inicialize. A seqüência dessas operações são mostradas na Figura 9.4.

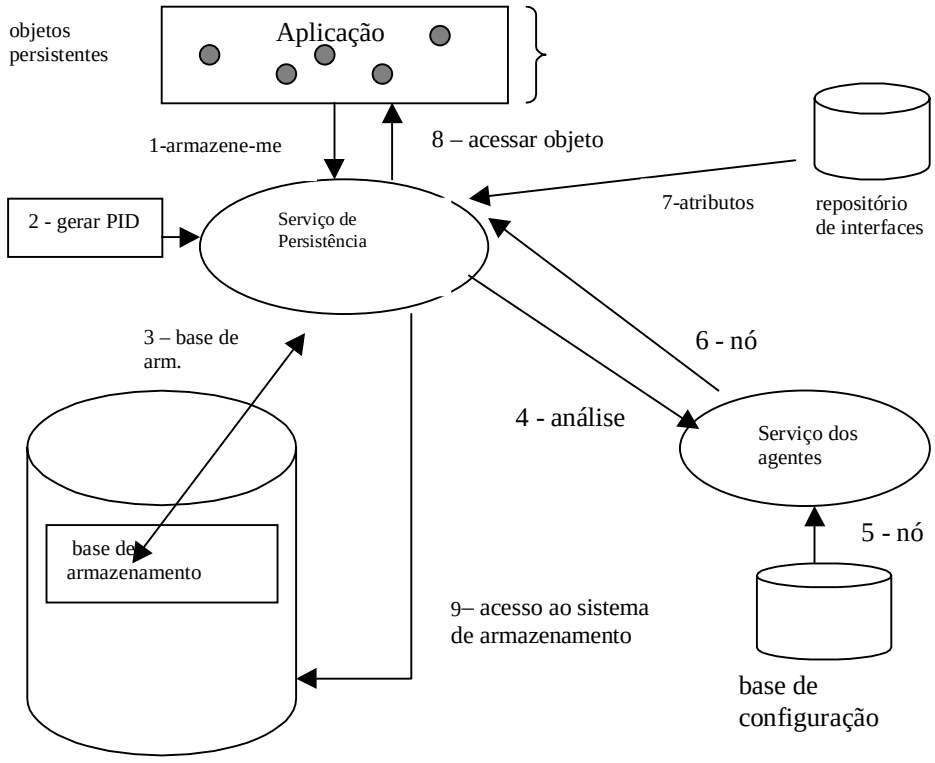


Fig. 9.4 – Funcionamento do Serviço de Persistência.

Seqüência de operações:

1- A rotina “start” de cada nó, definido para instalação da arquitetura proposta, responsabiliza-se pela carga dos serviços dos agentes, balanceamento, segurança, persistência e pelos objetos da aplicação para controle de satélites. Uma vez instanciados, os objetos da aplicação podem utilizar os serviços prestados pelo serviço de persistência. Os objetos acionam, através de mensagens, o serviço de persistência, e este responsabiliza-se pelo acesso ao sistema de armazenamento de dados. Como por exemplo, a rotina “start” cria uma instância do objeto telemetria. Os “frames” de telemetria que chegam do satélite são processados pelo objeto telemetria e, em seguida, devem ser armazenados no banco de dados. O processamento do “frame” consiste basicamente em verificar o estado interno do satélite (quais chaves estão ligadas, a voltagem da bateria etc.). O objeto

"telemetria" aciona, através de mensagens, o serviço de persistência para armazenar o conteúdo do frame.

2- O serviço de persistência recebe a mensagem do objeto da aplicação. Se a operação for de armazenamento gera-se o PID (Persistent identification). O próximo passo consiste em identificar a base de armazenamento para esse objeto.

3- Acessando a *base de armazenamento,* recupera-se a identificação da base de dados onde deve ser armazenado o objeto. Como já foi mencionado, atribui-se ao Administrador de banco de dados a tarefa de manter atualizadas as informações contidas nessa base. O processo de recuperação consiste em disponibilizar para o serviço de persistência a relação de todos os possíveis nós onde esse objeto pode ser armazenado.

4- A análise do melhor nó para armazenar esse objeto depende da taxa de disponibilidade de I/O em cada nó. Obtém-se esta informação consultando o serviço dos agentes que, dentro do contexto da arquitetura proposta, assume o papel de gerente da base de configuração do sistema.

5- O serviço dos agentes consulta a base de configuração, mais especificamente a *tabela de nós,* que contém a relação da taxa de disponibilidade de I/O de todos os nós do sistema distribuído. Recupera-se o nome do nó que apresentar a maior taxa.

6- O serviço de persistência recebe o nó mais indicado para fazer a operação de acesso ao Sistema de armazenamento de dados.

7- O serviço de persistência recupera do repositório de interfaces os atributos, o tipo de cada um deles, bem como os métodos de acesso a esse objeto na memória. Detalha-se este processo na seção 9.5.

8- O serviço de persistência acessa o objeto na memória. A interação entre o serviço de persistência e o objeto persistente apresenta-se detalhada na seção 9.5.

9- O serviço de persistência acessa o Sistema de armazenamento de dados, realizando o serviço desejado (armazenamento, recuperação ou exclusão).

9.5 - A INTERFACE ENTRE O SERVIÇO DE PERSISTÊNCIA E O REPOSITÓRIO DE INTERFACES

O repositório de interfaces apresenta-se como um gerenciador de interfaces de todos os objetos distribuídos pelo sistema. As informações geridas pelo repositório estão disponibilizadas para todos os objetos e podem ser acessadas em tempo de execução. O repositório mantém as informações armazenadas de forma hierárquica. Em uma primeira camada o repositório disponibiliza as interfaces de todos os objetos com os tipos de dados para cada atributo; em seguida, os métodos de acesso e, finalmente, caso exista, os parâmetros utilizados por cada método.

A interface de acesso dinâmico habilita qualquer objeto acessar um outro objeto, sem ter um conhecimento prévio de sua interface, ou seja, um objeto pode explorar, em tempo de execução, os recursos de outro objeto, desde que a interface do segundo esteja armazenada no repositório de interfaces.

A utilização das informações gerenciadas pelo repositório de interface, no acesso dinâmico a um objeto, permite aos objetos da aplicação algumas facilidades, tais como:

- Pode-se utilizar outros serviços dos objetos, além daqueles previamente definidos no processo de compilação; basta, para isso acessar o repositório.

- Objetos que utilizam uma interface dinâmica para explorar os serviços de outro objeto servidor não precisam ser compilados, caso haja alteração no código do objeto servidor.

Como já foi citado anteriormente, existe somente uma instância do serviço de persistência em cada nó da rede; portanto, esse serviço recebe solicitações para acessar o sistema de armazenamento de diferentes tipos de objetos persistentes. Cada objeto contém seus próprios atributos, com diferentes tipos. Por exemplo, o objeto telemetria contém os atributos “tempo”, que é do tipo *time* e “frame”, que é do tipo *vetor de bytes*. O objeto “ranging” contém os atributos “estação de origem”, que é do tipo *string* e “medidas de distância”, que é do tipo *vetor de números reais*. Além disso cada objeto contém seus próprios métodos de acesso aos seus atributos, tanto para ler o conteúdo de seus atributos (gets) quanto para atribuir novos valores (sets).

Considerando que, em tempo de execução, o serviço de persistência só recebe como parâmetro o nome do objeto que solicita o serviço de acesso ao sistema de armazenamento, faz-se necessário implementar mecanismos que viabilizem a interação entre o serviço de persistência e o objeto solicitante. Essa interação abrange três etapas. A primeira, quando o serviço de persistência recebe uma solicitação de serviço de um objeto persistente. Neste caso, a primeira providência tomada pelo serviço consiste em acessar o repositório de interfaces para identificar:

- os métodos de acesso a esse objeto;
- os atributos e
- o tipo de dados de cada atributo.

A segunda etapa caracteriza-se pelo acesso do serviço de persistência a esse objeto na memória. Nesse processo, o serviço de persistência utiliza as informações recuperadas desse objeto do repositório de interface, ou seja, o serviço de persistência faz um acesso direto a esse objeto, através de seus métodos de acesso, recuperados da interface de repositório. Finalmente, o serviço de persistência mostra-se capaz de construir para cada objeto, a SQL (Structured Query Language), de acordo com os atributos e seus respectivos tipos. A Figura 9.5 elucida a interface entre o serviço de persistência e o repositório de interfaces.

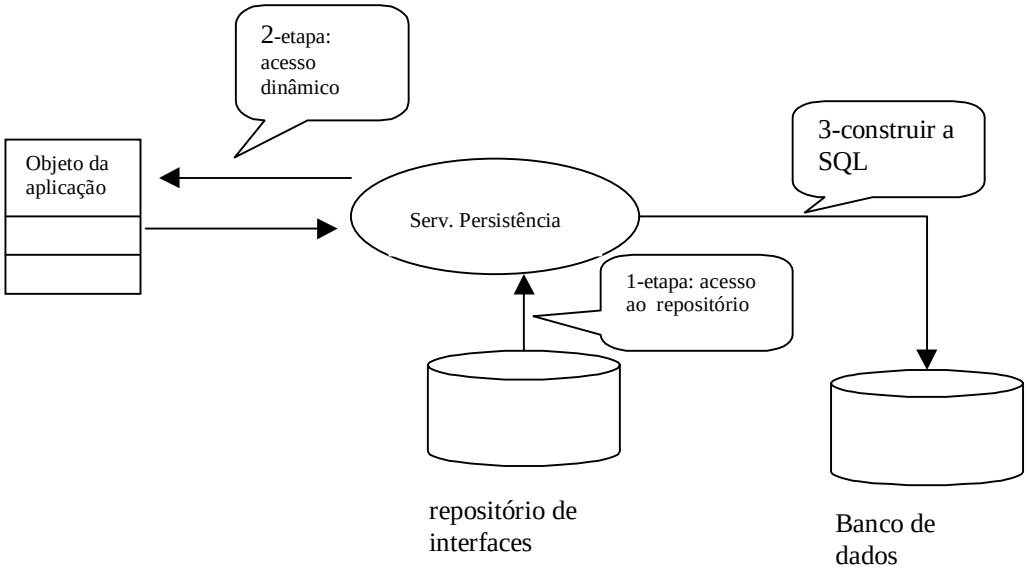


Fig. 9.5- Interface entre o Serviço de persistência e o repositório de interfaces.

9.6 – A IMPORTÂNCIA DO SERVIÇO DE PERSISTÊNCIA PROPOSTO

O objetivo atual da equipe de desenvolvimento de software nas empresas consiste na análise, projeto e implementação dos objetos de negócios. Estes são objetos específicos de uma aplicação para atender aos requisitos dos usuários. Por exemplo, uma telemetria com o seu código, sua descrição, seu limite inferior e limite superior pode ser considerada um objeto de negócio de uma aplicação para controle de satélites. Os outros objetos que esta aplicação contém, como por exemplo objetos que fazem a interface com o usuário ou acessam um sistema de armazenamento de dados, podem ser encontrados mais freqüentemente no mercado.

A especificação CORBA propõe que para cada classe da aplicação, que contenha objetos persistentes, exista um arquivo PSDL. Esse arquivo, após ser compilado, gera o código necessário para o armazenamento de objetos persistentes dessa classe. Percebe-se, então, que o desenvolvedor precisa criar duas interfaces para um mesmo objeto, uma para disponibilizar seus serviços para a aplicação em tempo de execução (IDL), outra para acessar o sistema de armazenamento (PSDL). O Serviço de Persistência CORBA necessita de um controle rígido de configuração de versões das classes desenvolvidas no

sistema para garantir que mudanças nos atributos de classes, que contenham objetos persistentes, sejam também refletidas nos respectivos arquivos PSDL.

O Serviço de Persistência proposto elimina a necessidade do desenvolvedor de software construir os arquivos PSDL. O desenvolvedor concentra esforços na especificação dos objetos de negócio. Cada objeto de negócio contém métodos de acesso ao sistema de armazenamento de dados. O código interno desses métodos contém simplesmente uma solicitação de serviço ao Serviço de Persistência Proposto. No serviço proposto, o mapeamento de classes, que contém objetos persistentes, é feito através da configuração e não da programação. Ou seja, caso mude a base de dados de armazenamento de uma classe, basta alterar a *base de armazenamentos*, a qual mostra-se configurável. Não é necessário alterar os códigos da aplicação.

No serviço PSS, mudanças nos atributos de uma classe, ou na base de armazenamento, implicam alterações no código e, conseqüentemente, uma recompilação.

O número de objetos instanciados no Serviço Proposto praticamente se mantém, isso porque é instanciado um único serviço de persistência em cada nó do sistema. Caso haja falha nesse serviço, os objetos persistentes desse nó poderão utilizar os recursos do serviço de persistência de outros nós, dentro do domínio de rede, configurado para a arquitetura SICSD.

Finalmente, talvez a grande contribuição do serviço de persistência proposto esteja na capacidade desse serviço explorar recursos já existentes no Repositório de Interfaces. A partir das informações geridas pelo Repositório de Interfaces, o serviço de persistência tem a capacidade de acessar os dados de um objeto em tempo de execução, montar a SQL de conformidade com os atributos e seus respectivos tipos e, finalmente, armazená-los em um banco de dados.

O ganho de explorar o conteúdo do Repositório de Interfaces reside no fato de que as alterações nos atributos de um objeto são imediatamente refletidas no repositório de interfaces e automaticamente visíveis para o serviço de persistência. Ou seja, não é necessária a compilação do serviço de persistência. Entretanto, mudanças nos atributos

de um objeto, utilizando o serviço de persistência CORBA necessitam da compilação e geração de códigos para realmente refletirem as alterações realizadas.

CAPÍTULO 10

O SERVIÇO DE SEGURANÇA

Mire no final e nunca pare para duvidar; nada é tão difícil, mas a busca irá descobri-lo. (Robert Herrick)

A implementação de mecanismos de segurança tem sido considerada em diversos sistemas computacionais. Contudo, no contexto de sistemas distribuídos, problemas de segurança, decorrentes da própria característica de distribuição, devem ser analisados e tratados adequadamente. A funcionalidade presente em sistemas distribuídos, oferecida pela interconexão de elementos computacionais (programas, arquivos etc.), apresenta-se como fonte causadora de alguns problemas, ou seja, como manter a privacidade e a integridade de dados que trafegam pela rede e como garantir a confiança no acesso aos elementos computacionais de software. Coulouris, Dollimore e Kindberg (Coulouris, 1994) identificaram quatro requisitos necessários ao tratamento da segurança em sistemas distribuídos:

- Os canais de comunicação devem estar seguros contra espionagem e modificação indevida do conteúdo de mensagens.
- Os servidores devem estar aptos a verificar a identidade de seus clientes.
- Os clientes devem estar capacitados a verificar a autenticidade de servidores.
- Deve ser possível verificar a identidade do remetente de uma mensagem, após sua passagem por um outro computador.

As ameaças a um sistema de segurança, que possibilitam a violação das restrições de acesso a informações e recursos, podem ser classificadas em :

- *Leakage*: Aquisição ilegal de informações.
- *Tampering*: Alteração indevida de informações, inclusive programas.
- *Resource Stealing:* Uso de facilidades sem autorização.

- ♦ *Vandalism:* Interferência no sistema sem objetivo de ganho algum, podendo, por exemplo, torná-lo indisponível.

O acesso ilegal a um sistema distribuído, para efetuar algumas das violações citadas, pode ser conseguido através da obtenção de acesso ao canal de comunicação ou do estabelecimento de uma conexão falsificada, com algum componente do sistema (arquivo ou programa). Os ataques ao sistema podem ser efetuados da seguinte forma (Coulouris, Dollimore, Kindberg, 1994):

- *Eavesdropping*: Obtenção indevida de informações pela cópia de mensagens, que são transmitidas através da rede, ou pelo exame de dados armazenados, inadequadamente protegidos. Por exemplo, observando pacotes não criptografados, que trafegam pela rede, ou examinando um arquivo de senhas desprotegido, podem ser conseguidas identificações e senhas de usuários.
- *Masquerading*: Uso de identidade alheia no envio e recebimento de mensagens. Utilização de identificações para burlar procedimentos de autenticação e obter acessos privilegiados.
- *Replay:* Armazenamento de mensagens para posterior reenvio. Pode ser usado para vandalismo, lotando o receptor de mensagens.
- *Data Manipulation*: Dano à integridade de dados armazenados ou transmitidos pela rede. Modificação do conteúdo de uma mensagem, provocando um comportamento não esperado do receptor.

As infiltrações em sistemas distribuídos, a fim de lançar algum tipo de ataque, podem ser realizadas por (Coulouris, Dollimore, Kindberg, 1994):

- *Password Cracking*: Descoberta de senhas, através de tentativas manuais ou uso de programas próprios. Regras para o estabelecimento de senhas difíceis de serem descobertas podem ser adotadas.
- *Virus:* Código anexado a um programa legítimo, que se auto-instala quando o programa legítimo é executado e, posteriormente, replicando. Devem ser usadas ferramentas para detecção e eliminação de vírus.

- *Trojan Horse*: Programa oferecido ao usuário para realizar uma tarefa útil, o qual, traz escondido uma outra função encapsulada. Um exemplo bastante comum consiste nos programas que simulam uma tela de *login* para capturar senhas.

Os mecanismos de segurança tentam buscar respostas para as seguintes perguntas:

- Quem está acessando o sistema? (Identificação)
- Pode-se estar seguro a respeito de quem fez esta requisição? (verificação)
- Pode-se garantir que essa mensagem não foi modificada após o seu envio? (Integridade)
- Esse usuário possui permissão para usar esse recurso? (controle de acesso)
- Pode-se enviar, com segurança, essa mensagem em uma rede pública? (privacidade)

O objetivo deste capítulo concentra-se em apresentar os requisitos básicos de segurança que devem ser incorporados à arquitetura SICSD:

- identificação e autenticação;
- política de proteção dos objetos;
- segurança no tráfego de mensagens e
- auditoria.

Considerando a existência de outros mecanismos de segurança, optou-se pelos citados anteriormente, por considerar que estes mostram-se suficientes para garantir que a arquitetura SICSD funcione em um ambiente confiável. Não estão sendo considerados outros mecanismos que podem ser também agregados, tais como “firewall”, salas especiais para a operação de um satélite, cartões magnéticos etc.

10.1 - IDENTIFICAÇÃO E AUTENTICAÇÃO

O mecanismo de identificação garante o acesso à arquitetura SICSD, somente de usuários previamente cadastrados e autorizados. A identificação de um usuário engloba três outras propriedades:

a) **A identificação de acesso:** A identificação de acesso expressa o tradicional "username", utilizado em qualquer sistema multiusuário;

b) **O grupo**: O objetivo dessa propriedade é agrupar os usuários da arquitetura SICSD em grupos. Os usuários de um grupo gozam dos mesmos privilégios, por exemplo, na arquitetura SICSD existem vários grupos dos quais pode-se citar:

CONSAT: Controlador de satélites;

ENSAT: Engenheiro de satélites;

RESOFT: Responsável pelo software;

DIROP: Diretor de operações e

DIRMIS: Diretor de missão

c) **Privilégios**: Este atributo descreve os privilégios atribuídos a cada grupo de usuários, por exemplo: os usuários CONSATs podem visualizar telemetrias, enviar telecomandos. Os usuários RESOFTs podem modificar a configuração do sistema, como cadastrar novos usuários, atribuir/excluir funções aos usuários etc.

O processo de autentificação objetiva garantir que o "usuário é quem ele diz que ele é". Para isso podem ser exigidas do usuário outras informações, tais como senhas, impressão digital etc.

A especificação CORBA(CORBA,1999) prevê um objeto autenticador, denominado *PrincipalAuthenticator*. Um vez autenticado, o sistema gera para o usuário suas credenciais de acordo com o grupo que ele pertence. A Figura 10.1 ilustra o acesso à arquitetura SICSD do usuário "José", pertencente ao grupo dos controladores de satélite (CONSAT). Uma vez tendo sido aprovado o acesso do usuário pelo *PrincipalAuthenticator*, o sistema disponibiliza as funções previamente definidas para o grupo ao qual pertence o usuário.

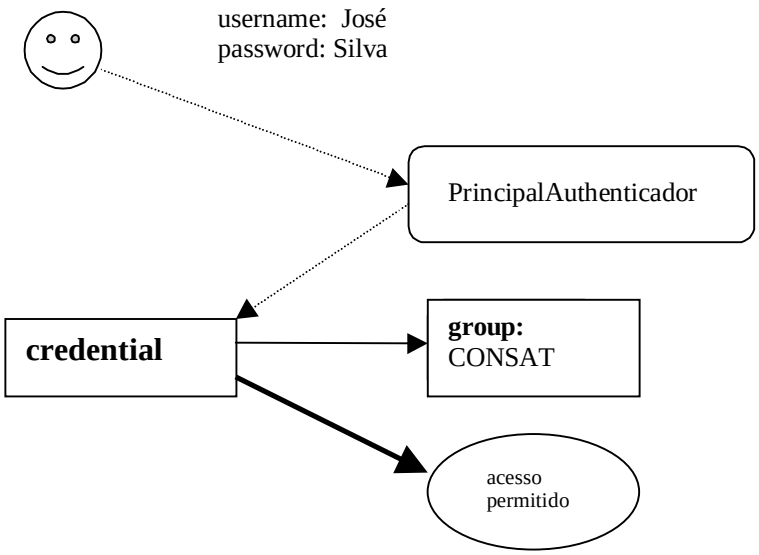


Fig. 10.1 - Autenticação do usuário.
FONTE: adaptada de Blakley (1999, p.42).

10.2 - POLÍTICA DE PROTEÇÃO DOS OBJETOS

Em um sistema distribuído, os objetos podem receber solicitações de serviços, tanto dos usuários do sistema como também de outros objetos. Considerando este fato, sistemas distribuídos, baseados em objetos, devem implementar mecanismos que restringem o acesso aos objetos, somente para usuários ou outros objetos autorizados.

O grande problema para definir políticas de proteção para objetos caracteriza-se pelo número de regras que pode ser gerado para proteger todos os objetos. Como exemplo, toma-se um sistema com 1000 objetos e 100 usuários. Considerando que cada objeto tem em média dez métodos, pode-se gerar as seguintes regras: A Figura 10.2 ilustra algumas possibilidades:

Usuário1 pode executar metodo1 do objeto1

Usuário1 não pode executar metodo2 do objeto1

Usuário1 pode executar metodo3 do objeto1

...

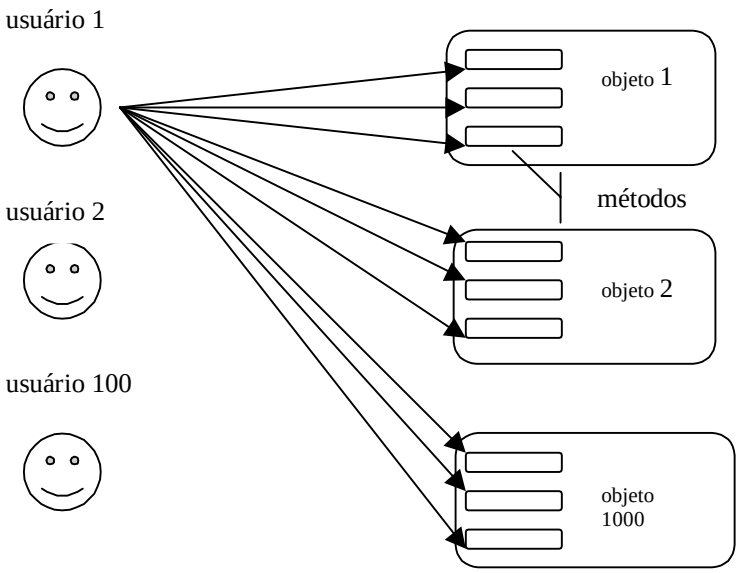


Fig. 10.2- Regras de proteção para objetos.
FONTE: adaptada de Blakley (1999 p. 46).

Com base no exemplo da Figura 10.2, o administrador de segurança do sistema deve gerar em média 10.000 regras somente para o usuário 1 (1000 objetos X 10 métodos). Para todo o sistema, seria necessário um total de 1.000.000 de regras (10.000 X 100), o que tornaria impossível implementar políticas de segurança nesse sistema.

Tentando buscar solução para o problema acima, propõe (Blakley,1999):

a) agrupar os usuários e

b) definir domínios para objetos.

10.2.1 Agrupar os Usuários

O objetivo dessa estratégia consiste em agrupar os usuários e estabelecer a mesma política de segurança para o grupo. Dentro da arquitetura SICSD, todos os usuários pertencentes ao grupo CONSAT estão regidos pelas mesmas regras de segurança. Partindo deste pressuposto, o autenticador de segurança (Figura 10.1) identifica o usuário e o grupo que o usuário pertence. De acordo com o grupo, identificam-se as regras de segurança que esse grupo está subordinado.

10.2.2 Definir Domínios para os Objetos

Um domínio de objeto compreende um conjunto de objetos regidos por uma mesma política de segurança. Ou seja, cabe ao administrador do sistema definir quais objetos devem ser agrupados dentro de um mesmo domínio. A Figura 10.3 elucida possíveis domínios existentes dentro da arquitetura SICSD; em seguida, comenta-se a divisão desses domínios.

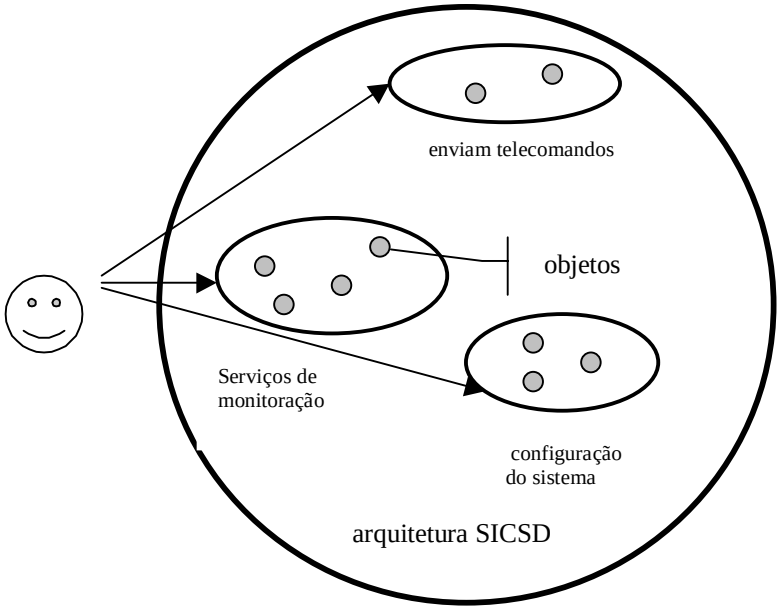


Fig. 10.3 - Agrupamento dos objetos em domínios.
FONTE: adaptada do CORBA (1999, p. 52).

As características de funcionalidades e serviços prestados por cada objeto devem ser levadas em consideração ao se formar os domínios. Vários critérios podem ser utilizados para agrupar os objetos em domínios. A seguir mencionam-se alguns deles:

- Objetos que implementam serviços de monitoração.
- Objetos que modificam a configuração do sistema.
- Objetos que enviam telecomandos para o satélite.
- Objetos fazem a interface com o banco de dados.

- Objetos que cadastram os usuários do sistema etc.

Dentre os possíveis domínios existentes na arquitetura SICSD, a Figura 10.3 ilustra três.

Os objetos telemetria e equipamento podem fazer parte do domínio dos objetos que implementam *serviços de monitoração,* já que ambos são constituídos de métodos que implementam serviços de monitoração do sistema. Por exemplo, o objeto telemetria contém métodos que simplesmente visualizam as telemetrias recebidas do satélite. Esse método é passivo e qualquer usuário pode acessar. Já o objeto equipamento contém métodos que monitoram os estados de um equipamento. Portanto, objetos pertencentes a esse domínio, podem ser acessados pela maioria dos usuários (CONSAT, DIRMIS, ENSAT etc.).

Por medidas de segurança, somente um grupo de usuários pode pertencer ao domínio de objetos *que enviam telecomandos* para o satélite, no caso, o grupo dos CONSAT.

Os objetos ranging e estação podem pertencer ao domínio de objetos *que modificam a configuração do sistema.* Uma das funções do objeto ranging consiste em calibrar o equipamento de ranging da estação, já o objeto estação, disponibiliza métodos para configurar os equipamentos. Os grupos de usuários CONSAT e ENSAT podem utilizar os serviços prestados por estes objetos.

Agrupando os usuários e definindo domínios para os objetos, o número de regras necessárias para definir políticas de segurança cai substancialmente, se comparado com o número de regras necessários para implementar mecanismos de segurança na arquitetura elucidada na Figura 10.2. A Figura 10.4 ilustra o agrupamento de usuários e os objetos da arquitetura SICSD. Várias regras para implementar segurança poderiam ser criadas, dentre elas, pode-se destacar:

ENSAT pode acessar o domínio "Serviços de monitoração".

ENSAT pode acessar o domínio "Configuração do sistema".

CONSAT pode acessar o domínio "Serviços de monitoração".

CONSAT pode acessar o domínio "Configuração do sistema".

CONSAT pode acessar o domínio "Enviar telecomandos".

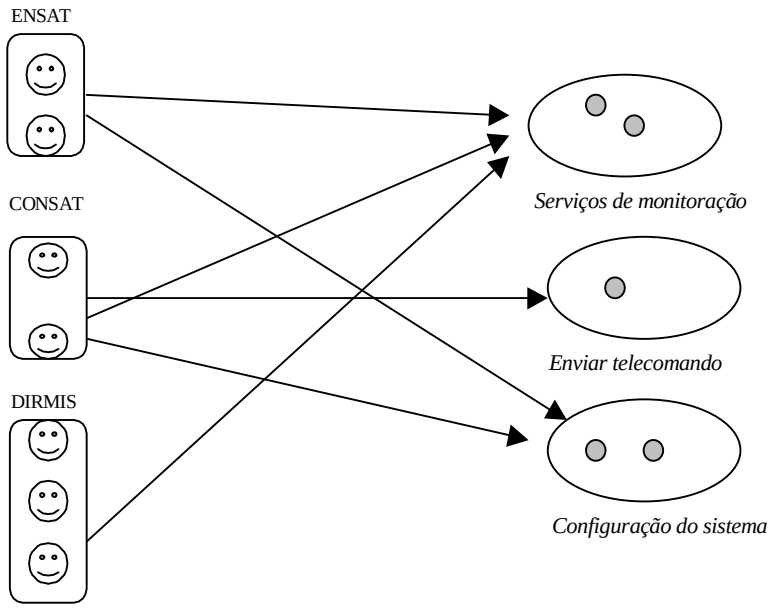


Fig. 10.4 – Objetos e usuários agrupados na arquitetura SICSD.

10.3 - SEGURANÇA NO TRÁFEGO DE MENSAGENS

O item anterior enfocou políticas de segurança aplicada aos objetos distribuídos na arquitetura SICSD. Não basta, entretanto, proteger os objetos do acesso de usuários não autorizados. A interação entre os usuários e os objetos acontece por troca de mensagens entre os mesmos. Uma mensagem está presente tanto na solicitação de serviço de um usuário para um objeto, quanto na resposta obtida pelo mesmo. As políticas de proteção de mensagens surgem para evitar que, usuários não autorizados, leiam as mensagens trafegadas entre os usuários e objetos, em um sistema distribuído.

A criptografia se apresenta como um dos recursos mais utilizados para proteger a troca de mensagens em uma rede. No contexto da arquitetura SICSD, não necessariamente todas as mensagens necessitam ser criptografadas. Dentre as principais mensagens, que trafegam entre o usuário e os objetos da aplicação pode- se citar:

a) *Solicitação de conexão com equipamento telemetria*: sugere-se que esta mensagem seja criptografada porque ela aloca o equipamento e não deixa nenhum outro usuário acessar.

b) *Telemetrias recebidas do satélite*: esta mensagem é a resposta da mensagem anterior, ou seja, primeiro estabelece-se a conexão com equipamento, para depois receber as telemetrias do satélite. Esta mensagem não deve ser criptografada por dois motivos: primeiro, pela taxa de recepção (duas mensagens por segundo). O tempo utilizado no processo de criptografar e descriptografar pode comprometer a operação. Segundo, porque essas informações ainda têm que passar por um processamento para, finalmente, refletir o estado interno do satélite.

c) *Solicitação de conexão com equipamento telecomando*: Sugere-se que esta mensagem seja criptografada porque ela aloca o equipamento e não deixa nenhum outro usuário acessar.

d) *Envio de telecomando*: Esta mensagem deve ser criptografada, já que o telecomando pode mudar o estado interno de um satélite (ligar/desligar determinado circuito interno).

10.4 – AUDITORIA

O serviço de auditoria consiste em inspecionar o sistema para averiguar se ele está sendo utilizado de forma segura, ou se está sendo invadido por usuários não autorizados. Os eventos que ocorrem no sistema podem ser armazenados em arquivos de "logs", impressos ou visualizados na tela de um computador. Para evitar a criação de imensos arquivos de "logs", bem como sobrecarregar o sistema, processando mensagens de "logs", o serviço de auditoria não monitora todos os eventos que ocorrem no sistema. Várias tentativas frustradas de entrar no sistema, bem como a exclusão de um arquivo, caracterizam-se eventos que devem ser monitorados.

Na especificação CORBA (CORBA,1999), as políticas de auditoria são tratadas pelo objeto *AuditPolicy objects.* Cada *AuditPolicy objects* tem um conjunto de regras para decidir se um determinado evento deve ser armazenado ou não. Cada uma dessas regras descreve quais eventos devem ser monitorados e armazenados. Dessa forma, o serviço de auditoria desempenha o seu papel na arquitetura SICSD, consultando o *AuditPolicy objects* a cada novo evento ocorrido no sistema; se não existe nenhuma restrição, esse

evento não é armazenado. Citam-se alguns dos eventos monitorados pelo serviço de segurança CORBA :

- "login" de usuários;
- criação/destruição de objetos;
- troca de mensagens;
- invocação de métodos de objetos.

A cada requisição de serviço gerada pelo sistema, o ORB aciona o *AuditDecision Object* para verificar se esse evento deve ser armazenado. O *AuditDecision objects* consulta o *AuditPolicy objects*, o qual contém a relação dos eventos que devem ser monitorados. Em caso afirmativo, armazena-se este evento. A Figura 10.5 elucida esta idéia.

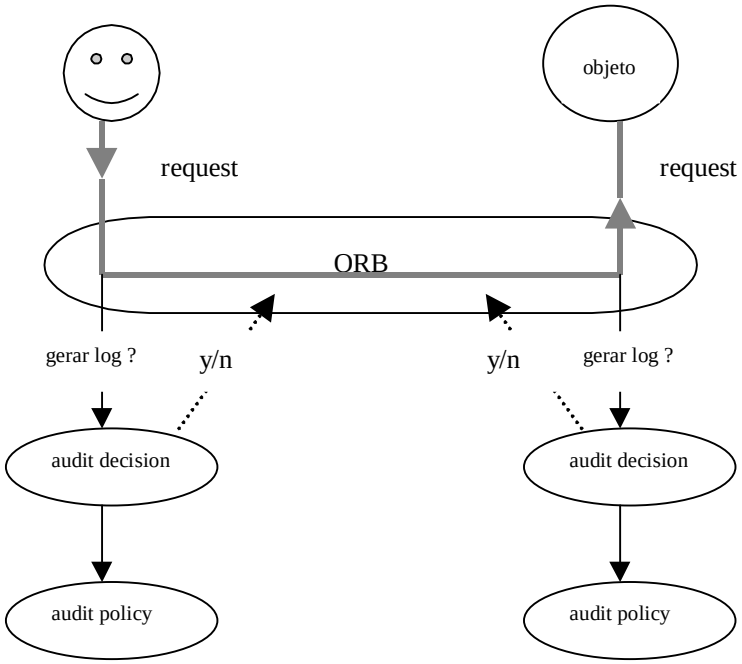


Fig. 10.5 - Política de auditoria de eventos.

10.5 - MECANISMOS DE PROTEÇÃO PARA MIGRAÇÃO DE OBJETOS OU AGENTES NA ARQUITETURA SICSD

O tratamento recebido no processo de migração pelos objetos dinâmicos da aplicação para Controle de Satélites e os agentes móveis da arquitetura SICSD apresentam características similares. A diferença entre um objeto móvel da aplicação e um agente reside no fato de que o primeiro só migra com a autorização do serviço de balanceamento, enquanto que o segundo é autônomo.

Tanto os agentes como os objetos da aplicação devem conter uma assinatura digital capaz de garantir a sua integridade em qualquer ponto da rede. Esta assinatura é codificada no nó de origem do agente ou objeto móvel e decodificada pelo nó destino. Entretanto, outros mecanismos de segurança devem ser agregados à arquitetura SICSD para garantir a segurança, tanto dos agentes/objeto móvel em relação ao sistema, quanto do sistema em relação aos agentes/objeto móvel.

10.5.1 Requisitos de Segurança

- **Confidencialidade:** Qualquer dado privado, armazenado em uma plataforma ou transportado por um agente, deve permanecer como informação confidencial. As plataformas devem possuir mecanismos que garantam confidencialidade às comunicações internas e interplataformas. Alguma entidade "espiã" pode recolher informações das atividades de um agente, não apenas do conteúdo das mensagens trocadas, mas também do fluxo de mensagens de um agente para outros agentes. A localização dos agentes móveis também pode ser guardada confidencialmente. Comunicando-se através de um “proxy”, o agente pode ocultar sua presença numa plataforma particular. Os agentes podem decidir se a sua localização estará publicamente disponível nos diretórios da plataforma, fazendo com que as plataformas, por consequência, reforcem as políticas de segurança sobre os agentes que optem pelo “anonimato”.
- **Integridade:** A plataforma deve proteger os agentes/objeto contra modificações não-autorizadas de seus dados, código e estado, e garantir que apenas agentes ou processos autorizados realizem alteração nos dados compartilhados. A operação segura dos sistemas de agentes/objeto móveis depende, também, da integridade das

próprias plataformas locais e remotas. Um “host” mal intencionado pode facilmente comprometer a integridade de um agente/objeto móvel, enquanto visita uma plataforma remota. Isto porque, o agente/objeto móvel normalmente é migrado para uma porta TCP/IP, conhecida pelo “host” remoto. O bloqueio ou a desabilitação da porta pode comprometer a migração de um agente/objeto móvel. A tendência em direção à proliferação de sistemas operacionais e plataformas, com código livre (open source) pode facilitar a atuação de administradores ou organizações inescrupulosas, que modificam tanto a plataforma como o sistema de suporte. Os ataques orientados a metas, contra as comunicações dos agentes, intentam comprometer a integridade das mensagens, alterando o seu conteúdo, substituindo por completo a mensagem, reutilizando uma mensagem antiga, excluindo a mensagem ou alterando a sua origem/destino. As plataformas dependem de protocolos de baixo nível para garantir a integridade das comunicações (TCP/IP) (Caglayan, Harrision, 1997).

- **Responsabilidade-Prestação de Contas:** Cada processo, usuário ou agente/objeto, numa dada plataforma, deve ser responsabilizado (prestar contas) por suas ações. Logo, devem ser unicamente identificados e autenticados. A “prestação de contas” requer a manutenção de um “audit log” dos eventos ocorridos, relevantes à segurança e à listagem de cada evento, com o respectivo agente responsável. Mecanismos de autenticação fornecem “prestação de contas” para ações dos usuários. Os agentes/objeto devem estar aptos para autenticar suas identidades para as plataformas e para outros agentes/objeto. A “prestação de contas” também é essencial para reforçar a confiança entre os agentes/objeto e as plataformas, visto que um agente/objeto autenticado pode obedecer as políticas de segurança da plataforma e ainda exibir comportamento mal intencionado, disseminando informações falsas.

- **Disponibilidade**: A plataforma de agente deve possuir capacidades para: a) garantir a disponibilidade de dados e serviços para os agentes locais e remotos, a provisão de concorrência controlada, suporte a acessos simultâneos, o gerenciamento de “deadlock”; b) executar a detecção e recuperação de falhas de software/hardware; c)

manipular as requisições de inúmeros “visitantes” e agentes remotos ou ocorrências não intencionadas de recusa de serviço (Jansen, Karygiannis, 1999).

10.5.2 Políticas de Segurança

- **Protegendo a Plataforma de Agentes:**

Uma das principais exigências, referente à implementação de sistemas de agentes, é a garantia de que um agente não possa interferir nos outros agentes e na própria plataforma base. Uma abordagem comum satisfatória se resume no estabelecimento de domínios separados e isolados para o agente e para a plataforma, controlando todos os acessos entre domínios. Tradicionalmente, tal conceito é referido como “reference monitor” (CRU, 1983). Implementações deste conceito têm surgido desde o início dos anos 80, empregando diversas técnicas de segurança convencionais, aplicáveis ao ambiente de agentes, tais como:

1) Mecanismos para isolar processos (entre si e isolamento do processo de controle).
2) Mecanismos para controlar o acesso aos recursos computacionais.
3) Métodos de criptografia para cifrar trocas de informação.
4) Métodos de criptografia para identificar e autenticar usuários, agentes e plataformas.
5) Mecanismos de auditoria aos eventos relevantes à segurança que ocorrem na plataforma.

- **Protegendo os Agentes:**

Enquanto as medidas tomadas para a segurança da plataforma evoluíram diretamente dos mecanismos empregados para os próprios "hosts", enfatizando, assim, medidas ativas de prevenção, as medidas para a proteção dos agentes tendem a ser caracterizadas como medidas de detecção, visto que os agentes estão completamente suscetíveis às

plataformas. Ou seja, não podem se prevenir contra ocorrências de comportamentos mal intencionados; estão aptos, apenas, para detectá-las (Jansen, Karygiannis, 1999). Este problema tem como origem a inabilidade para estender o ambiente seguro da "plataforma-home" para as outras plataformas remotas. Algumas técnicas de âmbito geral, para a proteção dos agentes, incluem:

1) Privacidade e integridade das informações obtidas: Mecanismos de segurança devem ser implantados, como por exemplo a criptografia, para garantir que as informações coletadas ou transportadas pelos agentes móveis não sofram nenhum ataque.

2) "Trace" de todos os métodos executados em todos os nós: Pode-se rastrear todos os nós percorridos pelo agente, bem como os métodos executados em cada um deles. Essa auditoria garante que somente os métodos previamente autorizados estão sendo executados nos respectivos nós.

10.6 - A IMPORTÂNCIA DO SERVIÇO DE SEGURANÇA

As aplicações distribuídas requerem políticas mais rígidas de segurança do que as aplicações centralizadas, já que por definição uma aplicação distribuída apresenta um maior número de pontos vulneráveis a ataques de usuários mal intencionados do que uma aplicação centralizada. O rigor na implementação dessas políticas assume dimensões ainda maiores, em se tratando de uma aplicação voltada para o controle de satélites.

Este capítulo fez um esboço dos mecanismos de segurança que devem ser implementados na arquitetura SICSD, para garantir a privacidade das informações trocadas entre os objetos, a autenticidade dos usuários do sistema, a integridade dos agentes e objetos da aplicação para controle de satélites no processo de migração entre os nós.

O agrupamento de usuários que acessam a arquitetura SICSD pode ser a primeira atividade em prol da garantia de acesso ao sistema de controle de satélites. Sugere-se, então, a criação dos grupos: CONSAT, ENSAT, DIRMIS etc. Esta estratégia viabiliza a criação de políticas de segurança, agora adotadas para um grupo e não mais para um

único usuário. O processo de “login” consiste em identificar e autenticar o usuário. O próximo passo baseia-se em identificar o grupo que esse usuário pertence e atribuir os privilégios pertinentes ao grupo.

A criação dos domínios de objetos, no contexto da arquitetura SICSD, apresenta-se como uma ferramenta importante para implementar políticas de segurança específicas para cada domínio. A identificação dos objetos pertencentes a um determinado domínio consiste em agrupar os objetos que prestam serviços similares à comunidade de usuários da arquitetura SICSD. Por exemplo, objetos que prestam serviços de monitoração devem estar separados dos objetos que prestam serviços de configuração.

Uma vez delimitados os domínios do sistema, deve-se definir quais os grupos de usuários podem acessar esses domínios. Ao término da implementação desses dois mecanismos, espera-se ter protegido o acesso de usuários não autorizados aos objetos da arquitetura SICSD.

O próximo passo consiste em implementar políticas que garantam o sigilo da comunicação entre os grupos de usuários e os domínios de objetos. Nesse caso a criptografia apresenta como um recurso capaz de assegurar o acesso aos dados somente de usuários previamente autorizados.

O ambiente flexível e dinâmico, anunciado na arquitetura SICSD, está fundamentado na possibilidade de os objetos da aplicação de controle de satélites migrarem ou serem replicados de um nó origem para um nó destino. Essas características propostas no ambiente são suportadas pela atuação dos agentes que mantêm atualizada a base de configuração do sistema. Tanto os agentes como os objetos da aplicação devem conter uma assinatura digital para garantir sua integridade no processo de migração em qualquer ponto da rede.

Caso as políticas de segurança já sugeridas (autenticação, agrupamento do usuários, criação de domínios de objetos e criptografia) não se mostrarem suficientes para garantir a segurança da arquitetura SICSD, deve-se explorar os recursos de auditoria para capturar prováveis infratores.

Definiu-se um panorama das principais políticas de segurança que devem ser aplicadas à arquitetura SICSD; entretanto, não se descarta a possibilidade de incluir novas políticas, caso as já mencionadas não sejam suficientes para sanar o problema.

CAPÍTULO 11

IMPLEMENTAÇÃO

Nada se torna real até ser experimentado – mesmo um provérbio não significa nada pra você até sua vida ilustrá-lo. (John Keats)

Este capítulo assume como objetivo apresentar o ambiente de teste utilizado no desenvolvimento do protótipo para a arquitetura proposta. Os principais códigos utilizados nesta implementação encontra-se no apêndice A. Considerando a complexidade da arquitetura proposta, foram desenvolvidos protótipos das seguintes implementações:

- principais objetos de um sistema para controle de satélite, tais como telemetria, ranging, telecomando, estação etc.
- um simulador, com a capacidade de simular um satélite, que contenha cinco telemetrias.
- o serviço dos agentes.
- o serviço de balanceamento de carga.
- o serviço de persistência.

11.1 – O AMBIENTE DE TESTES

O ambiente foi composto por três máquinas pertencentes à rede local do Departamento de Controle de Satélites, as quais apresentavam as seguintes características:

Patras: Pentium III , 64 M de Ram;

Andros: Pentium II, 64 M de Ram; e

Pelion: Pentium II, 64 M de Ram.

11.1.1 Softwares Utilizados

O protótipo foi desenvolvido sobre a plataforma java(Java Development Kit – JDK, versão 1.1.7). Além disto, utilizou-se 2 softwares aplicativos:

- Visibroker, versão 3.2 e
- Voyager, versão 2.0.2

a) Visibroker

A empresa Visigenic implementou as especificações CORBA no produto denominado Visibroker. Esse "middleware" permite o desenvolvimento e gerenciamento de objetos distribuídos. Os objetos construídos com Visibroker podem ser acessados pela Internet e Intranet, utilizando o protocolo internamente agregado ao produto (IIOP- Inter ORB Protocol). O Visibroker implementa os principais requisitos da especificação CORBA, dentre eles pode-se mencionar(Visibroker,1998):

- Repositório de Interfaces;
- Interface de invocação dinâmica;
- Interface de invocação estática;
- Serviço de nomes;
- Tratamento de exceções etc.

b) Voyager

A companhia ObjectSpace, uma líder na tecnologia de software avançada, anunciou seu produto Voyager (04/1997), como uma das primeiras plataformas para computação distribuída em Java, aperfeiçoada pela tecnologia de agentes. Voyager foi a primeira plataforma comercial que permitia o desenvolvimento de sistemas, utilizando mecanismos para comunicação remota além da tecnologia de agentes(Voyager,1998).

11.1.2 Contabilização dos Códigos Desenvolvidos

A maioria dos códigos desenvolvidos para a implementação de um protótipo da arquitetura SICSD foram desenvolvidos na linguagem java. A Tabela 11.1 elucida um resumo geral de todo o desenvolvimento.

TABELA 11.1 - RESUMO GERAL DO CÓDIGO DESENVOLVIDO

Atividade	Número de classes desenvolvidas	Linhas de código
Carga do sistema	1	100
Simulador da arquitetura SICSD	-	3000
Simulador de Satélites	3	260
Objetos da aplicação para Controle de Satélites	10	850
Serviço dos Agentes	10	1000
Serviço de Balanceamento	6	1400
Serviço de Persistência	10	650
TOTAL	**40**	**9260**

11.1.3 – Funcionamento da Arquitetura SICSD

A Figura 11.1 ilustra o funcionamento do protótipo desenvolvido para a arquitetura SICSD. De acordo com essa Figura pode-se observar:

- Os serviços disponíveis na arquitetura SICSD (balanceamento, persistência, agentes e segurança), bem como a base de configuração e de dados, encontram-se instanciados nos nós Patras, Pelion e Andros .

- Após a carga do sistema, o serviço dos agentes, apresenta-se como o responsável pela monitoração de todos os objetos❶ da aplicação para controle de satélites. As informações monitoradas são armazenadas na Base de configuração❷. Os agentes mensageiros assumem o compromisso de disseminar❸ o conteúdo da base de configuração para os outros nós (Pelion e Andros).

- Toda alteração no conteúdo da base de configuração é analisa pelo serviço de balanceamento,❹ o qual, tem a capacidade de atuar sobre os objetos da aplicação de controle de satélites, quer seja executando a operação de migração ou cópia❺. Sem nenhum demérito, outro serviço prestado pelo serviço de balanceamento, consiste em intermediar a utilização do sistema pelo usuário❻. Ou seja, qualquer requisição solicitada pelo usuário é analisada pelo serviço de balanceamento, que assume o compromisso de apontar o objeto melhor adaptado para atender determinada solicitação.

- Compete ao serviço de persistência, armazenar/recuperar os objetos persistentes da aplicação para controle de satélites❼. Esse serviço recebe solicitações dos objetos

persistentes para armazená-los, e localiza o banco de dados que deve ser armazenado esse objeto. Ele implementa a transparência da localização física de armazenamento de um objeto, ou seja, como a arquitetura proposta é dinâmica, a localização física de um objeto pode ser diferente da localização do banco de dados onde ele deve ser armazenado❽.

As responsabilidades do serviço de segurança não estão limitadas apenas em garantir a utilização do sistema por usuários previamente autorizados. Outras funções são atribuídas a esse serviço, tais como implementar políticas que protejam as mensagens trafegadas na rede, auditoria etc.

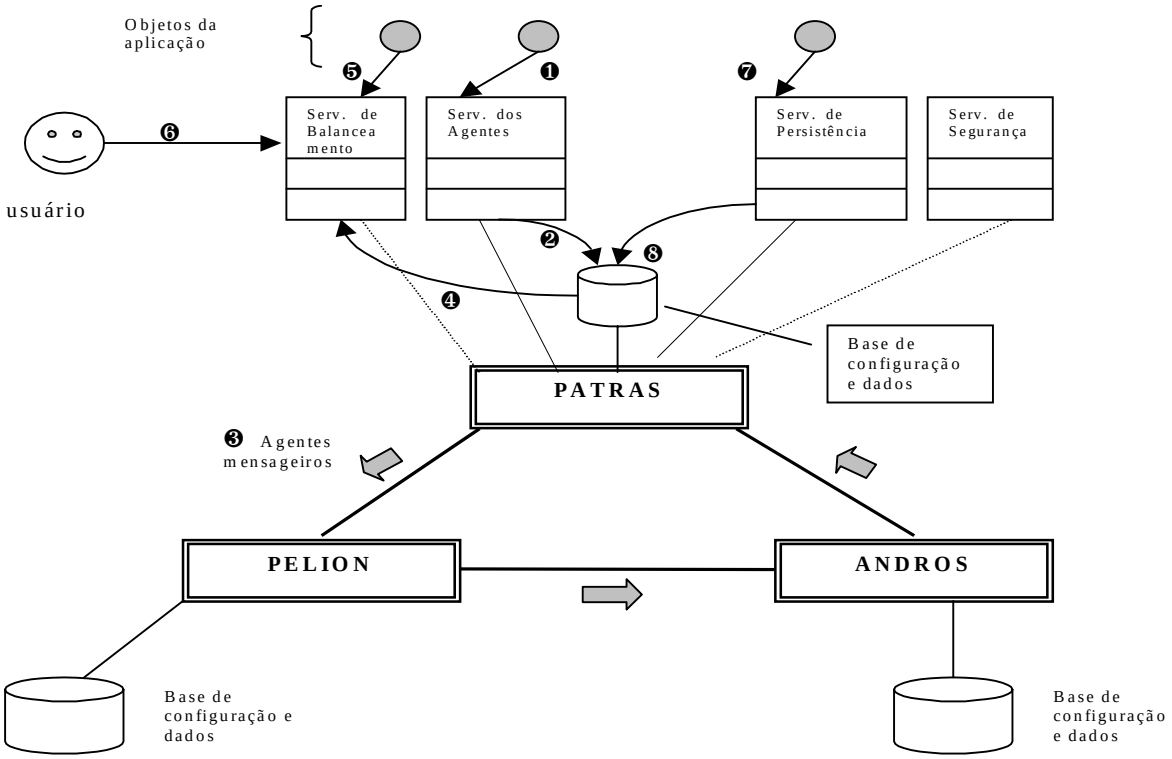


Fig. 11.1 – Funcionamento da arquitetura SICSD.

11.2 – A CARGA DA ARQUITETURA SICSD

A responsabilidade pela carga da arquitetura SICSD concentra-se na rotina “start”, presente em cada nó da rede. Ela se apresenta como a responsável por instanciar os objetos da aplicação para controle de satélites, bem como os serviços (agentes, persistência, segurança e balanceamento).

A Figura 11.2 ilustra os objetos instanciados pela rotina de carga da arquitetura SICSD. No item A-1 do Apêndice A encontra-se o código da rotina “start”; entretanto, parte desse código é mostrado a seguir:

```
// Objeto telemetria de alcântara
tms_alc =
(Ialcantara)Factory.create(alcantara.class.getName(),"//150.163.20.7:8200/tms_alc");

// Serviço de Persistência
 persistencia =(ipersistencia)
Factory.create(persistencia.class.getName(),"//150.163.20.7:8203/persistencia");
```

OBS.: O código acima ilustra a criação do objeto tms_alc e do serviço de persistência, no nó cujo endereço IP=150.163.20.7 e as portas 8200 e 8203 respectivamente.

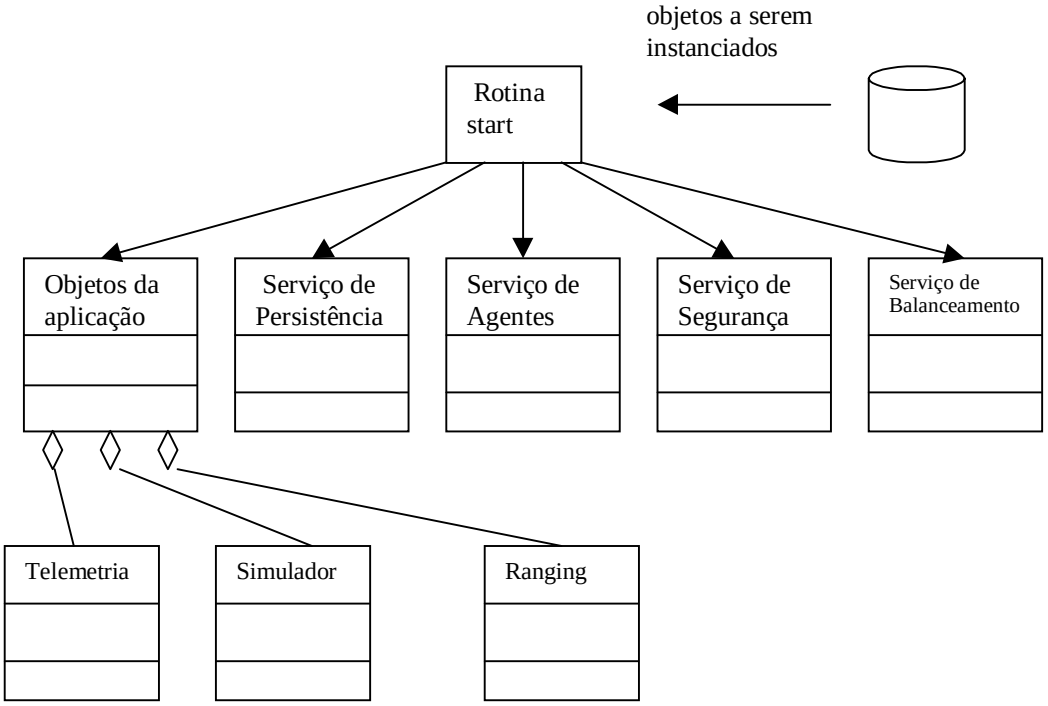


Fig. 11.2 Objetos instanciados para cada nó na rotina “start”.

11.3 – IMPLEMENTAÇÃO DAS INTERFACES ENTRE OS OBJETOS DA APLICAÇÃO, SIMULADOR E OS SERVIÇOS DISPONÍVEIS

O protótipo implementado utilizou produtos que seguem a especificação CORBA. Portanto, cada objeto ou serviço implementado, disponibiliza uma interface de acesso.

Seguindo o padrão UML (Unified Modeling Language), pode-se representar a interação dos principais objetos com suas respectivas interfaces da seguinte forma (Figura 11.3).

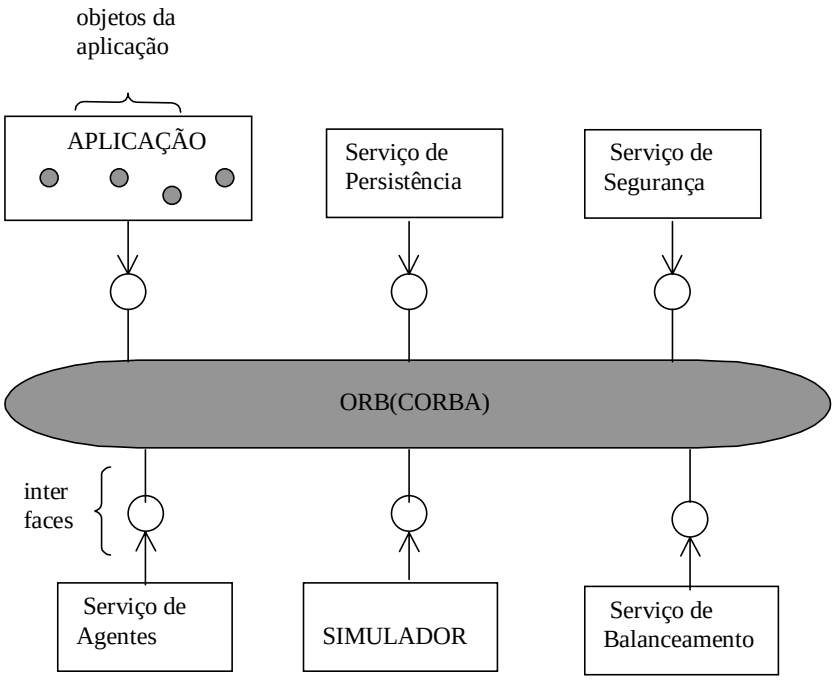
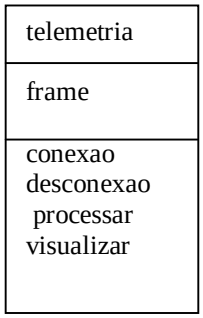


Fig.11.3- Uma visão da arquitetura proposta com suas principais interfaces.

11.3.1 – IDL dos objetos da aplicação

Selecionaram-se dois objetos da aplicação (telemetria e telecomando) para exemplificar a utilização das IDLs.

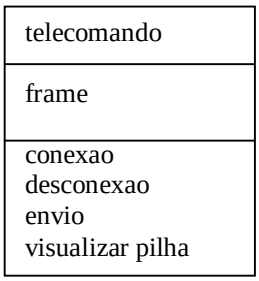
Objeto telemetria **IDL correspondente**




```
interface Itelemetria {
  String get_frame();
  void conexao();
  void desconexao();
   void processar();
 void visualizar();

}
```

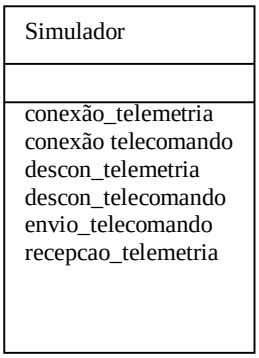
Objeto telecomando **IDL correspondente**




```
interface itelecomando
  {
 String get_frame();
   void conexao();
  void desconexao();
   void envio (in string frame);
   void visualizar pilha();
    };
```

11.3.2 – IDL do Simulador

Objeto Simulador **IDL correspondente**




```
interface isimulador
  {
   void conexao_telemetria();
   void conexao_ telecomando();
   void  descon_telemetria();
   void  descon_telecomando();
   void  envio_telecomando(in string
frame);
   String recepcao_telemetria();
   void envio_telecomando (in string
frame);
    };
```

11.4 - SIMULADOR DE SATÉLITES

O simulador desenvolvido nesse trabalho, incorpora modestas características, se comparado como um real simulador de satélites. O satélite, por ele simulado, compreende cinco telemetrias, sendo duas analógicas e três digitais. As telemetrias analógicas expressam os valores dos parâmetros internos de um satélite, que podem sofrer variações entre um valor mínimo e um valor máximo. Por exemplo, os valores de 5 volts à 12 volts, podem ser atribuídos a telemetria analógica que contém a voltagem de uma bateria. Já as telemetrias digitais, exprimem as posições das chaves existentes em um satélite. Normalmente o valor zero é atribuído a telemetria digital responsável por monitorar uma determinada chave, se a mesma apresentar desligada, e o valor um, caso contrário.

O Simulador agrega basicamente duas funções. A primeira, consiste em gerar de forma aleatória, valores para as telemetrias analógicas simuladas. A Segunda, em refletir nas telemetrias digitais, o resultado da execução dos telecomandos enviados. Ou seja, se for enviado um telecomando para ligar uma determinada chave, a telemetria digital, responsável pela monitoração do estado desta chave, deve alterar de “desligado” para “ligado”.

O Simulador foi instanciado em uma única máquina, e disponibiliza uma interface para acessar as telemetrias geradas e para enviar telecomandos. A Figura 11.4 ilustra o esquema elétrico do satélite simulado e no apêndice A, item (A-2), encontra-se o código do Simulador para a estação de alcântara.

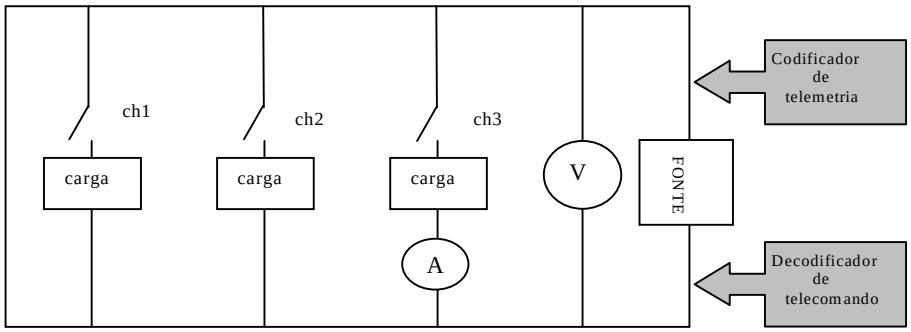


Fig. 11.4 – Esquema elétrico do satélite simulado.

De acordo com a Figura 11.4:

- O *codificador de telemetria* apresenta-se como o responsável pela coleta de todas as telemetrias existentes no satélite.
- Atribui-se ao decodificador de telecomando, a responsabilidade de processar os telecomandos recebidos pelo satélite.
- Para cada chave do circuito elétrico(ch1, ch2 e ch3), existe uma telemetria digital, que monitora o seu estado, podendo ser ligado(valor = 1) ,ou desligado (valor=0).
- Existe uma telemetria analógica que monitora o valor coletado pelo amperímetro (A), e outra que monitora a voltagem da fonte (V).

A Tabela 11.2 ilustra o conjunto de chaves existentes no satélite, as telemetrias digitais que monitoram o estado de cada uma delas, bem como os telecomandos responsáveis por ligá-las/desligá-las.

TABELA 11.2 – TELEMETRIA E TELECOMANDOS EXISTENTES NO SATÉLITE SIMULADO

Chave	**Telemetria-valor**	**Telecomando atuante**
Ch1	Tlm01 = 1	Tc101 – liga
	Tlm01 = 0	Tc102 – desliga
Ch2	Tlm02 = 1	Tc103 – liga
	Tlm02 = 0	Tc104-desliga
Ch3	Tlm03 = 1	Tc105-liga
	Tlm03 = 0	Tc106-desliga

11.5 - OBJETOS DA APLICAÇÃO

Os objetos telemetria e telecomando foram os principais objetos implementados da aplicação para controle de satélites. No apêndice A, item A-3, encontra-se o código do objeto telemetria para a estação de alcântara. Após o processo de carga dos objetos da aplicação, o serviço de balanceamento assume o controle sobre o ciclo de vida desses objetos. Ou seja, se incumbe pela migração, replicação, ou até mesmo exclusão desses objetos, de acordo com as necessidades operacionais, as quais estão envolvidas no controle de um determinado satélite.

11.6 - SERVIÇO DOS AGENTES

Apresenta-se nesse item, as principais classes do serviço dos agentes.

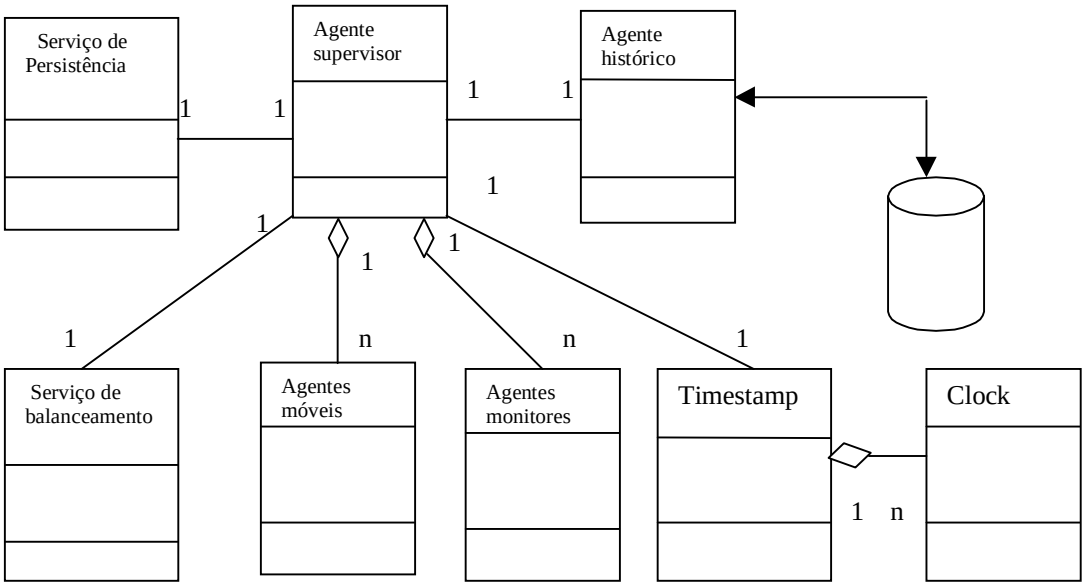


Fig. 11.5 – Diagrama de classes do Serviço de Agentes.

11.6.1 Agente Supervisor

Como já foi mencionado, o agente supervisor encontra-se fixo em cada nó da rede. Ele atua como servidor de inicialização, portanto, são introduzidas as classes **Comp** e **CompTable,** que complementam o cenário de atuação do Agente Supervisor. Como o ambiente de teste utilizou três nós (Patras, Pelion e Andros), no apêndice A, item A-4, ilustra a criação do agente supervisor no nó Patras (Silva,2000).

11.6.2 Agente Móvel

Os agentes móveis são instanciados pelo Agente Supervisor de cada nó. Realizam tarefas pré-determinadas, movendo-se por todos os nós da rede configurados para a implantação da Arquitetura Proposta, retornando à sua origem após completar suas missões. O item A-4, do apêndice A, elucida o funcionamento de um agente móvel.

11.6.3 Agente Monitor

Os agentes monitores são fixos e instanciados pelo Agente Supervisor de cada nó. Sua tarefa consiste em monitorar os objetos da aplicação para controle de satélites. Para cada objeto da aplicação, instancia-se também um agente monitor. O Item A-4 do apêndice A, ilustra o funcionamento de um agente monitor.

11.6.4 Agente histórico

Seguindo requisitos e funções primordiais, os Agentes Supervisores devem manter-se atualizados quanto a situação, utilização e tipos de operações requeridas dos objetos no âmbito local e global do sistema. Estas informações são mantidas na memória na base de configuração e o agente Histórico provê uma estrutura para um armazenamento persistente desta base.

11.7 - SERVIÇO DE BALANCEAMENTO

Esse item mostra e comenta as principais classes do serviço de balanceamento, bem como a implementação de seus principais métodos (Figura11.6).

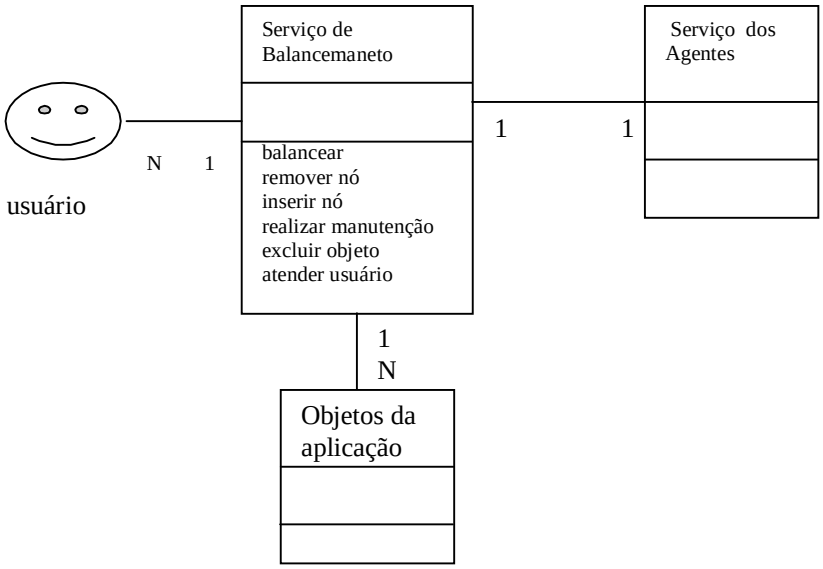


Fig. 11.6 Diagrama de classes do serviço de balanceamento.

11.7.1- Serviço dos Agentes

O serviço de balanceamento utiliza as informações (base de configuração), gerenciadas pelo serviço dos agentes. As características de flexibilidade e dinamismo incorporadas à arquitetura proposta são consequências das ações geridas pelo serviço de balanceamento, após a análise das informações contidas na base de configuração. O resultado destas ações podem ser uma migração ou cópia de um objeto. Os principais serviços prestados por essa interface se resumem em: apresentar o nó da rede que exprime a maior disponibidade de cpu (busca_maior_cpu()), a disponibidade média de CPU de todos os nós (busca_media_cpu()), o status de um nó (busca_status_no(String n)) etc.

```
public interface Agente
  String busca_no_livre();
  tabela_no busca_maior_cpu();
  tabela_no busca_menor_cpu();
  float busca_media_cpu();
  String busca_endereco(String aux);
  float busca_tab_no(String n, float tc);
  String busca_status_no(String n);
  tabela_con busca_tab_con_patras();
  tabela_con busca_tab_con_pelion();
  tabela_con busca_tab_con_andros();
   String porta_livre_patras();
```

```
  String porta_livre_pelion();
  String porta_livre_andros();
} // fim da interface
```

11.7.2 A classe Serviço de Balanceamento

Aciona-se os serviços dessa classe a cada alteração no conteúdo da base de configuração. Esses serviços podem migrar ou copiar os objetos da aplicação. O serviço de balanceamento foi dividido em quatro casos de uso, para atender as necessidades da arquitetura SICSD. A definição formal de um caso de uso, segundo Jacobson (Eriksson, Penker, 1998), consiste em:

"Um conjunto de sequências de ações que um sistema desempenha para produzir um resultado observável de valor a um ator específico".

De um ponto de vista pragmático, casos de uso podem ser empregados para a especificação de necessidades e funcionalidades oferecidas de uma classe. Considerando esse conceito, concebeu-se os seguintes casos de uso:

a) Balancear carga: Sem nenhum demérito aos outros casos de uso, esse apresenta-se como o de maior importância. A fundamentação desse caso de uso concentra-se na necessidade de otimizar os recursos computacionais existentes, para uma aplicação de software de controle de satélites, bem como distribuir o melhor o processamento, utilizando computadores que estejam ociosos.

b) Realizar manutenção no sistema: Uma aplicação de software, independente da área de atuação, está suscetível a mudanças, quer seja em função do surgimento de novas versões, ou até mesmo por um processo de manutenção no próprio hardware onde ela se encontra instalada. Partindo dessa premissa, a arquitetura SICSD deve estar apta a receber uma intervenção, por exemplo do administrador, para suspender todas as atividades que estão em processamento nesse nó.

c) Remover um nó: Prever todas as possíveis falhas que ocorram em um sistema distribuído é praticamente inviável. Entretanto, considerando que, a arquitetura SICSD está distribuída em um domínio de rede pré-definido, pode-se concentrar esforços, objetivando tornar transparente para os usuários, uma falha em um dos nós integrantes desse domínio. O primeiro nó que identificar a falha de seu nó "vizinho", apresenta-se

apto a instanciar os objetos desse nó em outros nós ociosos, já que todos os nós mantém uma base de configuração local que informa os objetos instanciados remotamente.

d) Atender solicitação de usuários: O serviço de balanceamento apresenta-se como o responsável pela interface entre a arquitetura SICSD e o usuário. Após a identificação do usuário, o serviço de balanceamento disponibiliza para o mesmo um conjunto de funções, de acordo com o seu perfil. Para cada solicitação do usuário, o serviço de balanceamento pesquisa e identifica na base de configuração, o nó que apresenta maior taxa de disponibilidade de CPU para atender a esta requisição.

A Figura 11.7 ilustra os principais casos de uso do serviço de balanceamento e a Tabela 11.3, uma descrição geral de cada um deles.

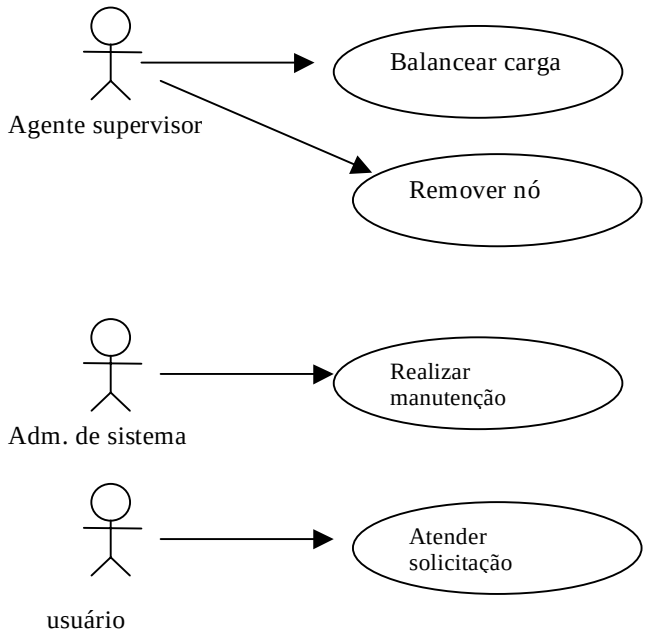


Fig. 11.7. Principais casos de uso do serviço de balanceamento.

TABELA . 11.3 - DESCRIÇÃO GERAL DOS PRINCIPAIS CASOS DE USO DO SERVIÇO DE BALANCEAMENTO

Caso de uso	Quem inicia a ação	Descrição do caso de uso
Balancear carga	Agente supervisor	O agente supervisor, ao identificar uma alteração na base de configuração do nó, aciona o serviço de balanceamento.
Realizar manutenção	Administrador de sistemas	O administrador de sistemas aciona o serviço de balanceamento, caso haja necessidade de instalar novos softwares ou mesmo alterar a configuração de hardware de um nó.
Remover nó	Agente supervisor	A falha em um nó é detectada pelo agente supervisor instanciado no nó vizinho. O próximo passo, consiste em decidir o nó que irá assumir o controle do nó que entrou em falha. A atuação do nó que assume o controle, através do serviço de balanceamento, consiste em instanciar os objetos do nó que entrou em falha, em outros nós que contenha maior taxa de disponibilidade de CPU.
Atender solicitação	Usuário	Todo processo de solicitação de serviços do usuário é analisado pelo serviço de balanceamento. Esse procura identificar o objeto que apresenta em melhores condições para atender o usuário. Normalmente o objeto mais apto, encontra-se instanciado no nó com maior disponibilidade de CPU.

A implementação dos principais métodos desse serviço encontra-se no item A-5, do apêndice A; entretanto, o código abaixo, ilustra parte da implementação do método balancear carga. Esse serviço é executado em cada nó, portanto, a primeira ação desse método, consiste em verificar a disponibilidade de CPU local❶. Se a taxa de disponibilidade de CPU está abaixo da média, a primeira tentativa de balanceamento consiste em replicar o objeto mais solicitado do nó local para o nó solicitante❷. Caso o nó solicitante também esteja com taxa de disponibilidade abaixo da média, deve-se procurar outro nó que tenha disponibilidade de CPU ❸.

```
public   void balanceamento()
 {
  ...

  ❶// buscar a média de cpu disponível do sistema
  media_cpu = agente_sup.busca_media_cpu();

  if (taxa_cpu < media_cpu)
    { // if 01

      ❷ // A primeira tentativa de balanceamento consiste em
       // migrar o objeto mais solicitado da máquina local para a
       // máquina remota
```

```
 // Obter o objeto que tem o maior número de conexões com o no origem

 tabela_con aux_con = new tabela_con();
 aux_con = agente_sup.busca_tab_con_patras( );


// replicar o objeto recuperado no no com o maior numero de conexões
 // Verificar se nó origem não esta congestionado
if (agente_sup.busca_tab_no(aux_con.origem, tc) > media_cpu)
 {  // if 03

   try
    {
     // Buscar endereco do no
     endereco= agente_sup.busca_endereco(aux_con.origem);
     System.out.println("enderco  ="+ endereco);

     if (aux_con.objeto.equals("tms_alc"))
      {    // if 04
       endereco = endereco + "tms_alc";
       System.out.println("enderco  ="+ endereco);
       tms_alc = (Ialcantara)Factory.create(alcantara.class.getName(),endereco);
      }
       ...

  } // if 03
  else
  {  // if 03
 ❸ // Procurar nó com maior disponibilidade de CPU
  no_livre = agente_sup.busca_no_livre();

   if (no_livre != null)
    {   // if 05
     // replicar o objeto recuperado no no LIVRE
     try
     {
      // Recuperacao do endereco

      if (aux_con.objeto.equals("tms_alc"))
       {
       endereco = no_livre + "tms_alc";
       tms_alc = (Ialcantara)Factory.create(alcantara.class.getName(),endereco);
       }
   }
}
```

```
  } // fechar o if 02

  } // if 01

 } // fim do método balanceamneto
```

11.7.3 Objetos da Aplicação

Os serviços de balanceamento de carga da arquitetura proposta atuam diretamente sobre os objetos da aplicação para controle de satélites, quer seja copiando ou migrando objetos de máquinas saturadas para máquinas ociosas.

11.7.4 Interface com o Usuário

O serviço de balanceamento disponibiliza para o usuário a seguinte interface:

```
package douto.tms_age3;

   public interface Ibalance {
     public String atender_solicitacao(String obj);

     };
```

Toda solicitação de serviço do usuário, ao software de controle de satélites, é analisada pelo serviço de balanceamento, através do método “Atender_solicitação”. O código abaixo ilustra a utilização desse método pelo usuário.

```
// copyright 1997, 1998 objectspace

package douto.tms_age3;

import java.io.*;
import java.io.RandomAccessFile;
import java.util.Vector;
import com.objectspace.lib.timer.*;
import com.objectspace.lib.facets.*;
import com.objectspace.voyager.*;
import com.objectspace.voyager.corba.*;

public class Client1
  {
   static  Ialcantara alc;
   static  Ibalance balance;
  .
  .
  .
  .
```

```
  public static void main( String[] args )
    {
     Vector tab_aux = new Vector();
    try
     {
     Voyager.startup();

     // Conexão com o serviço de balanceamento
     balance  = (Ibalance)Namespace.lookup("//150.163.20.7:8400/balance" );


// Recuperar a origem da conexão

     String   origem   = System.getProperty("user.name");

     Stopwatch watch = new Stopwatch();
     watch.start();
     //
     //Solicitar o serviço de balanceamneto
     String endereco = balance.atender_solicitacao("tms_alc");
      System.out.println("endereco retornado "+ endereco);

      alc  = (Ialcantara)Namespace.lookup(endereco );


    System.out.println( "frame = " + alc.get_frame());
     watch.stop();
     System.out.println( "time = " + watch.getTotalTime() + "ms" );


     }
    catch( Exception exception )
     {
     System.err.println( exception );
     }

    Voyager.shutdown();
    }
  }
```

11.8 - SERVIÇO DE PERSISTÊNCIA

No protótipo implementado do serviço de persistência, encontrou-se as seguintes classes (Figura 11.8):

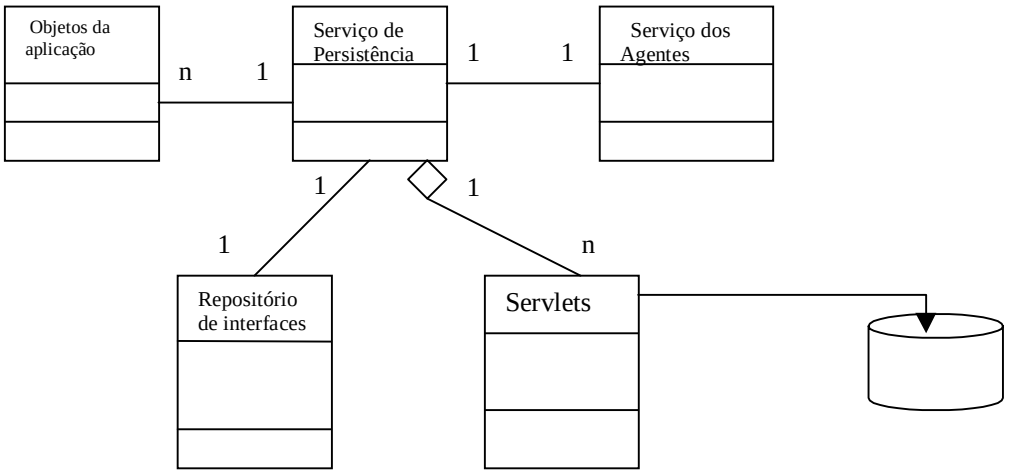


Fig.11.8 Diagrama de classes do Serviço de Persistência.

a) Serviço dos agentes: O serviço dos agentes disponibiliza para o serviço de persistência os parâmetros *Taxa de disponibilidade I/O e nome do nó,* armazenados na base de configurações. O serviço de persistência utiliza esses parâmetros no processo de armazenamento/recuperação de um objeto, no sistema de armazenamento. Primeiro, esse serviço consulta a *base de armazenamentos,* para verificar onde dever ser armazenado/recuperado esse objeto. Segundo, faz-se um levantamento dos nós que contém a *maior taxa disponibilidade de I/O (*informação recuperada da *tabela de nós).* Finalmente, seleciona o nó com a maior taxa, desde que ele esteja incluso na *base de armazenamento*.

b) Objetos da aplicação

Esta interface disponibiliza para os objetos da aplicação, os serviços de acesso a banco de dados, tais como, os serviços de armazenamento, recuperação e exclusão de objetos no banco de dados.

```
package douto.tms_age3;

  public interface IPersistencia {

    public void armazenar( String nome_interface, String nome_objeto);
    public void recuperar (String nome_interface, String nome_objeto);
```

};

c) Repositório de Interfaces

O repositório de interfaces apresenta-se como a fonte de informações de todos os objetos da aplicação para controle de satélites. O conteúdo destas informações descreve, com detalhes, os metadados de cada objeto (os atributos com seus respectivos tipos e os métodos que acessam esses atributos). De posse das informações mantidas pelo repositório de interfaces, o serviço de persistência tem a capacidade de acessar dinamicamente esse objeto na memória. A Figura 11.9 elucida a carga da interface do objeto telemetria da estação de Cuiabá. Esse objeto contém dois atributos: o atributo "frame", compreende as telemetrias recebidas do satélite, e o atributo "time", a data e a hora da recepção. Além desses dois atributos, o objeto contém os métodos:

Get_frame: Para recuperar o conteúdo do "frame" do objeto.

Atrib_frame: Para atribuir um novo "frame" ao objeto.

Get_time: Para recuperar a data de processamento do "frame".

Atrib_time: Para atribuir uma data ao "frame" que está sendo recebido do satélite.

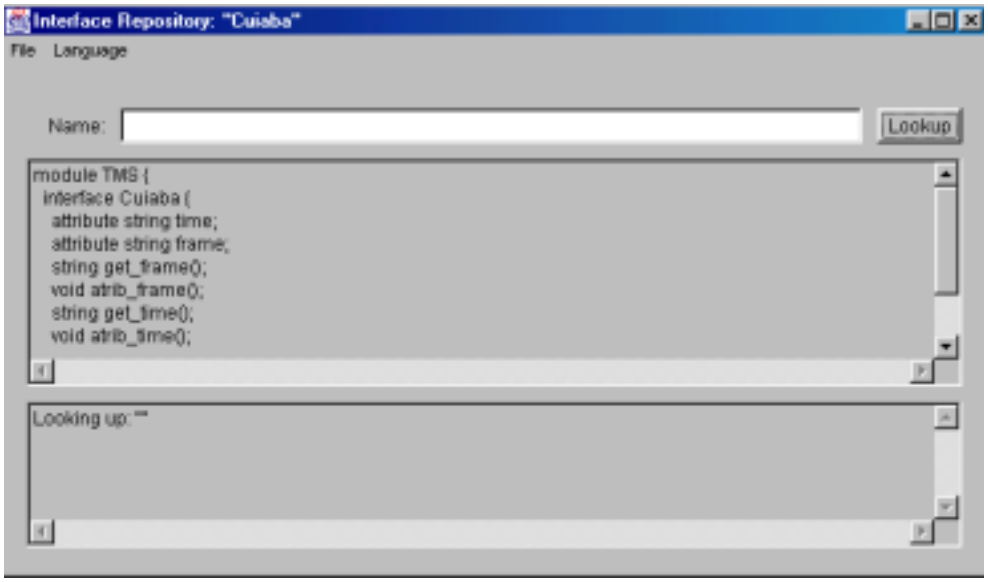


Fig. 11.9 – Carga do repositório de interfaces para o objeto Telemetria da estação de Cuiabá.

d) Serviço de Persistência

O objetivo desse item consiste em apresentar os principais métodos agregados ao serviço de persistência. A Figura 11.10 ilustra um possível diagrama de sequência para a interação entre o serviço de persistência e um objeto da aplicação, a seguir comenta-se os principais métodos que participam desse diagrama.

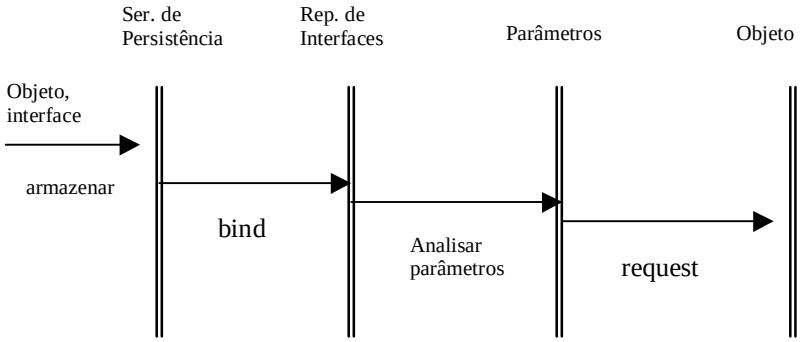


Fig. 11. 10 Diagrama de sequência do serviço de persistência.

a) Método "bind"

O método "bind" apresenta-se como o responsável pelo acesso ao repositório de interfaces e a recuperação dos metadados de um objeto (atributos e métodos de acesso de um objeto) . O item A-6, do apêndice A, contém a implementação desse método.

b) Método "Analisar parâmetros"

O método "analisar parâmetro", consiste em identificar para cada atributo de um objeto, os seus respectivos tipos. Esse método antecede a execução do método "request", ou seja, antes de acessar qualquer atributo de um objeto na memória é necessário conhecer qual o seu tipo de dado. Vide item A-6 do apêndice A.

c) Método "Request"

Após a identificação dos parâmetros de cada objeto e seus respectivos tipos, com o auxílio do método "bind" e "analisar_param", respectivamente, atribui-se ao método "Request" a responsabilidade de acessar e recuperar os atributos do objeto da memória. Vide item A-6 do apêndice A.

11.9- SIMULAÇÃO DA ARQUITETURA SICSD

Objetivando testar de forma exaustiva os algoritmos propostos no serviço de balanceamento, construi-se um simulador, buscando refletir um ambiente computacional similar ao existente para o teste da arquitetura SICSD. Como já foi mencionado na seção 11.1, utilizou-se três computadores; entretanto, na simulação considerou-se a existência de quatro nós para o ambiente. Para a implementação do simulador da arquitetura SICSD, levou-se em conta, a possibilidade de distribuir seis objetos entre esses quatro nós. A disponibilidade de CPU de cada nó varia de acordo com:

- o número de objetos que foram instanciados;
- o número de conexões existentes em cada objeto e
- as operações de I/O realizadas por aquele nó.

Como por exemplo, a carga de um objeto em um nó pode consumir de 5 à 15 % da disponibilidade de CPU. A cada solicitação de serviço que esse objeto atende, consome de 2% à 10% da disponibilidade de CPU, e a cada operação de I/O de 5% à 10% da disponibilidade de CPU. Vale ressaltar que esses valores podem ser configuráveis; entretanto, nesta simulação, os valores decrementados da disponibilidade de CPU em cada nó foram:

- 8%, para cada instância de um objeto em um nó;
- 2%, para cada conexão recebida pelo objeto e
- 5%, para cada operação de I/O realizada no nó.

Considerando os valores acima, um nó se apresenta com a disponibilidade de CPU igual a 86%, caso exista um objeto instanciado, que consome uma carga de 8% de CPU e atenda a três solicitações de serviços(100% - 8% - 3*2%). A Figura 11.11 elucida a interface principal do simulador com o usuário.

O processo de simulação consiste em gerar de forma aleatória, conexões para os objetos, oriundas também de forma aleatória, dos quatro nós existentes. O objetivo desta simulação é validar as características de flexibilidade e o dinamismo propostos na

arquitetura. Ou seja, a arquitetura deve ser capaz de replicar, ou migrar objetos, objetivando distribuir o carga de processamento entre os nós existentes. Para facilitar a visualização dos objetos nos nós, defini-se a seguinte legenda:

 O nó contém a base de armazenamento deste objeto

 Este objeto foi replicado

 Este objeto foi migrado

E para facilitar a visualização, definiu-se cores para cada um dos objetos instanciados em um dos quatro nós:

 Objeto 1

 Objeto 2

 Objeto 3

 Objeto 4

 Objeto 5

 Objeto 6

A simulação considerou quatro cenários:

a) Cenário 1: O cenário 1 elucida uma distribuição similar de carga entre os quatro computadores. No nó 1 foram instanciados os objeto 1, 2 e 4; no nó2 os objetos 3, 4, 5, 6; no nó 3 os objetos 4 e 5 e no nó 4 os objetos 2 e 6. A Tabela 11.4 mostra a carga inicial de cada nó e a Figura 11.12 elucida esse cenário.

TABELA.11.4 – SITUAÇÃO INICIAL DO CENÁRIO 1

Nó	**Objetos**	**Carga inicial**	**Disponibilidade de CPU**
No1	1,2,4	8*3 = 24	100-24=76%
No2	3,4,5 e 6	8*4=32	100-32=68%
No3	4 e 5	8*2=16	100-16=84%
No4	2 e 6	8*2=16	100-16=84%

A Figura 11.13 ilustra a distribuição final de objetos instanciado em cada nó e o gráfico da Figura 11.14 ilustra a disponibilidade de CPU nos quatro nós depois de um determinado período de simulação. De acordo com a Figura 11.13, ocorreram tanto migrações quanto replicações de objetos entre os nós. No gráfico da Figura 11.14 observa-se que a disponibilidade de CPU dos quatro nós tendem a cair, já que o processo de simulação continua a gerar solicitações de serviços de forma aleatória. Entretanto, a diferença entre a disponibilidade de CPU dos quatro nós tende a ser menor, isto em função da atuação do algoritmo de balanceamento, que objetiva distribuir de forma proporcional, a carga de CPU entre os nós.

b) Cenário 2: Dois nós sobrecarregados

O cenário 2 ilustra a atuação do serviço de balanceamento, quando dois nós estão sobrecarregados e existem outros dois nós com alta taxa de disponibilidade. A Tabela 11.5 e a Figura 11.15 ilustram a situação inicial do cenário 2.

TABELA 11.5 - SITUAÇÃO INICIAL DO CENÁRIO 2

Nó	Objetos	Carga inicial	Disponibilidade de CPU
No1	1,2,3,4,5 e 6	8*6 = 48	100-48=52%
No2	1,2,3,4 e 6	8*5 = 40	100-40=60%
No3	0	0	100%
No4	0	0	100%

A Figura 11.16 mostra os objetos migrados e replicados em cada nó e a Figura 11.17 elucida a disponibilidade de CPU dos quatro nós, durante o processo de simulação. Os nós disponíveis no contexto passam a receber migrações e replicações de objetos dos nós saturados, por conseguinte, depois de um determinado tempo, ambos se apresentam com taxas de disponibilidade de CPU semelhantes.

c)Cenário 3 – Uma máquina sobrecarregada

O cenário 3 descreve a possibilidade de carregar a aplicação para controle em um único nó. Espera-se, que a atuação do serviço de balanceamento, otimize os recursos computacionais existentes na a arquitetura SICSD, migrando e replicando esses objetos, para os outros nós definidos para a arquitetura. A Tabela 11.6 e a Figura 11.18 ilustram a situação inicial do cenário 3.

TABELA 11.6 - SITUAÇÃO INICIAL DO CENÁRIO 3

Nó	Objetos	Carga inicial	Disponibilidade de CPU
No1	0	0	100%
No2	0	0	100%
No3	1,2,3,4,5 e 6	8*6 = 48	100-48=52%
No4	0	0	100%

A Figura 11.19 elucida os objetos migrados e replicados em cada nó, ao término da simulação do cenário 3 e a Figura 11.20 ilustra a disponibilidade de CPU dos quatro nós, durante o processo de simulação. Nesse caso extremo, onde a concentração de carga apresenta-se em apenas um nó, o papel do serviço de balanceamento é fundamental, à medida que, procura migrar ou replicar objetos de um máquina saturada para uma máquina ociosa.

d)Cenário 4: Manutenção em um nó

O serviço de balanceamento também presta o serviço de manutenção em um nó. Sua atuação consiste em alterar o status desse nó de “ready” para “suspend”, em seguida, na criação desses objetos, em máquinas remotas mais ociosas. A Tabela 11.7 e a Figura 11.21 ilustram a situação inicial do cenário 4.

TABELA. 11.7 - SITUAÇÃO INICIAL DO CENÁRIO 4

Nó	Objetos	Carga inicial	Disponibilidade de CPU
No1	1, 2 e 3	8*3=24	100-24=76%
No2	2	8*1 = 8	100-8=92%
No3	1,2,3,4,5 e 6	8*6 = 48	100-48=52%
No4	0	0	100%

A Figura 11.22 elucida os objetos migrados em cada nó, no término da simulação do cenário 4, considerando que o nó 3 entrará em processo de manutenção.

CAPÍTULO 12

CONCLUSÃO

Existe somente um avanço. A hora mais escura da noite é o prenúncio do alvorecer. Nos dias sombrios de inverno, prosseguem incessantemente os preparativos para a primavera. Tudo que parece dificuldade é, na verdade, valioso "trampolim" para um grande saldo. (Do livro Inochi Hikidasun Rensukai- Seicho Tanuguchi).

Este trabalho de pesquisa concentrou-se em apresentar uma arquitetura que pode ser aplicada ao software de controle de satélites do INPE, com a finalidade de suprir as limitações das versões atuais que foram elucidadas no final do Capítulo 5, bem como agregar novas funcionalidades. Como foi apresentado nos capítulos anteriores, esta arquitetura engloba os objetos da aplicação propriamente ditos (telemetrias, telecomandos etc.), bem como os serviços propostos, os quais são os responsáveis por disponibilizar uma infra-estrutura flexível e dinâmica para os objetos da aplicação.

O serviço de balanceamento proporciona a disponibilidade de serviços, uma vez que sua atuação consiste em replicar ou migrar objetos que estejam sendo muito solicitados. A predisposição de mais de um objeto, que presta o mesmo serviço, garante a continuidade das operações de controle de um satélite, mesmo em falhas em um dos objetos ou no nó onde ele se encontra instanciado.

O serviço de balanceamento sustenta as características de flexibilidade e dinamismo anunciadas na arquitetura. A primeira provém da capacidade de adaptabilidade da arquitetura proposta, em face das situações a que expõe um aplicativo de controle de satélites durante a sua existência. A fase de lançamento talvez seja a mais crítica, quando um número variável de usuários (projetistas, controladores, engenheiros) desejam obter as primeiras informações do satélite. Nesse estágio, o software para controle de satélite deve ser robusto, flexível e capaz de disponibilizar os serviços de acordo com a demanda. Ao término dessa fase, o satélite entra em operação de rotina e, nesse caso, a operação se restringe

aos usuários controladores de satélites. Novamente aflora a característica de flexibilidade do sistema, já que nessa fase o serviço de balanceamento atua no sentido de eliminar o excesso de objetos que foram criados para atender à primeira fase e agora se encontram ociosos. Entretanto, o período crítico não se restringe apenas ao lançamento. A demanda pela utilização do aplicativo de controle de satélite aumenta nos períodos de manobra. As operações de manobra ocorrem mensalmente e objetivam mudar a atitude de um satélite; portanto, envolve usuários de dinâmica orbital, usuários responsáveis pelos painéis solares etc. Novamente, a atuação do serviço de balanceamento garante a disponibilidade do software para controle de satélites nesses períodos de "picos".

A capacidade de os objetos migrarem de uma máquina para outra, conforme as solicitações de serviços dos usuários e a disponibilidade de CPU do nó onde o objeto está instanciado, confere à arquitetura um caráter dinâmico.

Um ambiente distribuído flexível e dinâmico difere de um ambiente distribuído estático, pela capacidade de o primeiro se ajustar à demanda por solicitações de serviços. O aumento da demanda propicia a criação de novas réplicas do objeto que atendam a essa solicitação. Em contrapartida, a queda na demanda pode disparar o processo de destruição dos objetos ociosos no sistema.

O balanceamento de carga ainda são técnicas caras e difíceis de ser implementadas. A maioria dos aplicativos que implementam a especificação CORBA propala uma distribuição seqüencial de carga. Ou seja, se o sistema distribuído contém mais de um objeto, para atender a uma mesma solicitação de um usuário, a conexão é estabelecida de forma seqüencial. Primeiro, com o objeto 1, depois com o objeto 2 etc. Ao alcançar o último objeto, retorna-se para o primeiro.

O balanceamento de carga proposto nesse trabalho difere do proposto pela especificação CORBA em importantes aspectos:

- A tentativa de balancear a carga do sistema acontece desde o processo de atendimento da solicitação de serviço de um usuário. Ou seja, o processo de atendimento das solicitações de serviços dos usuários é sustentado por um

algoritmo capaz de localizar qual o objeto está mais apto para atender a essa solicitação.

- A capacidade do serviço de balanceamento em migrar ou replicar objetos de máquinas saturadas para máquinas ociosas.

Talvez a imaturidade, até o momento, dos banco de dados orientados a objetos, proporcione uma instabilidade tecnológica no desenvolvimento dos serviços de persistência. O reflexo deste quadro pode ser observado ao analisar o histórico dos serviços de persistência propostos pela especificação CORBA. A primeira versão, Persistent Object Service-POS, foi cancelada e a Segunda, Persistent Sate Service (PSS), até o momento ainda não foi implementada pela maioria dos ORBs.

O serviço de persistência proposto incrementa o PSS proposto na especificação CORBA, à medida que elimina a necessidade do desenvolvedor de software construir os arquivos PSDLs. O desenvolvedor concentra esforços na especificação dos objetos de negócio. Cada objeto de negócio contém métodos de acesso ao sistema de armazenamento de dados. O código interno desses métodos simplesmente aciona o serviço de persistência proposto.

A grande contribuição do serviço de persistência proposto talvez esteja na capacidade deste serviço explorar os recursos já existentes no repositório de interfaces. A partir das informações geridas pelo repositório de interfaces, o serviço de persistência tem a capacidade de acessar os dados de um objeto, em tempo de execução, montar a SQL de conformidade com os atributos e seus respectivos tipos e, finalmente, armazená-los em um banco de dados.

O ganho de explorar o conteúdo do repositório de interfaces reside no fato de que as alterações nos atributos de um objeto são imediatamente refletidas no repositório de interfaces e automaticamente visíveis para o serviço de persistência. Ou seja, não é necessária a compilação do serviço de persistência. Entretanto, mudanças nos atributos de um objeto, utilizando o serviço de persistência CORBA, necessita da compilação e geração de códigos para realmente refletir as alterações realizadas**.**

A união da utilização do conceito de distribuição com o conceito de agentes permitiu

construir uma aplicação dinâmica e flexível, capaz de se adaptar às necessidades do usuário. Se a demanda por determinado serviço aumenta, os agentes supervisores são capazes de detectar o objeto que atenda esse serviço, bem como o número de usuários solicitantes. Esta informação é repassa para os agentes mensageiros responsáveis por disseminá-la para os outros nós, dentro do domínio de rede, que o sistema está carregado. O objeto que atenda a esta demanda poderá migrar ou ser replicado. Desta forma, o sistema se auto-ajusta para atender às solicitações de seus usuários.

A tecnologia de agentes poderia ter sido substituída por programas ou qualquer outro recurso computacional que implementassem as mesmas funcionalidades, como por exemplo, os agentes móveis poderiam ter sido substituídos por banco de dados distribuídos. Ou até mesmo utilizar o recurso de troca de mensagens entre os agentes supervisores para substituir os agentes mensageiros. A utilização das funcionalidades dos agentes foi com intuito de explorar esta tecnologia, já que pelas próprias características intrínsecas, os agentes, se apresentam melhor adaptado para monitoração de aplicações distribuídas.

Dentre as contribuições preponderantes deste trabalho, para o grupo de desenvolvimento de software de controle de satélites do INPE, deve-se ressaltar:

- A arquitetura proposta, sustentada pela atuação do serviço de balanceamento, permite a continuidade do controle de um satélite, mesmo em situações de contingências, como por exemplo falha em um servidor, falha em um objeto etc. Isto em função da disponibilidade de serviços proporcionada pelas réplicas dos objetos.
- A disponibilidade de serviços pode trazer, como consequência imediata, um incremento no desempenho, já que a arquitetura proposta tem cópias de um mesmo objeto distribuído em máquinas distintas para atender às solicitações dos usuários. Atualmente, as versões do SICS estão centradas na arquitetura cliente-servidor. Desta forma, o aumento do número de clientes compromete o desempenho do servidor.
- Flexibilidade para atender à dinâmica dos programas espaciais (satélites a serem controlados poderão aumentar e até diminuir ao longo do tempo).Atualmente, a cada

nova missão é necessário criar um ambiente dedicado para controle do satélite (um servidor e um conjunto de clientes). Além dos custos envolvidos, existem serviços semelhantes em cada novo ambiente. A idéia é ter um conjunto de servidores para controlar um conjunto de satélites, sem ter a rigidez na alocação dos serviços de controle e servidor dedicado a um dado satélite.

Alguns resultados deste trabalho de pesquisa já foram publicados no SPACE OPS – jun/2000 (Ferreira,2000) e outros serão publicados no 19th AIAA International Communications Satellite Systems Conference (Ferreira, 2001), ambos na França.

Além dos trabalhos já publicados, a contribuição deste trabalho de pesquisa não se restringiu apenas à tese de doutorado aqui apresentada, pois foram originados pelo menos três vertentes de pesquisa. O serviço dos agentes foi a motivação do primeiro trabalho de pesquisa, tendo sido tema da dissertação de Silva (Silva,2000). O serviço de persistência será explorado em mais detalhes em uma futura dissertação de mestrado. Acredita-se que um ambiente flexível e dinâmico exija, do serviço de segurança, facilidades que até o momento não estão implementadas em nenhum ORB; portanto, uma análise mais criteriosa neste serviço pode ser tema de tese de outros trabalhos de pesquisa.

Enfim, os serviços agregados (persistência, segurança, agentes e balanceamento) ao aplicativo para controle de satélites tornaram-se fatores preponderantes e proporcionaram a criação de um ambiente capaz de se adaptar dinamicamente às solicitações dos controladores e demais usuários do sistema, melhorando um conjunto de características, como desempenho, flexibilidade, confiabilidade e utilização dos recursos computacionais disponíveis.

REFERÊNCIAS BIBLIOGRÁFICAS

Albuquerque, F. O desenvolvimento de aplicações distribuídas com a linguagem java. **Developers' Magazine**. v. 2, n.23, p.20-21, July 1998.

Andleigh, P. K.; Gretzinger, M. R. **Distributed object-oriented data-systems design**. New York: Prentice-Hall, 1992. 495 p.

Barth, T.; Flender, G.; Freisleben, B.; Thilo F**. Load Distribution in a CORBA Environment. Proceedings of the International Symposium on Distributed Objects and Applications**. IEEE Computer Society. [on-line]. Disponível em: <http://www.ieee.com>. 1999. Acesso em: Mar. 2000.

Bernstein, P. A. Middleware: A Model for distributed system services. **Communication of the ACM**. v. 39, n. 2, p.86 - 98, Feb. 1996.

Betz, M. Interoperable objects. **Dr. Dobb's Journal**. v. 19, n. 11, p.18 - 39, Oct. 1994.

Blakley B. **CORBA Security-** An introduction to safe computing with objects. Massachusetts: Addison-Wesley, 1999. 135 p.

Bray, M. **Middleware**. [online]. Jan. 1997. Disponível em: <http://www.sei.cmu.edu/str/descriptions/middleware.html>. Acesso em: Set. 1998.

Caglayan, A; C. Harrison. **Agent sourcebook:** a complete guide to desktop, internet, and intranet agents. New York : Wiley Computer Publishing. 1997. 570 p.

Cardoso, P.E.; Gonçalvez, L.S.C.; Ambrosio, A.M. Lessons learned in adopting PCs at the Brazilian Satellite Control Center, [CD-ROM]. In: Symposium on space mission operations and ground data systems, 5., SPACEOPS98, Tóquio, Japão, June, 1998. **Anais**. National Space Development Agency of Japan(NASDA), Tóquio, 1998.

Chappell, D. **Understanding activeX and OLE**. New York: Microsoft Press, 1996. 328 p.

Chin, R. S.; Chanson, S. T. Distributed object-based programming systems. **ACM Computing Surveys**. v. 23, n. 1, p.91 – 124, Mar. 1991.

Microsoft Component Services(COM): a technical overview. [online]. Disponível em: < http://www.microsoft.com/com/wpaper/default.asp>. Ago. 1998. Acesso em: Out. 1998.

CORBA - Object Management Architecture(OMG): **Architecture and specification.** [online]. Disponível em: <http://www.omg.org/>. Jun. 1997. Acesso em: Mar. 1998.

CORBA - Object Management Architecture(OMG): **Security service specification**. [online]. Disponível em: <http://www.omg.org/>. Jun. 1997. Acesso em: Ago. 1999.

CORBA - Object Management Architecture(OMG): **CORBAfacilities -Common facilities architecture**. [online]. Disponível em: <http://www.omg.org/>. Set. 1998. Acesso em: Ago. 1999.

CORBA - Object Management Architecture(OMG): **CORBAfacilities- Mobile Agent Facility(MAF).** [online]. Disponível em: <http://www.omg.org/pub/docs/formal/>. Mar. 1998. Acesso em: Jan. 2000.

CORBA - Object Management Architecture(OMG): **CORBAservices- Common object services specification**. [online]. Disponível em: <http://www.omg.org/library/csindx.html>. Set. 1998. Acesso em: Jan. 2000.

CORBA - Object Management Architecture(OMG): **CORBAPersistence- Persistent Object Service**. [online]. Disponível em: <http://www.omg.org/technology/documents/formal/persistent_object_service.htm>. Abr. 1999. Acesso em: Jan. 2000.

Costa. S. R. **Objetos distribuídos:** conceitos e padrões. São José dos Campos. 198p. Dissertação(Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, 2000.

Coulouris, G.; Dollimore, J.; Kindberg, T. **Distributed systems concepts and desing**. New York: Addison-Wesley, 1994. 356 p.

CRU - Trusted computer system evaluation criteria. [online]. Department of Defense, CSC-STD- 001-83, Library No. S225 711. [online]. Disponível em: <:http://www.cru.fr/securite/Documents/generaux/orange.book>. Ago.1983. Acesso em: Fev. 1999.

Cyclades Brasil. **Guia internet de conectividade**. São Paulo: 1996. 156 p.

Daval, C.; Lacroix, M.; Guyennet, H. **Resource Balancing Using Trader Federation**. Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000). IEEE Computer Society. Disponível em: <http://www.ieee.com>. 1999. Acesso em: Mar. 2000.

DCE- AES/Distributed computing – Remote procedure call. Revision B, Open Software Foundation. [online].Disponível em: <http://www.osf.org/mall/dce/free_dce.htm>. Out. 1998. Acesso em: Fev. 2000.

Eastman, J. Building scalable CORBA distributed systems. **Object Magazine**. v. 7, n. 2 , p.35 - 37 e 49, Abr. 1997.

Eriksson H.; Penker M. **UML Toolkit**. New York: Wiley Computer . 1998. 397 p.

Etzioni, O. Moving Up the information food chain: Deploying softbots on the world wide web. In: National Conference on Artificial Intelligence ,13., (AAAI-96), Portland, 1996. Proceeding. Portland, 1996. AAAI Press / MIT Press, p.1322-1326.

Ferreira, M.G.V. **Componente gerenciador de dados configurável**. São Jóse dos Campos. 177p. Dissertação(Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, 1996.

Ferreira, M.G.V., Nakanishi, T., Cunha, J.B.S, Silva, J.O. A Dynamic and flexible architecture for distributed objects applied to Satellites Control Software. [CD-ROM]. In: SPACE OPS, Toulouse, França, 2000. **Proceedings**. Centre National D' Etudes Spatiales (CNES). Toulouse, 2000. Centre Informatique.

Ferreira, M.G.V., Nakanishi, T., Cunha, J.B.S, Silva, J.O. Distributed objects and inteligent agents in the satellite control software. [CD-ROM]. In: AIAA International Communications Satellite Systems Conference, 19., Toulouse, 2001. **Proceedings**. Centre National D' Etudes Spatiales (CNES). Toulouse, 2001. Centre Informatique.

Franklin, S.; Graesser, A. **Is it an Agent, or just a program?: A taxonomy for autonomous agents**. In: International Workshop on Agent Theories, Architectures, and Languages, 3., Proceedings. Springer-Verlag, University of Memphis, 1996. Disponível em: < http://www.msci.memphis.edu/~franklin/AgentProg.html >. 1996. Acesso em: Mar. 1998.

Gonçalvez L.S.C.; Cardoso, P.E.; Ambrosio, A.M. Satellite Control Center: solution for an adaptable system. [CD-ROM]. In.: Symposium os Small Satellites Systems and Services – SSSS, 4., Antibes, France, 1998. **Anais**. Centre National D' Etudes Spatiales (CNES). Toulouse, 1998. Centre Informatique.

Gopal S.; Vajapeyam, S. Load balancing in a heterogeneous computing environment. In: Hawaii International Conference on System Sciences (HICSS'98) 31., IEEE Computer Society. Disponível em: <http://www.ieee.com>. 1998. Acesso em: Mar. 2000.

Grimes, R. **DCOM programming professional**. Canada: Wrox Press, 1997. 565 p.

Gunji T. **Plataforma multware:** suporte a objetos em tempo de execução. Campinas. 185p. Dissertação(Mestrado em Computação)- Universidade de Campinas, 1995.

Hermans, B. **Intelligent software agents on the Internet:** an inventory of currently offered functionality in the information society & a prediction of (near-) future development . [online]. PhD Thesis. Tilburg University, 1996. Disponível em: <http://www.firstmonday.dk/issues/issue2_3/ch_123/index.html>. 1996. Acesso em: Mar. 1998.

Hohl, F.; Klar, P.; Baumann, J. **Efficient code migration for modular mobile agents**. [Online]. Disponível em: http://ncstrl.informatik.uni-stuttgart.de/Dienst/UI/2.0/Describe/ncstrl.ustuttgart_fi/TR> Mar. 1997. Acesso em: Fev. 1999.

Instituto Nacional de Pesquisas Espaciais. Centro de Rastreio e Controle de Satélites(INPE/CRC). **Descrição interna do controle de satélites**. Versão 1.0. São José dos Campos, 1989. 448p. (CCSSCSDI-SWR-002).

Instituto Nacional de Pesquisas Espaciais(INPE). **SCD2 operation handbook**. São José dos Campos, 1993. 359p.(A-MIN-0085).

Intel Celeron processor and Intel 810E performance brief. [online]. Disponível em: <http://www.intel.com/procs/perf/Celeron/brief/charts.htm>. Fev. 1998. Acesso em: Mar. 2000.

Kunz T. The influence of different workload descriptions on a heuristic load balance scheme. **IEEE Trans. Software Eng**. v. 17 p. 725-730. July. 1991

Java **- Java distributed object model.** [online]. Disponível em: <http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmi-objmodel.doc.html>. Jan. 1997. Acesso em: Mar. 1999.

Jacobson, I. **Object-oriented software engineering –** a use case driven approach. New York: Addison-Wesley. 1992. 528 p.

Jansen, W.; Karygiannis, T. **Mobile agent security** NIST Special Publication 800-19, National Institute of Standards and Technology - Computer Security Division, 1999, Gaithersburg, MD 20899. {jansen, karygiannis} @nist.gov

Lee, G.H. Issues of the State Information for Location and Information policies in distributed load balancing algorithm. In: Euromicro Conference (EUROMICRO '99), 25., IEEE Computer Society. Disponível em: <http://www.ieee.com>. 1999. Acesso em: Mar. 2000.

Manola, F. **Object model capabilities for distributed object management**. [online]. Disponível em: <http://info.gte.com/pubs/index.htm>. Mar. 1997. Acesso em: Abr. 1999.

Montez, C.; Oliveira, R. S. ; Fraga , J. An adaptive model for programming distributed real-time applications in CORBA. In International Conference of the Chilean Computer Science Society, 18., IEEE Computer Society. Disponível em: <http://www.ieee.com>. 1998. Acesso em: Mar. 2000.

Mowbray, T. J.; Ruh, W. A. **Inside CORBA distributed object standards and application**. New York: Addison-Wesley, 1997. 376 p.

Nimer, F. A Importância da padronização de ambientes distribuídos. **Developers' Magazine**. v. 3, n.26, p.16-17, Out. 1998.

OMG, **Agent Technology – Green Paper** – Agent Working Group. [online]. Disponível em: <http://www.omg.org>. Mar. 1998. Acesso em: Mar. 2000.

Orfali, R.; Harkey, D.; Edwards, J. **The essential distributed objects:**survival guide. New York: John Wiley & Sons, 1996. 604 p.

Orfali, R; Harkey, D. **Client/Server programming with Java and CORBA**. New York: Wiley Computer Publishing, 1997. 657 p.

Otte, R.; Patrick, P.; Roy, M. **Understandig CORBA**. New York: Prentice Hall, 1996. 250 p.

Pedrick, D.; Weedon, J.; Goldberg, J.; Bleifield, E. **Programming with visibroker**. New York: Wiley Computer Publishing, 1998. 435 p.

Purao, S.; Jain, H.; Nazareth, D. Effective Distribution of object-oriented applications. **ACM Comunications**, v. 41, n. 8, p.100-108, Aug. 1998.

Queiroz, J. A. G.; Madeira, E. CORBA: Plataforma aberta de objetos distribuídos. **Developers' Magazine**, v. 2, n.21, p.26-30, May. 1998.

Russell, S.; Norvig, P. **Artificial Intelligence:** a modern approach. Englewood Cliffs. Prentice Hall, 1995. 912 p.

Schnekenburger, T.; Stal, M. **Load distribution for CORBA Environments**. [online]. Disponível em: <http://wwwpaul.informatik.tu-muenchen.de/projekte/dcw/da_rackl/>. Mar. 1998. Acesso em: Abr. 1999.

Sessions, R. **Ten rules for distributed object systems**. [online]. Disponível em: <http:// www.objectwatch.com/magart5.htm>. Ago. 1998. Acesso em: Jan. 1999.

Silva, J. O. **Agentes móveis para apoio ao sistema de controle de satélites distribuído e dinâmico**. São José dos Campos. 198p. Dissertação(Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, 2000.

Stone, C.; Curtis, D.; Bradley, M. **IIOP: OMG's internet inter-ORB protocol - A brief description**. [online]. Disponível em: <http://www.omg.org/library/iiop4.html>. Set. 1998. Acesso em: Nov. 1998.

Sun Microsystems, **The Java Language:** A White Paper, Technical Report. Sun Microsystems [online]. Disponível em: <http://www.sun.com>. Jan. 1997. Acesso em: Abr.1999.

Tallman, O.; Bradford, J. K. **COM versus CORBA: A decision framework**. [online]. Disponível em: <http://www.quoininc.com/quoininc/COMCORBA.html>. Ago. 1998. Acesso em: Mar. 1999.

Taylor, C. J. Object-oriented concepts for distributed systems. **Computer Standards & Interfaces,** n.15, p.167 - 274, 1993.

Thißen, D.; Neukirchen, H**.** Managing services in distributed systems by integrating trading and load Balancing. In IEEE Symposium on Computers and Communications (ISCC 2000), 5., IEEE Computer Society. Disponível em: <http://www.ieee.com>. 1999. Acesso em: Mar. 2000.

Visibroker- **Visigenic- Programmer's guide**. Version 3.2. 1998. 235 p.

Vondrak, C. **Remote procedure call**. [online]. Disponível em: <http://www.sei.cmu.edu/str/descriptions/rpc.html>. Out. 1998. Acesso em: Mar. 1999.

ZDNET, **Where technology takes you.** [online]. Disponível em: <www.zdnet.com> Mar. 1997. Acesso em: Mar. 2000

Voyager - ObjectSpace, Inc., **Voyager core technology 2.0:** user guide**.** Disponível em: <http://www.objectspace.com/products/voyager/>. 1998. Acesso em: Out. 1998.

Wallnau, K. **Common object request broker architecture**. [online]. Disponível em: <http://www.sei.cmu.edu/str/descriptions/corba.html>. Set. 1998. Acesso em: Out.1999.

White, T.; Bieszczad, A.; Pagurek, B. **Mobile agents for network management**. [online]. Disponível em: <http://citeseer.nj.nec.com/context/849730/45266>. 1998. Acesso em: Abr. 1998.

Wong, D.; Paciorek, N.; Moore, D. Java-based Mobile Agents. **ACM Communications** . v. 42, n. 3, p.92 – 102, Mar. 1999.

APÊNDICE A

Este capítulo encontra-se dividido em seções, contendo os seguintes códigos:

A-1: A carga da arquitetura SICSD
A-2: Simulador de Satélites
A-3: Objetos da Aplicação
A-4: Serviço dos agentes
A-5: Serviço de balanceamento
A-6: Serviço de Persistência

A-1 : A carga da arquitetura SICSD

Parte do código abaixo ilustra o processo de instanciamento dos objetos (telemetria e telecomando), o serviço dos agentes e o serviço de persistência, instanciados no nó PATRAS.

```
.
.
.
try
  {

    // endereco do servidor para uso do cliente voyager
   Voyager.startup("7001");

   // NO  PATRAS

// Objeto telemetria de alcântara
tms_alc =
(Ialcantara)Factory.create(alcantara.class.getName(),"//150.163.20.7:8200/tms_alc");

// Objeto telecomando de alcântara
 tc_alc = (Itc_alc) Factory.create( tc_alc.class.getName(),"//150.163.20.7:8201/tc_alc");

// Objeto agente supervisor
 agente = (iagente)
Factory.create(agente.class.getName(),"//150.163.20.7:8202/agente");

// Serviço de Persistência
 persistencia =(ipersistencia)
Factory.create(persistencia.class.getName(),"//150.163.20.7:8203/persistencia");
.
.
.
```

A-2: Simulador de Satélites

Código do simulador para a estação de Alcântara.

//Classe Simulador de alcântara

```
package douto.tms_age2;

import java.util.*;
import java.lang.*;
import java.io.*;
import com.objectspace.lib.timer.*;
import com.objectspace.voyager.*;
import com.objectspace.voyager.agent.*;
import com.objectspace.voyager.corba.*;
import com.objectspace.lib.facets.*;


// alcantara.java: telemetrias de alcantara

public class simu_alc implements Isimu_alc, Serializable

{
  private String frame;
  private String tlm01;
  private String tlm02;
  private String tlm03;
  private String tlm04;
  private String tlm05;
  private String TC[] = new String[50];
  private int num_tc;
  public String ior;



  // Constructor
  public simu_alc()
  {
    System.out.println("Simulador criado de alc");
   tlm01 = "00000001";
    tlm02 = "00000000";
    tlm03 = "00000001";
     num_tc = 0;

  }
```

```
// Armazenar comandos enviados
//
public void envio(String tc)
{
  num_tc++;

  TC[num_tc] = tc;

}

// get sum
public  String  get_frame()
{
  // Definir variáveis auxiliares
 Random random;
  Date data;
  String aux;

  // gerar o frame ////
  random = new Random();

  // Simular as digitais de acordo com TC's ja enviados


  for (int i = 1; i<= num_tc; i++)
     {
       if (TC[i].equals("101")) {tlm01 = "00000001";}
       if (TC[i].equals("102")) {tlm01 = "00000000";}
       if (TC[i].equals("103")) {tlm02 = "00000001";}
       if (TC[i].equals("104")) {tlm02 = "00000000";}
       if (TC[i].equals("105")) {tlm03 = "00000001";}
       if (TC[i].equals("106")) {tlm03 = "00000000";}
     }


   float tlm4x = Math.abs(random.nextInt())%10000 /100f;
   tlm04 = Float.toString((float)(tlm4x));

     float tlm5x = Math.abs(random.nextInt())%10000 /100f;
     tlm05 = Float.toString(tlm5x);

   //frame

    data = new Date();
    aux = data.toString();
```

```
    return aux.substring(10,20) + tlm01+"/" + tlm02+"/"+tlm03+"/"+ tlm04 +"/"+
tlm05+ "///";

  }

  // set sum
  public  void frame(String val)
  { frame = val;
  }

  // Conexao method
  public void  conexao()
  {
  // A cada conexao zerar os TC's
  num_tc = 0;
   System.out.println("conexão TMS com simulador");
 System.out.println(num_tc);
  }

  // DesConexao method
  public void desconexao()
  {

  }

}
```

A-3 : Objetos da Aplicação

O código abaixo ilustra o protótipo do objeto telemetria para a estação de Alcântara.

```
package douto.tms_age3;

import java.util.*;
import java.lang.*;
import java.io.*;
import com.objectspace.lib.timer.*;
import com.objectspace.voyager.*;
import com.objectspace.voyager.agent.*;
import com.objectspace.lib.facets.*;


public class alcantara implements Ialcantara,  Serializable
{

// public Comp comp_aux;
```

```
Isimu_alc sim_alc;

public alcantara()
 {

    System.out.println("Estacao alc criada");
    //

   //
   // Procura pelo simulador
   try
   {
   sim_alc=(Isimu_alc) Namespace.lookup("//150.163.20.7:8100/sim_alc");
     }
   catch( Exception exception )
     {
     System.err.println( exception );
     }
 }
// Conexao method
 public void conexao()
 {

   sim_alc.conexao();

   }
   public void desconexao()
 { sim_alc.desconexao();
      }

  public String get_no()
 {
  return "patras";
   }

  public String get_frame()
 {

  return sim_alc.get_frame();
   }

public String get_nome()
 {
  return "alcantara";
   }
```

}

A- 4 - Serviço dos agentes

a) Agente Supervisor

```
package douto.tms_age;

import com.objectspace.lib.facets.*;
import com.objectspace.voyager.*;
import java.util.*;

public class SupAgente

 {

 public static void main( String[] args )
  {
  try
   {
   // Inicialização do servidor voyager na porta 8000
   Voyager.startup( "8000" ); String tabClass = CompTable.class.getName();

   // Criação das tabelas para referências interna e externa

   Object[] args2 = new Object[]{"SupAgentPatras", "WWW", "WWW", "WWW", new Integer(0),
   new Integer(0), new Integer(0)};
   ICompTable tabela1= (ICompTable)Factory.create(tabClass,
   args2,"//150.163.20.7:8000/SuperAgentPatras");

    //Agente Supervisor instanciado

    System.out.println("Agente Patras e agentes externos inicializados: operando...");

   }
  catch( Exception exception )
   {
   System.err.println( exception );
   }
  }
 }
```

b) Agente móvel

Interface ITraderADD.class

```
package douto.tms_age;

public interface ITraderADD
```

```
{
   void go(String hello, Comp novo_comp, String no);
   void defineRota(String no);
   String data(String hello);
}
```

// FIM DA INTERFACE ITRADERADD

// Classe TraderADD.class

```
package douto.tms_age;

import java.io.*;
import java.util.*;
import java.lang.*;
import com.objectspace.lib.timer.*;
import com.objectspace.voyager.*;
import com.objectspace.voyager.message.*;
import com.objectspace.voyager.agent.*;
import com.objectspace.lib.facets.*;

public class TraderADD implements ITraderADD, Serializable
{
   String machines          = "none ";
   String patras_user       = " ";
   String pelion_user       = " ";
   String andros_user       = " ";
   String patras_time       = " ";
   String pelion_time       = " ";
   String andros_time       = " ";
   String patras_agent_local      = "//150.163.20.7:8000/SuperAgentPatras";
   String patras_agent_na_pelion  = "//150.163.20.19:8000/SuperAgentPatrasExt";
   String patras_agent_na_andros  = "//150.163.20.4:8000/SuperAgentPatrasExt";
   String pelion_agent_local      = "//150.163.20.19:8002/SuperAgentPelion";
   String pelion_agent_na_patras  = "//150.163.20.7:8002/SuperAgentPelionExt";
   String pelion_agent_na_andros  = "//150.163.20.4:8002/SuperAgentPelionExt";
   String andros_agent_local      = "//150.163.20.4:8001/SuperAgentAndros";
   String andros_agent_na_pelion  = "//150.163.20.19:8001/SuperAgentAndrosExt";
   String andros_agent_na_patras  = "//150.163.20.7:8001/SuperAgentAndrosExt";

   String patras_time_agent = " ";
   String pelion_time_agent = " ";
   String andros_time_agent = " ";
   String return_time       = " ";

   // definicao de variaveis para interface com os agentes supervisores (Tabelas)
   ICompTable ag_sup_patras_local;
   ICompTable ag_sup_patras_na_pelion;
   ICompTable ag_sup_patras_na_andros;

   public Comp comp_aux;// para armazenar o conteudo do novo componente a ser adicionado
   Stopwatch my_watch     = new Stopwatch();
```

```
public TraderADD()
{
 System.out.println( "Hello Supervisor Agent constructed.\n" );
}

public void finalize()
{
 String end = data("Agent TraderADD is finalized and data is destroyed at");
}

public String data(String hello)
{
   System.out.println("\n" + hello + ": ");
   GregorianCalendar cal = new GregorianCalendar();
   cal.setTimeZone (TimeZone.getTimeZone( "ECT+1"));
   String actual_date = "Local time:  "
         +cal.get (Calendar.HOUR_OF_DAY)
         +":"
         +cal.get (Calendar.MINUTE)
         +":"
         +cal.get (Calendar.SECOND)
         +":"
         +cal.get (Calendar.MILLISECOND)+"\n";
   System.out.println( actual_date );
   return actual_date;
}

public void write_file( String hello)
{
   FileWriter f1;
   try
   {
      f1 = new FileWriter("hello.message");
      f1.write(hello);
      f1.close();
   }
   catch (IOException exception )
   {
      String message = "Error while writing file: "+hello;
      System.out.println(message);
      try
      {
        IAlarm alarm = (IAlarm) Factory.create(IAlarm.class.getName());
        Object[] args = new Object[] {exception, message};
        OneWay.invoke( alarm, "alert", args);
      }
      catch (Exception exception2 )
      {
         System.err.println (exception2);
      }
   }
}
```

```
public void go(String hello, Comp novo_comp, String no)
{
 defineRota(no); //definindo as tabelas a serem atualizadas
 System.out.println( "Before action, starting StopWatch.\n");
 comp_aux = novo_comp;// Lembrar-se que para qualquer alteracao, deve se referir a
                      // comp_aux ao inves de comp
    my_watch.start();
    try
    {
     Agent.of(this).moveTo(ag_sup_patras_local, "atpatras");
    }
    catch ( Exception exception )
    {
      System.err.println( exception );
      System.out.println("Error moving to Patras. \n");
      Agent.of(this).setAutonomous(false);
    }
}

public void atpatras(ICompTable tabela )
{
 //Adicionar componente na tabela
 tabela.addComp(comp_aux);
 machines     =  "Patras.";
 patras_user  = System.getProperty("user.name");
 patras_time  = data("I am at Patras. \n");
 write_file("I was at Patras!");

 //move to pelion
 try
 {
   Agent.of(this).moveTo( ag_sup_patras_na_pelion, "atpelion" );
  }
 catch ( Exception exception )
 {
  System.err.println( exception );
  System.out.println("Error moving to Pelion. \n");
  try
     {
      IAlarm alarm = (IAlarm) Factory.create(IAlarm.class.getName());
      Object[] args = new Object[] {exception, "moving to Pelion"};
      OneWay.invoke( alarm, "alert", args);
      Agent.of(this).setAutonomous(false);
      }
  catch (Exception exception2 )
  {
   System.err.println (exception2);
  }
 }
}

public void atpelion(ICompTable tabela2)
```

```
{
 tabela2.addComp(comp_aux);
 machines     =  "Patras and Pelion.";
 pelion_user  = System.getProperty("user.name");
 pelion_time  = data("I am at Pelion. \n");
 write_file("I was at Pelion!");

    //move to andros
    try
    {
     Agent.of(this).moveTo( ag_sup_patras_na_andros, "atandros" );
    }
    catch ( Exception exception )
    {
     System.err.println( exception );
     System.out.println("Error moving to Andros. \n");
     try
       {
        IAlarm alarm = (IAlarm) Factory.create(IAlarm.class.getName());
        Object[] args = new Object[] {exception, "moving to Andros"};
        OneWay.invoke( alarm, "alert", args);
        Agent.of(this).setAutonomous(false);
        }
     catch (Exception exception2 )
        {
         System.err.println (exception2);
        }
    }
}

public void atandros(ICompTable tabela3)
{
 tabela3.addComp(comp_aux);
 machines     =  "Patras, Pelion and Andros.";
 andros_user  = System.getProperty("user.name");
 andros_time  = data("I am at Andros. \n");
 write_file("I was at Andros!");

 //move home
 try
    {
     Agent.of(this).moveTo( "tcp://150.163.20.7:9999", "athome" );
    }
 catch ( Exception exception )
 {
   System.err.println( exception );
   System.out.println("Error moving to" + Agent.of(this).getHome() +" \n ");
   try
     {
      IAlarm alarm = (IAlarm) Factory.create(IAlarm.class.getName());
      Object[] args = new Object[] {exception, "moving home"};
      OneWay.invoke( alarm, "alert", args);
      Agent.of(this).setAutonomous(false);
```

```
        }
       catch (Exception exception2 )
        {
         System.err.println (exception2);
        }
       }
    }

    public void athome()
    {
     //stop stopwatch and present collected data
     return_time = data("I am back at Patras, stopping Stopwatch. \n");
     my_watch.stop();
     System.out.println("I was at: " + machines + "\n");
     System.out.println("Mission time: "  +my_watch.getLapTime() +" Milliseconds. \n");
     System.out.println("At " +patras_time +" user "
                        + patras_user
                        +" was logged in at Patras and running the server.\n");
     System.out.println("At "+pelion_time +" user " +pelion_user
                        +" was logged in at Pelion and running the server.\n");
     System.out.println("At "+andros_time +" user " +andros_user
                        +" was logged in at Andros and running the server.\n");

     //allow garbage collection
     Agent.of(this).setAutonomous(false);
    }

    public void defineRota(String no)
    {
     if(no.equals("Patras"))
          {
            try
            {
              ag_sup_patras_local        =(ICompTable) Namespace.lookup(patras_agent_local);
              ag_sup_patras_na_pelion  =(ICompTable)
Namespace.lookup(patras_agent_na_pelion);
              ag_sup_patras_na_andros =(ICompTable)
                                          Namespace.lookup(patras_agent_na_andros);
            }
            catch( Exception exception )
            {
              System.err.println( exception );
              System.out.println("Possivel Erro: Patras agente supervisor fora do ar!!");
            }
          }

       if(no.equals("Pelion"))
          {
            try
            {
              ag_sup_patras_local        =(ICompTable) Namespace.lookup(pelion_agent_local);
              ag_sup_patras_na_pelion  =(ICompTable)
Namespace.lookup(pelion_agent_na_patras);
```

```
            ag_sup_patras_na_andros =(ICompTable)
                                        Namespace.lookup(pelion_agent_na_andros);
          }
          catch( Exception exception )
          {
            System.err.println( exception );
            System.out.println("Possivel Erro: Pelion agente supervisor fora do ar!!");
          }
        }
      if(no.equals("Andros"))
        {
          try
          {
           ag_sup_patras_local         =(ICompTable) Namespace.lookup(andros_agent_local);
           ag_sup_patras_na_pelion  =(ICompTable)
Namespace.lookup(andros_agent_na_pelion);
           ag_sup_patras_na_andros =(ICompTable)
                                        Namespace.lookup(andros_agent_na_patras);
          }
          catch( Exception exception )
          {
           System.err.println( exception );
           System.out.println("Possivel Erro: Andros agente supervisor fora do ar!!");
          }
        }
   }
}
```

//FIM DA CLASSE TRADERADD

c) Agente monitor

```
package douto.tms_age;

import com.objectspace.voyager.*;
import java.io.*;
import java.util.*;
import java.lang.*;

public class MonitAgente
{
   public static void main( String[] args )
   {
      Vector his1 = new Vector();
      IHistory historico;
      String name = args[0];
      String hist = args[1];
      int con_pa = 0;
      int con_pe = 0;
      int con_an = 0;
      boolean alterou = false;
```

```
try
{
   //startup as local client
   Voyager.startup();
   System.out.println( "Voyager started locally!\n");

   //linha adicionada apenas para inicializar o historico
   historico = (IHistory) Namespace.lookup( "//150.163.20.7:9990/HisPatras" );

   if(hist.equals("patras"))
   {
      System.out.println( "acessando historico Patras!\n");
      historico = (IHistory) Namespace.lookup( "//150.163.20.7:9990/HisPatras" );
   }

   if(hist.equals("pelion"))
   {
      System.out.println( "acessando historico Pelion!\n");
      historico = (IHistory) Namespace.lookup( "//150.163.20.19:9992/HisPelion" );
   }

  if(hist.equals("andros"))
   {
      System.out.println( "acessando historico Andros!\n");
      historico = (IHistory) Namespace.lookup( "//150.163.20.4:9991/HisAndros" );
   }

   //A variavel origem vai ter o valor da string hist proveniente
   // de um dos parametros args[] do inicio do programa
   // Desta forma, quando for inicializar o programa posso especificar
   // qual historico trabalhar e qual roteiro vai ser definido para atualizacao
   // das tabelas. NAO ESQUECER QUE QUANDO CRIO O AGENTE HISGENT, SEU
   // CONSTRUTOR
   // EXIGE A STRING ROTA PARA DEFINICAO DO ITINERARIO
   //String   origem   = System.getProperty("user.name");

  String origem = hist;
  Object[] arg = new Object[]{origem};

   for(;;)
   {
     System.out.println("launch 5 sec to verify: " + name);
     System.out.println("Work in: " + hist);
     try{ Thread.sleep( 20000 ); } catch( InterruptedException exception ) {}

     Line nova_line1;
     his1 = historico.getVector();
     int tam1 = his1.size();
     if(tam1 != 0)
     {
       con_pa = 0;
       con_pe = 0;
       con_an = 0;
```

```
            System.out.println("History: " + hist + " com: " + tam1 + " lines");
            for (int i = 0; i < tam1; i++)
            {
              nova_line1 = (Line) his1.elementAt(i);
              String na1 = nova_line1.getName();
              String nb1 = nova_line1.getCod_operacao();
              if((na1.equals(name)) && (nb1.equals("CONEXAO")))
              {
                alterou = true; //variavel true pois o agente deve ser lancado
                int    ta1 = nova_line1.getTime();
                String oa1 = nova_line1.getOrigem();
                System.out.println("Line" + " " + i + ": " + na1 + " -" + nb1 + " -" + ta1 + " -" +
                                   oa1);
                if((oa1.equals("PATRAS")))
                    con_pa++;
                if((oa1.equals("PELION")))
                    con_pe++;
                if((oa1.equals("ANDROS")))
                    con_an++;
                //apos atualizar conexao devemos eliminar linha do historico
                String eliminou_line = historico.removeLine(na1, nb1, ta1 );
                if(eliminou_line.equals("OK"))
                   System.out.println("Remocao de linha efetuada!");
                if(eliminou_line.equals("null"))
                   System.out.println("Remocao SEM EFEITO!");
              }
            }
            if(alterou)
            {
                IHisgent myagent = (IHisgent) Factory.create( Hisgent.class.getName(), arg);
                System.out.println( "I am leaving: \n");
                System.out.println( "Atualizar com: " + "(" + con_pa + " -" + con_pe + " -" +
                                    con_an + ")" );
                myagent.go(name, con_pa, con_pe, con_an);
            }
          }
        }
      }
      //error handling
      catch( Exception exception )
      {
         System.err.println( exception);
         System.out.println
              ("Failure before reaching the test-environment.\n");
      }

      // end of program
  }
}
```

A-5 - Serviço de balanceamento

a) Método balancear

```
public   void balanceamento()
  {
   String   origem   = System.getProperty("user.name");
   float tc=0;
   String endereco = "";
   String no_livre = "";
   float taxa_cpu = 0;
   float media_cpu;
   tabela_no menor_cpu = new tabela_no();
    tabela_no maior_cpu = new tabela_no();

   //buscar a maior taxa de cpu disponivel do sistema
    // menor_cpu = agente.busca_menor_cpu();
     maior_cpu = agente_sup.busca_maior_cpu();
     System.out.println ("maior_cpu "+ maior_cpu.nome);

   // buscara taxa de cpu disponivel da origem
     taxa_cpu = agente_sup.busca_tab_no(origem, tc);
      System.out.println ("taxa cpu" + taxa_cpu);

   // buscar a média de cpu disponível do sistema
   media_cpu = agente_sup.busca_media_cpu();
   System.out.println ("media taxa cpu" + media_cpu);

   if (taxa_cpu < media_cpu)
      { // if 01
       // Esperar um tempo aleatório
       //..
       // buscar a média de cpu disponível do sistema
       media_cpu = agente_sup.busca_media_cpu();

       if (taxa_cpu < media_cpu)
          {    // if 02
           // A primeira tentativa de balanceamento consiste em
           // migrar o objeto mais solicitado da máquina local para a
           // máquina remota

          // Obter o objeto que tem o maior número de conexões com o no origem

          tabela_con aux_con = new tabela_con();
          aux_con = agente_sup.busca_tab_con_patras( );
          System.out.println("aux_con.objeto "+aux_con.objeto);
```

```
 System.out.println("aux_con.origem "+aux_con.origem);

// replicar o objeto recuperado no no com o maior numero de conexões
// Verificar se nó origem não esta congestionado
if (agente_sup.busca_tab_no(aux_con.origem, tc) > media_cpu)
 {  // if 03
  // replicar o objeto recuperado no
   try
    {
     // Buscar endereco do no
     endereco= agente_sup.busca_endereco(aux_con.origem);
     System.out.println("enderco  ="+ endereco);

     if (aux_con.objeto.equals("tms_alc"))
       {    // if 04
        endereco = endereco + "tms_alc";
        System.out.println("enderco  ="+ endereco);
        tms_alc = (Ialcantara)Factory.create(alcantara.class.getName(),endereco);
       }

     if (aux_con.objeto.equals("tc_alc"))
       {endereco = endereco + "tc_alc";
       tc_alc1 = (Itc_alc)Factory.create(tc_alc.class.getName(),endereco);
       }

    } // fechar o try
  catch (Exception exception)
   {
    System.err.println( exception);
    }
  } // if 03
  else
  {  // if 03
  // Procurar nó com maior taxa descongestionamento
  no_livre = agente_sup.busca_no_livre();
   System.out.println("No livre " + no_livre);
   if (no_livre != null)
    {   // if 05
     // replicar o objeto recuperado no no LIVRE
     try
     {
      // Recuperacao do endereco

      if (aux_con.objeto.equals("tms_alc"))
        {
        endereco = no_livre + "tms_alc";
```

```
            tms_alc = (Ialcantara)Factory.create(alcantara.class.getName(),endereco);
            }

           if (aux_con.objeto.equals("tc_alc"))
            {endereco = no_livre + "tc_alc";
             tc_alc1 = (Itc_alc)Factory.create(tc_alc.class.getName(),endereco);
             }
           } // fechar o try
          catch (Exception exception)
          {
           System.err.println( exception);
          }
         } // if 05
         else
          System.out.println (".....SISTEMA SATURADO.....");

       } // else if 03
  } // fechar o if 02

  } // if 01


 } // fim do método balanceamneto
```

b) Método Realizar manutenção

```
public   void manutencao()
 {
  String   origem   = System.getProperty("user.name");
  float tc=0;
  String endereco = "";
  String no_livre = "";
  float taxa_cpu = 0;
  String[] vet_obj = new String[10];
   tabela_no maior_cpu = new tabela_no();

   tabela_no status = new tabela_no();
   // Recuperar status do nó origem
   String status_origem  =  agente_sup.busca_status_no(origem);

   if (status_origem.equals("suspend"))
     {

    //buscar  a maior taxa de cpu disponivel do sistema
```

```
  maior_cpu = agente_sup.busca_maior_cpu();
  System.out.println ("maior_cpu "+ maior_cpu.nome);

  // recuperar objetos instanciados no no origem
   boolean chave = false;
   int i=0;
   while (chave ==false )
    {
      if (agente_sup.tab_obj_patras[i] != null)
       {

        System.out.println ("objeto " + agente_sup.tab_obj_patras[i].nome);
         String objeto = agente_sup.tab_obj_patras[i].nome;
         i++;
       // criar o objeto recuperado no no com o maior numero de conexões
        try
         {
         // Buscar endereco do no com MAIOR CPU
         endereco= agente_sup.busca_endereco(maior_cpu.nome);
         System.out.println("enderco  ="+ endereco);

         if (objeto.equals("tms_alc"))
          {
          endereco = endereco + objeto;
             System.out.println("enderco  ="+ endereco);
          tms_alc = (Ialcantara)Factory.create(alcantara.class.getName(),endereco);
          }

         if (objeto.equals("tc_alc"))
         {endereco = endereco + objeto;
         tc_alc1 = (Itc_alc)Factory.create(tc_alc.class.getName(),endereco);
         }

         } // fechar o try
        catch (Exception exception)
         { //
          System.err.println( exception);
          } //
        } // if exixte objetos
        else
          {chave=true;}
       } // fim do while
    } // fim do sattus = suspend
} // fim do método manutencao
```

c) Método Remover nó

```
public   void remover_no(String no)
  {
   String   origem   = System.getProperty("user.name");
   float tc=0;

   String endereco = "";
   String no_livre = "";
   float taxa_cpu = 0;
   String[] vet_obj = new String[10];
    tabela_no maior_cpu = new tabela_no();
    System.out.println("no removido ="+ no);
    tabela_no status = new tabela_no();

    //buscar  a maior taxa de cpu disponivel do sistema

     maior_cpu = agente_sup.busca_maior_cpu();
     System.out.println ("maior_cpu "+ maior_cpu.nome);

        // Testar o nó eliminado

        if (no.equals("PATRAS"))
         {
          // recuperar objetos instanciados no no origem
          boolean chave = false;
          int i=0;
          while (chave ==false )
          {
           if (agente_sup.tab_obj_patras[i] != null)
            {
             System.out.println ("objeto " + agente_sup.tab_obj_patras[i].nome);
             String objeto = agente_sup.tab_obj_patras[i].nome;
             i++;
            // replicar o objeto recuperado no no com o maior numero de conexões
             try
             {
             // Buscar endereco do no com MAIOR CPU
             endereco= agente_sup.busca_endereco(maior_cpu.nome);
             System.out.println("enderco  ="+ endereco);

             if (objeto.equals("tms_alc"))
              {
                endereco = endereco + objeto;
                System.out.println("enderco  ="+ endereco);
                tms_alc = (Ialcantara)Factory.create(alcantara.class.getName(),endereco);
```

```
        }

       if (objeto.equals("tc_alc"))
         {endereco = endereco + objeto;
         tc_alc1 = (Itc_alc)Factory.create(tc_alc.class.getName(),endereco);
         }

       } // fechar o try
      catch (Exception exception)
       { //
        System.err.println( exception);
        } //
      } // if existe objetos
      else
        {chave=true;}
    } // fim do while
   } // no equuasl patras
  //} // if origem
 } // fim do método remover no
```

d) Interface com o usuário

O serviço de balanceamento disponibiliza para o usuário apenas o serviço “atender solicitação”. Toda solicitação de serviço do usuário ao software de controle de satélites é analisada por este método. O código abaixo ilustra a interface do serviço de balanceamento com o usuário e em seguida um exemplo da utilização deste método pelo usuário.

```
package douto.tms_age3;

   public interface Ibalance {


     public String atender_solicitacao(String obj);

     };
```

Exemplo de utilização do serviço

```
// copyright 1997, 1998 objectspace

package douto.tms_age3;
```

```
import java.io.*;
import java.io.RandomAccessFile;
import java.util.Vector;
import com.objectspace.lib.timer.*;
import com.objectspace.lib.facets.*;
import com.objectspace.voyager.*;
import com.objectspace.voyager.corba.*;

public class Client1
  {
   static  Ialcantara alc;
   static  Ibalance balance;
   .
   .
   .
  public static void main( String[] args )
    {
     Vector tab_aux = new Vector();
    try
      {
      Voyager.startup();

      // Conexão com o serviço de balanceamento
      balance  = (Ibalance)Namespace.lookup("//150.163.20.7:8400/balance" );


// Recuperar a origem da conexào

      String   origem   = System.getProperty("user.name");

      Stopwatch watch = new Stopwatch();
      watch.start();
      //
      //Solicitar o serviço de balanceamneto
      String endereco = balance.atender_solicitacao("tms_alc");
       System.out.println("endereco retornado "+ endereco);

       alc  = (Ialcantara)Namespace.lookup(endereco );


     System.out.println( "frame = " + alc.get_frame());
      watch.stop();
      System.out.println( "time = " + watch.getTotalTime() + "ms" );


      }
```

```
   catch( Exception exception )
    {
    System.err.println( exception );
    }

   Voyager.shutdown();
   }
 }
```

A- 6 : Serviço de Persistência

a) Método bind

```
static org.omg.CORBA.Object bind(ORB orb,
                          String interface_name,
                          String object_name) {
    // try to resolve the interface with the interface repository
    org.omg.CORBA.Repository ir =
org.omg.CORBA.RepositoryHelper.bind(orb);
    org.omg.CORBA.InterfaceDef iDef =

org.omg.CORBA.InterfaceDefHelper.narrow(ir.lookup(interface_name));
    if(iDef == null) {
      throw new BAD_OPERATION("Interface not found: " +
interface_name);
    }
    String repository_id = iDef.id();
    if(object_name.equals("null")) {
      object_name = null;
    }
    return orb.bind(repository_id, object_name, null, null);
  }
```

b) Método Analisar parâmetros

```
static Any Analisar_param(String paramName,
                      TypeCode paramType,
                      Enumeration expr) {
    try {
      Any any = _orb.create_any();
      String string = next(expr, "Expected argument: " + paramName);
      TCKind kind = paramType.kind();
      if(kind == TCKind.tk_string) {
      any.insert_string(string);
      }
      else if(kind == TCKind.tk_long) {
      any.insert_long(new Integer(string).intValue());
      }
      else {
      throw new ParserException("Not supporting type: " + paramType);
      }
```

```
      return any;
    }
    catch(Exception e) {
      throw new ParserException(e.toString());
    }
  }
```

c) Método Request

```
 static void Request(String operation, Enumeration expr) throws IOException {
    String reason = "Invalid expression: (<operation> <type> <name> <args...>)";
       InterfaceDef iDef = object._get_interface();
    OperationDef opDef = OperationDefHelper.narrow(iDef.lookup(operation));
    if(opDef == null) {
      throw new ParserException("Operation not found: " + operation);
    }
    Request request = object._request(operation);
    ParameterDescription[] parDescs = opDef.params();
    for(int i = 0; i < parDescs.length; i++) {
      ParameterDescription parDesc = parDescs[i];
      Any param;
      if(parDesc.mode == ParameterMode.PARAM_OUT) {
          param = _orb.create_any();
          param.type(parDesc.type);
      }
      else {
          param = parseParameter(parDesc.name, parDesc.type, expr);
      }
     request.arguments().add_value(parDesc.name, param,
parModeToArgMode(parDesc.mode));
```

```
  }
  request.set_return_type(opDef.result());
  _out.println("\tbefore:\t" + request.arguments());
  ExceptionDef[] exDefs = opDef.exceptions()
  for(int i = 0; i < exDefs.length; i++) {
    request.exceptions().add(exDefs[i].type());
  }
  if(opDef.mode() == OperationMode.OP_ONEWAY) {
    request.send_oneway();
    return;
  }
  request.invoke();
  Exception exception = request.env().exception();
  if(exception != null) {
    _out.println("\texception:\t" + exception);
    return;
  }
  _out.println("\tafter:\t" + request.arguments());
  _out.println("\tresult:\t" + request.result().value());
}
```