MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-10052-TDI/888

ABORDAGENS DE OBJETOS DISTRIBUÍDOS APLICADAS AO SIMULADOR DE SATÉLITES DO INPE

Luciana Akemi Burgareli

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Maurício Gonçalves Vieira Ferreira, aprovada em 13 de junho de 2003.

INPE
São José dos Campos
2003

681.3.06

BURGARELI, L. A.
Abordagens de objetos distribuídos aplicadas ao simulador de satélites do INPE / L. A. Burgareli. – São José dos Campos: INPE, 2003.
155p. – (INPE-10052-TDI/888).

1.Disponibilidade. 2.flexibilidade. 3.Tolerância a falhas. 4.Processamento distribuído. 5.Objetos distribuídos. I.Título.

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

Dr. Solon Venâncio de Carvalho
Presidente/INPE, SJCampos-SP

Dr. Maurício Gonçalves Vieira Ferreira
Orientador/INPE, SJCampos-SP

Dr. Nilson Sant'Anna
Membro da Banca/INPE, SJCampos-SP

Dr. João Bosco Schumann Cunha
Membro da Banca/UNIFEI, Itajubá-MG

Drª Selma Shin Shimizu Melnikoff
Membro da Banca
Convidada Escola Politécnica da USP
São Paulo-SP

Candidata: Luciana Akemi Burgareli

São José dos Campos, 13 de junho de 2003.

Dedico este trabalho ao mestre e amigo
João Lino Filho.

AGRADECIMENTOS

Agradeço a Deus por me conceder esta oportunidade.

Aos meus pais Jurandir e Tomoco pela dedicação, apoio e incentivo.
Por acreditarem nos meus sonhos. A eles, todo meu amor e gratidão.

Ao meu marido Fábio pelo amor, respeito e compreensão.

Ao meu orientador Maurício pela dedicação e paciência.
Por encorajar-me nos momentos de incerteza e desânimo.
Por auxiliar-me em todas as etapas deste trabalho, contribuindo para que o mesmo se concretizasse.

RESUMO

Na tentativa de melhor aproveitar as características da distribuição de objetos, técnicas de como melhor distribuir os objetos são apresentadas e aplicadas aos subsistemas do Simulador de Satélites do INPE. A computação distribuída vem rapidamente ganhando espaço no campo da informática, entretanto, existem poucas técnicas a fim de projetar e modelar os sistemas de objetos distribuídos. É observada uma deficiência no aspecto de como modelar corretamente esta distribuição, ou seja, como melhor distribuir os objetos de acordo com as características da aplicação e necessidades do usuário, explorando-se mais ativamente as vantagens desta tecnologia e alcançando-se assim, maior qualidade. Com o objetivo de melhorar a organização da distribuição dos objetos, são apresentadas técnicas de modelagens baseadas na análise do comportamento dos objetos, na aplicação e nas necessidades dos usuários.

AN APPROACH FOR DISTRIBUTED OBJECTS APPLIED TO SATELLITE SIMULATOR

ABSTRACT

The purpose of this paper is to present a study of a distributed object application to the Satellite Simulator, present in INPE. The distributed computing is rapidly gaining more importance in the computer field, however, there are a few tools available to design and model the distributed object system. A deficiency is noted in the process of correctly modeling this distribution, that is, how to improve the distribution according to the application characteristics and the user's necessity, exploiting more actively this technology advantages and obtaining a better quality this way. Therefore, besides exploring characteristics of object distribution, the intention is to build up an analysis through the object distribution modeling, noting the user needs and aspects of the application. For this purpose, some techniques that will establish criteria and different methods of modeling will be proposed.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

CBERS	-	Satélite Sino-Brasileiro de Recursos Terrestres
CCS	-	Centro de Controle de Satélites
CORBA	-	Common Object Request Broker Architecture
CRC	-	Centro de Rastreio e Controle de Satélites
DCOM	-	Distributed Component Object Model
ICMP	-	Internet Control Message Protocol
INPE	-	Instituto Nacional de Pesquisas Espaciais
JVM	-	Java Virtual Machine
MECB	-	Missão Espacial Completa Brasileira
OMG	-	Object Management Group
ORB	-	Object Request Broker
RMI	-	Remote Method Invocation
RPC	-	Remote Procedure Call
SCD1	-	Satélite de Coleta de Dados 1
SICS	-	Sistema de Controle de Satélites
SICSD	-	Sistema de Controle de Satélites Distribuído
STC	-	Simulador Telecomando
STMS	-	Simulador Telemetria
TC	-	Telecomando
TMS	-	Telemetria
UML	-	Unified Modeling Language

CAPÍTULO 1

INTRODUÇÃO

O contínuo desenvolvimento da tecnologia espacial no Brasil vem contribuindo para que a sociedade brasileira possa usufruir os benefícios propiciados por este setor. Neste contexto, iniciou-se em 1978, no Instituto Nacional de Pesquisas Espaciais (INPE), a Missão Espacial Completa Brasileira (MECB), onde um dos seus objetivos era desenvolver uma família de pequenos satélites, visando fornecer ao país um sistema de coleta de dados ambientais direcionados às necessidades brasileiras.

A década de 90 foi marcada pelos primeiros resultados da MECB. Em 1993, foi colocado em órbita o primeiro satélite brasileiro, o Satélite de Coleta de Dados (SCD-1) 1, demonstrando a capacidade brasileira no desenvolvimento e operação de sistemas espaciais. Em 1998, o SCD-2 também foi lançado com sucesso, operando com melhor desempenho do que o primeiro, devido às inovações tecnológicas. O Satélite Sino-brasileiro de Recursos Terrestres (CBERS-1), fruto da cooperação entre os governos Brasileiro e Chinês, foi lançado em 1999 (INPE, 2003).

Para possibilitar ao INPE realizar todas as operações necessárias ao controle e utilização destes satélites, o Instituto utiliza-se do Centro de Controle de Satélites (CRC), composto pelo sistema de controle de solo e sua estrutura funcional. Vários subsistemas compõem o complexo computacional de sistema de solo, entre eles, o software aplicativo denominado Sistema de Controle de Satélites (SICS), projetado especialmente para controlar os satélites desenvolvidos pelo INPE.

O Software Simulador de Satélites é um subsistema também presente no sistema de controle de solo do INPE, uma ferramenta que permite criar um ambiente operacional realístico, reproduzindo com fidelidade cada fase prevista

de vida útil do satélite (Rozenfeld et al.,1990), auxiliando no desenvolvimento e validação de procedimentos de operação e controle do mesmo.

Com intuito de apresentar uma nova abordagem para o software de controle de satélite, atualmente centralizado, foi considerada recentemente, como objeto de pesquisa, uma nova tecnologia para o SICS, o Sistema de Controle de Satélites Distribuído (SICSD), explorando os recursos da distribuição, aplicando-lhe uma arquitetura flexível e dinâmica para objetos distribuídos.

Constata-se que uma nova tendência, voltada às aplicações baseadas em sistemas distribuídos, está surgindo em conseqüência do aumento significativo da utilização dos serviços de rede e das aplicações disponíveis para a Internet (Ahuja et al., 2000).

Nos últimos anos, a indústria do software fez com que a comunidade da informática assistisse a substituição dos sistemas centralizados por ambientes distribuídos (Butler, 1995). Promessas por melhor desempenho e melhores custos na implantação motivaram ainda mais esta tendência, o que reforçou a idéia da distribuição como uma manifestação inevitável e inovadora.

Os objetos distribuídos oferecem uma série de atrativos como transparência, integridade dos dados, tolerância a falhas, disponibilidade, recuperabilidade, autonomia dos objetos e concorrência no processamento (Chin et al., 1991). Por outro lado, este paradigma também apresenta algumas dificuldades como, por exemplo, a necessidade de milhares de linhas de código para compor sua linguagem complexa; ou ainda, a exigência de uma seleção criteriosa para o padrão de distribuição a ser empregado (Sessions, 1996).

O objetivo deste trabalho de mestrado é realizar um estudo sobre como modelar objetos distribuídos, apresentando técnicas que estabelecem critérios para realizar a distribuição de objetos. Apresenta-se as características de cada técnica de distribuição, observando vantagens e desvantagens de cada uma, obtidas num estudo comparativo, que envolve desempenho e disponibilidade.

O estudo de caso considerado é o Software Simulador de Satélites. Assim, aplica-se, ao Software Simulador de Satélites, uma abordagem para objetos distribuídos.

1.1 - MOTIVAÇÃO

Nos últimos anos, empresas desenvolvedoras de software vêm lançando no mercado, vários novos padrões a serem aplicados aos sistemas de objetos distribuídos. Com relação à distribuição de objetos, estes padrões se tornam, a cada ano, mais abrangentes e eficientes. Porém, é observada uma carência no aspecto de como conduzir corretamente esta distribuição, ou seja, como melhor distribuir os objetos de acordo com as características da aplicação e necessidades do usuário, explorando-se mais ativamente as vantagens desta tecnologia e alcançando-se maior qualidade. Assim, a computação distribuída vem rapidamente ganhando espaço no campo da informática, entretanto, existem poucas técnicas a fim de projetar e modelar os sistemas de objetos distribuídos.

Em geral, a maioria dos autores apenas se preocupa em utilizar as características dos objetos distribuídos, ou ainda oferecer serviços relacionados com a distribuição, como pode ser evidenciado em (Zhou et al., 1999), (Butler, 1995), (Chen, 1998), (Emmerich et al., 2000), (Kalogeraki et al., 1999), (Purao et al., 1998) e (Katsaros et al., 2002).

Alguns trabalhos que estão sendo realizados na área de objetos distribuídos possuem um objetivo comum em utilizar este paradigma de forma a minimizar problemas das aplicações, melhorando o desempenho e a disponibilidade de seus projetos.

Outros trabalhos apresentam estudos sobre ferramentas de distribuição, como pode ser verificado na conceituada obra de Tanenbaum (Tanenbaum et al.,

2002), que constitui um respeitado trabalho sobre sistemas distribuídos. Neste trabalho são abordados muitos aspectos relevantes dos sistemas distribuídos desde definições, características, etc. No Capítulo 9 desta referência, encontra-se um detalhamento minucioso sobre as muitas ferramentas utilizadas para a distribuição. Entretanto, não são citados critérios para realizar esta distribuição.

Porém, ainda que de forma pouco expressiva, nota-se que os autores começam a perceber que o modo como a distribuição é manipulada é provavelmente um importante aspecto no contexto de objetos distribuídos, já que a distribuição afeta diretamente o desempenho e a disponibilidade do sistema (Zhou et al., 1999). Estes autores ainda deixam claro em suas pesquisas que, para se conseguir obter um bom projeto, modelos são essenciais (Butler, 1995).

No ambiente distribuído, algumas preocupações são constantes: dificuldades são encontradas para garantir que os serviços estejam sempre disponíveis ou sempre eficientes. Além disso, os objetos nos sistemas distribuídos podem, fatalmente, apresentar falhas. As possibilidades destas falhas aumentam à medida em que se aumenta a dependência entre objetos nos sistemas distribuídos, já que, o número de aplicações que requer tolerância a falhas também aumenta. Neste contexto, observa-se a importância em se obter um estudo focando a modelagem da distribuição, ou seja, uma tentativa de se estabelecer técnicas de como melhor distribuir os objetos com intuito de aumentar a disponibilidade e melhorar o desempenho do sistema.

Diante deste fato, apresenta-se um estudo, modelando a distribuição de objetos em diferentes aspectos. Estas técnicas de modelagens são empregadas na distribuição dos objetos do Software Simulador de Satélites, a fim de se acentuar os benefícios fornecidos pela distribuição.

Um outro fator motivador para a realização deste trabalho, está na possibilidade de se minimizar alguns problemas, advindos do ambiente centralizado em que atualmente se encontra o Software Simulador. Percebe-se

que muitos projetos exigem características indispensáveis ao seu perfeito funcionamento. Estas necessidades, uma vez não satisfeitas, acabam por impor obstáculos ao desempenho de todo o processo. Algumas destas necessidades são citadas a seguir:

- A elevação da **disponibilidade** dos serviços é um fator desejável em todos os projetos. A indisponibilidade de um subsistema pode interferir no funcionamento de um outro subsistema do projeto.

- A obtenção de **Tolerância a Falhas** se torna uma exigência nos sistemas atuais, já que uma falha ocorrida pode por em risco todo o funcionamento do sistema. Em um ambiente distribuído, o funcionamento de um sistema como um todo, se encontra mais vulnerável, devido às alterações da rede e de outras máquinas que participam do sistema. Portanto, assegurar tolerância a falhas é indispensável.

- **Desempenho** é um fator de extrema importância nos projetos atuais. O avanço tecnológico cobra a existência desta característica como um fator fundamental.

- A **flexibilidade** deve ser também considerada para atender a situações, como por exemplo, uma maior demanda da utilização de serviços, ou ainda, um acréscimo no número de usuários.

O Software Simulador de Satélites, apresenta algumas limitações que podem ser melhoradas com o emprego das características citadas anteriormente. Desta forma, percebe-se a possibilidade de minimizar estes problemas, empregando os recursos da tecnologia de objetos distribuídos ao Simulador de Satélites.

1.2 - ORGANIZAÇÃO DO TRABALHO

São apresentados, no Capítulo 2, os conceitos necessários para a compreensão deste trabalho. Detalham-se conceitos de distribuição, assim como os padrões de distribuição, como as tecnologias: CORBA, DCOM e Java RMI. O estado da arte também é tratado neste capítulo, onde pode ser observada uma série de trabalhos relacionados à distribuição. O objetivo é prover informações para que se possa realizar uma análise de como autores estão conduzindo seus trabalhos relacionados à distribuição.

No Capítulo 3, o trabalho proposto é detalhado. Apresenta-se o SICSD, o Software Simulador de Satélites, e as propostas de alternativas para se modelar a distribuição dos objetos. Detalha-se também como se pretende minimizar algumas limitações do Software Simulador de Satélites.

Já no Capítulo 4, observa-se o desenvolvimento do projeto proposto. Pode-se verificar a modelagem utilizada e a especificação dos requisitos do Software Simulador. Explica-se o funcionamento do sistema proposto, detalhando seus serviços e funcionalidades.

O Capítulo 5 é responsável por elucidar a implementação do trabalho proposto. Após detalhar o ambiente de desenvolvimento, as partes mais importantes do código são apresentadas, juntamente com os diagramas da UML.

No Capítulo 6, testes e resultados podem ser apreciados para que se possam confirmar as conclusões que são apresentadas logo a seguir, no Capítulo 7.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo destina-se a apresentar conceitos importantes das tecnologias utilizadas no desenvolvimento deste trabalho.

Vários estudos e sistemas estão sendo desenvolvidos baseados em objetos distribuídos, utilizando ferramentas que seguem a especificação *Common Object Request Broker Architecture* (CORBA) da *Object Management Group*, (OMG) ou produtos como o *Distributed Component Object Model* da Microsoft (DCOM) e o *Remote Method Invocation* da Sun MicroSystem (RMI).

Desta forma, apresenta-se um resumo destas tecnologias, dando maior ênfase ao padrão RMI, padrão escolhido para o desenvolvimento do presente trabalho. O capítulo apresenta também trabalhos relacionados à distribuição de objetos.

2.1 - ORIENTAÇÃO A OBJETOS

Atualmente, em um ambiente corporativo, a utilização do paradigma Cliente/Servidor se torna cada vez mais indispensável para as aplicações modernas. Entretanto, com a evolução da informática, observam-se características computacionais bem diferentes das utilizadas no passado. O cliente, hoje, possui processador com maior capacidade de processamento, provendo total autonomia ao usuário, podendo atender perfeitamente as solicitações de um segundo cliente sempre que necessário, agindo, desta forma, como um servidor. Por outro lado, o servidor, para prover resultados a uma determinada chamada do cliente, pode buscar informações em outros servidores da rede, representando assim, um cliente. Portanto, observa-se que a filosofia Cliente/Servidor presente neste ambiente é marcada pelo fato de clientes e servidores não assumirem papéis únicos, ou seja, todos os

computadores da rede podem tornar-se clientes e servidores, dependendo da aplicação (Mainetti,1997).

O ambiente descrito acima é denominado de Sistema Distribuído, onde duas ou mais máquinas interligadas cooperam entre si, sem memória compartilhada, e executam tarefas em paralelo, oferecendo um melhor desempenho às aplicações, e aparentando a seus usuários um único computador, proporcionando, desta forma, um ambiente cuja distribuição física poderá ser transparente ao usuário final.

Sistemas distribuídos, integração de máquinas em diferentes arquiteturas e sistemas operacionais, Internet e outras tecnologias modernas são realidades evidentes na informática atual. Para se desenvolver softwares em novos ambientes como estes, uma tecnologia muito empregada é a Orientação a Objetos (Mainetti,1997).

A orientação a objetos vem se consagrando na computação moderna, onde o software é organizado como uma coleção de objetos, nos quais armazenam-se seus dados e a estrutura lógica que dita seu comportamento.

A união destas duas tecnologias: Sistemas Distribuídos e Orientação a Objetos origina a área de Objetos Distribuídos.

2.2 - OBJETOS DISTRIBUÍDOS

As aplicações distribuídas vêm aumentando sua complexidade a cada dia, gerando uma tendência inevitável rumo a sistemas de objetos distribuídos. Sistemas de objetos distribuídos são sistemas distribuídos onde todas as entidades são modeladas como objetos. Estes sistemas utilizam os conceitos e benefícios já consagrados da orientação a objetos (Saleh et al., 1999).

Objetos distribuídos são programas independentes que podem estar localizados em qualquer nó de uma rede, sendo acessados por clientes remotos via invocação de métodos. Os clientes não precisam conhecer onde eles residem, isto é, se estão localizados na mesma máquina do cliente ou não. (Orfali et al., 1996). Esta habilidade apresentada, de construir um objeto em um nó e poder interagir como este, em outro nó, traz entre outras, a promessa de prover flexibilidade e alta disponibilidade. Por isso, muitas organizações têm se voltado para sistemas distribuídos e orientação a objetos com intuito de obter da distribuição, benefícios acoplados com a previsão de um desempenho eficiente (Purao et al., 1998).

Objeto distribuído é uma evolução do objeto convencional e possui uma interface específica onde os compiladores geram um código especial para que estes objetos possam ser acessados por outros objetos locais ou remotos, de maneira que o programa que o solicite desconheça o local onde o objeto chamado está instanciado, detalhes de implementação do mesmo ou o sistema operacional que estiver sendo utilizado.

As características listadas a seguir, comprovam a importância da tecnologia de objetos distribuídos (Chin et al;1991):

- Distribuição: o sistema é executado numa rede de computadores independentes e heterogêneos.
- Transparência: o sistema não apresenta detalhes de implementação e localização dos objetos.
- Tolerância a Falhas: a falha de um objeto representa apenas uma falha parcial no sistema
- Disponibilidade: o sistema assegura disponibilidade de seus serviços independente de falhas nos computadores.

- Recuperabilidade: numa situação de falha, é possível recuperar automaticamente objetos persistentes residentes no computador.

- 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 servidor pode atender a múltiplas invocações de clientes concorrentemente.

- Melhor desempenho: a sobrecarga de uma máquina pode ser eliminada através da migração de objetos mais solicitados para outras máquinas.

2.3 - MIDDLEWARE

Em um sistema formado por objetos distribuídos, um objeto cliente pode solicitar um outro objeto localizado no mesmo computador do sistema em execução, ou em qualquer estação de uma rede, entretanto, ele desconhece detalhes de localização e implementação uns dos outros. Isto é possível através da tecnologia *middleware*.

Middleware é um software de conectividade que consiste em um conjunto de serviços que permite interação, através da rede, de múltiplos processos executando em uma ou mais máquinas (Costa, 2000). Esse software de conectividade se localiza entre a aplicação e o sistema operacional (Bernstein, 1996).

Assim, *middleware* pode ser entendido como a ponte necessária para comunicação entre componentes distribuídos. Esta ponte é constituída de padrões, protocolos, tecnologias desenvolvidas cada uma com um grau de abrangência.

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 as funções devem ser colocadas no lado cliente e quais devem estar 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).

Dentre os principais padrões de *middlewares* existentes no mercado, pode-se citar:

- CORBA;
- DCOM e
- RMI.

2.4- CONSIDERAÇÕES

Atualmente percebe-se um aumento significativo em aplicações baseadas em RMI como soluções para o desenvolvimento de aplicações distribuídas, devido o modelo ser caracterizado por uma notável facilidade de manutenção e extensão.

Diferente de outras programações distribuídas, Java RMI é uma linguagem específica. Em um primeiro ponto de análise, o fato de não precisar se preocupar com uma arquitetura de forma a suportar múltiplas linguagens, possibilita RMI ter a habilidade de fornecer mais avanços em suas próprias características como serialização, segurança, tornando seu padrão cada vez mais compreensível.

Por um segundo aspecto, a linguagem singular e a simplicidade encontrada em sua utilização, afastam o padrão RMI de toda a infra-estrutura projetada por outras interfaces presentes na computação distribuída.

Para o presente trabalho, as características fornecidas pelo padrão RMI foram classificadas, através de estudos, como suficientes para o desenvolvimento do mesmo. Este fato, aliado as facilidades apresentadas pelo padrão, surgiram como um ponto decisivo para optar pela utilização do RMI.

Observa-se que, por não apresentar todos os serviços que outros padrões oferecem, o padrão RMI não pode ser tido como uma proposta global para aplicações distribuídas em geral, mas é uma tecnologia promissora que atende muitas necessidades da computação distribuída.

A seguir, são apresentadas, resumidamente, todas as tecnologias citadas. Detalha-se, mais profundamente, o RMI, já que o mesmo se encontra como padrão selecionado para o desenvolvimento deste trabalho.

2.5 - CORBA

A plataforma CORBA é a especificação de um *middleware* (Bernstein, 1996) orientado a objetos desenvolvido pela OMG (Pereira, 2002).

CORBA é baseado em orientação a objetos e no modelo de computação distribuída cliente Servidor (Ferreira, 2001).

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 objeto cliente e objeto servidor (Mowbray et al., 1997).

O ORB forma o núcleo do sistema distribuído CORBA e é responsável por habilitar a comunicação entre os objetos, enquanto esconde detalhes relacionado à distribuição e heterogeneidade (Tanenbaum,2002).

O padrão CORBA ainda apresenta serviços como de persistência de Objetos, identificação de objetos, segurança e etc. Os 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. (Costa, 2000). 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. Os serviços CORBA são fundamentais e globalmente aceitos, sendo úteis a todos os tipos de aplicação e independentes do domínio da aplicação (Ferreira, 2001).

2.6 - DCOM

DCOM é uma extensão do *Component Object Model* (COM), ou seja, a base de DCOM é formada por uma tecnologia de componente de objeto da Microsoft denominada COM (Tanenbaum,2002). O DCOM permite que o modelo de objeto definido pelo COM possa ser utilizado para distribuição de objetos. Assim, DCOM é o COM distribuído através da rede (Costa, 2000).

O objetivo do COM é suportar o desenvolvimento de componentes que podem ser dinamicamente ativados e que podem interagir com outros objetos.

Desta forma, 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 (Ferreira, 2001). Um componente COM é um código executável contendo uma biblioteca dinâmica. (Tanenbaum,2002).

DCOM, como já citado, baseia-se no modelo de objeto definido pelo COM, acrescentando a ele três novos elementos (Chappell, 1996, Grimes, 1997):

- Técnicas para criação de um objeto remoto.
- Protocolo para invocação dos métodos de um objeto remoto.
- Mecanismos para assegurar um acesso seguro a um objeto remoto.

2.7 - RMI

O RMI pode ser visto como um conjunto de classes e interfaces em Java que encapsulam vários mecanismos de troca de dados, a fim de simplificar a execução da chamada a métodos remotamente localizados em uma outra (JVM) *Java Virtual Machine*. O RMI permite que objetos Java executando no mesmo computador ou em computadores separados se comuniquem entre si, via chamadas de método remoto (Deitel et al., 2001).

O funcionamento de RMI consiste basicamente em dois programas: um cliente e um servidor. Uma típica aplicação servidora instancia objetos remotos, faz referência para estes acessíveis objetos remotos e esperam pelos clientes que invoquem seus métodos. Uma típica aplicação cliente consegue a referência para um ou mais objetos remotos no servidor e então invoca os métodos dos mesmos (Sun Microsystems, 2003). Cada objeto remoto implementa uma interface remota que especifica quais de seus métodos podem ser invocados pelos clientes.

Para que o cliente possa invocar métodos de um objeto remoto, ele deve obter primeiro a referência deste objeto. Essa referência é passada ao servidor que

localizará a implementação do objeto. O RMI trata os detalhes de comunicação entre cliente e servidor: o cliente vê a invocação de um método de um objeto remoto como a de um objeto local.

RMI fornece um servidor de nomes (*naming service*) para localizar objetos remotos, um carregador de classes (*class loader*) para carregar *stubs* no servidor. RMI provê ainda alguns recursos extras de segurança para garantir o bom comportamento do sistema.

Objetos distribuídos podem falhar pelos mais variados motivos, motivos estes ainda mais variados do que nos objetos locais. Existe então, a capacidade de tratar exceções que possam ocorrer durante a chamada de um método remoto. Para isso, RMI estendeu as características de tratamento de exceções em Java, criando novas classes de exceções, facilitando a captura de falhas com objetos remotos.

2.7.1 - Interfaces, Objetos e Métodos Remotos

Como qualquer outra aplicação, uma aplicação distribuída utilizando Java RMI é constituída de interfaces e classes. As interfaces definem os métodos, e as classes implementam os métodos definidos na interface e, também podem definir métodos adicionais.

Em uma aplicação distribuída, algumas das implementações são assumidas para residirem em máquinas virtuais diferentes. Objetos que possuem métodos que podem ser chamados por máquinas virtuais são objetos remotos.

No RMI, um objeto torna-se remoto pela implementação de uma interface remota, com as seguintes características (Sun Microsystems, 2003):

- Uma interface remota estende a interface *java.rmi.Remote*.

- Cada método da interface declara *Java.Rmi.RemoteException* para exceções específicas.

2.7.2 - Marshaling e Unmarshaling

Para que objetos diferentes possam se comunicar, a formatação de parâmetros e valores de retorno devem ser conhecidas pelo cliente para que este possa fazer a correta invocação de métodos. No caso de uma comunicação remota, o cliente deve ainda estar ciente de como máquinas diferentes representam tipos como: caracteres, *floats*, inteiros, etc.

É necessário, desta forma, além de um formato padrão para transmissão, uma operação que realize as transformações necessárias tanto no cliente quanto no servidor. A operação de empacotar parâmetros, num formato padrão, para transmissão é denominada *marshaling* (ordenação) e a operação inversa, de desempacotamento de parâmetros do formato padrão para um formato apropriado à recepção é denominado de *unmarshaling* (desordenação).

2.7.3 - As Camadas da Arquitetura RMI

- Para garantir a compatibilidade com os programas e implementações Java existentes e fazer o processo tão transparente para o programador quanto possível, a comunicação entre um cliente e um servidor é implementada em uma série de camadas independentes entre si, como ilustra a Figura 2.1 (Costa, 2000):

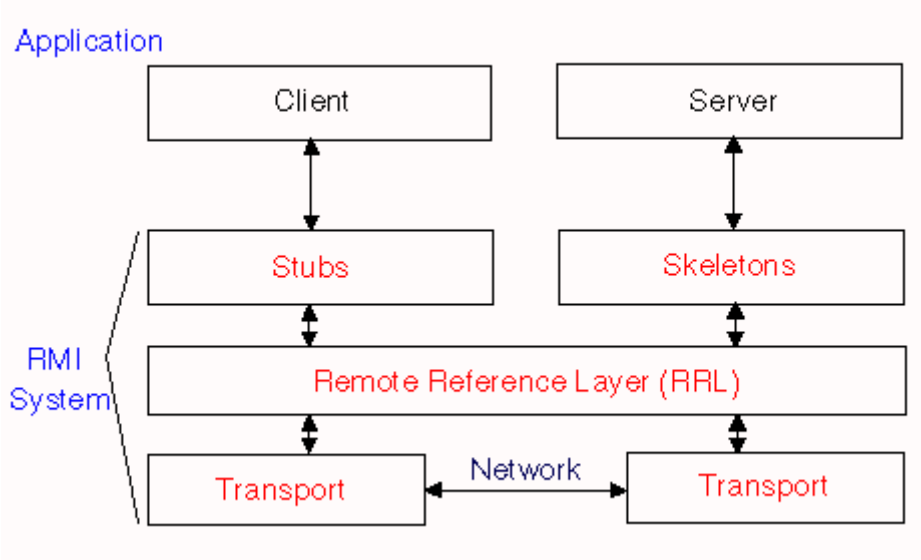

FIGURA 2.1 – Arquitetura RMI.

FONTE: Matthews (1998).

- Camada *Stub/Skeleton*;
- Camada de Referência Remota e
- Camada de Transporte.

2.7.3.1 - Camada Stub/Skeleton

Esta camada é responsável por gerenciar a interface de objeto remoto entre o cliente e o servidor, ou seja, é a interface que as aplicações clientes e servidoras utilizam para interagirem uma com a outra (Matthews, 1998).

RMI utiliza-se de um mecanismo padrão, empregado em sistemas (RPC) *Remote Procedure Call,* para comunicação com objetos remotos, representados por códigos nos pontos finais da conexão para deixar transparente o uso do RMI pelo usuário: no lado cliente, um *stub* e no lado servidor, *skeletons*.

Stubs:

Um *stub* para um objeto remoto age como uma representação de um cliente. Quando o objeto local invoca um método num objeto remoto, o *stub* fica responsável por enviar a chamada ao método para objeto remoto.

Quando um método *stub* é invocado, suas funções são (Costa, 2000):

- Inicializar a conexão com a JVM remota contendo o objeto remoto (acionando a camada de referência remota),
- Escrever, fazendo o *marshaling* e transmitir os parâmetros para a camada de referência remota,
- Informar a camada de referência remota que a chamada pode ser invocada,
- Esperar pelo resultado da invocação do método,
- Ler, fazendo o *unmarshaling* do valor ou exceção retornada,
- Retornar o valor para o objeto que executou a chamada.
- Informar a camada de referência remota que a chamada foi completada.

Com o objetivo de simplificar o mecanismo de realização da invocação, o *stub* esconde a serialização dos parâmetros e toda a comunicação no nível de rede presente.

Skeletons:

Na JVM remota, cada objeto remoto deve ter um *skeleton* correspondente. Os *skeletons* são responsáveis por enviar a chamada para a implementação do objeto remoto.

Quando um *skeleton* recebe a chegada de um método de invocação, fica responsável pelos seguintes passos (Costa, 2000):

- Leitura (fazendo o *unmarshaling*) dos parâmetros recebidos,

- Invocação do método na implementação do objeto remoto que executará o serviço requerido pelo cliente,

- Escrita (fazendo o *marshaling*) e transmite o resultado (retorna valor ou exceção) ao objeto que executou a chamada.

2.7.3.2 - Camada de Referência Remota

A camada de referência remota funciona como um *middleware*, entre os *stubs/skeletons* e a camada de transporte (Ahuja et al., 2000). Esta camada gerencia a comunicação entre cliente e servidor e a JVM e trabalha com uma interface de transporte de baixo nível por meio de um protocolo, protocolo este independente de *stubs* e *skeletons*.

Atualmente, a implementação da camada de referência está restrita a invocação ponto-a-ponto, onde um cliente interage com um único objeto não replicado cuja referência é somente válida quando o processo servidor está executando. A classe que implementa este protocolo é a *java.rmi.server.UnicastRemoteServer* (Costa, 2000).

2.7.3.3 - Camada de Transporte

A camada de transporte manipula os detalhes da conexão e providencia um canal de comunicação entre as JVMs. O lado cliente deve possuir a interface do objeto remoto para poder referenciá-lo, enquanto que o lado do servidor possui a implementação do mesmo.As responsabilidades desta camada são:

- Estabelecer conexões entre espaços de endereçamentos remotos, isto é, entre JVMs diferentes;

- Gerenciar conexões;
- Escutar chamadas que estejam sendo recebidas;
- Manter uma tabela que contém os objetos remotos residentes num determinado espaço de endereçamento;
- Estabelecer a conexão para as chamadas que estejam sendo recebidas;
- Localizar o alvo de uma chamada remota passando a conexão para quem a chamou.

2.7.4 - Interação entre camadas

Quando um cliente faz uma requisição para um objeto remoto, para o programador, o cliente parece acessar diretamente o servidor. Na realidade, o programa cliente acessa somente o *stub*.

O *stub* faz o *marshaling* dos argumentos, utilizando o mecanismo de serialização e então aciona a camada de referência remota para passar a requisição do método e argumentos para o objeto remoto apropriado.

A camada de referência remota converte a requisição do cliente para uma requisição baixo nível de transporte, para que a camada de transporte a envie.

No lado do servidor, a camada de referência remota recebe o nível de transporte e o converte para uma requisição para o *skeleton* servidor (Ahuja et al., 2000).

O *skeleton* converte a requisição remota para uma chamada apropriada ao objeto servidor. Isto envolve o método *unmarshaling*. Os argumentos enviados como referências remotas são convertidos para *stubs* locais no servidor e

argumentos enviados como objetos serializados são convertidos como cópias dos originais.

Se o método retorna um valor ou exceção, o *skeletons* utiliza-se de *marshaling* para transportar o objeto de volta para o cliente.

2.8 - PESQUISA BIBLIOGRÁFICA

Percebe-se atualmente, uma rápida evolução dos recursos tecnológicos relacionados a transações remotas, sejam estas comerciais, domésticas ou acadêmicas. Estes recursos acabam por necessitar cada vez mais das características encontradas nos objetos distribuídos. Assim, a computação distribuída vem mantendo seu espaço no campo de pesquisa da informática, sustentando cada vez mais esta evolução.

Muitos padrões comerciais para distribuição destes objetos, como o CORBA ou o RMI já citados na seção anterior, se estabilizaram no mercado, e passaram a se desenvolver de forma abrangente e eficiente, facilitando o emprego das características da distribuição, tentando minimizar sua complexidade e propondo novos recursos.

Entretanto, existem poucas ferramentas com o objetivo de projetar e modelar os sistemas de objetos distribuídos, a fim de demonstrar como melhor distribuir estes objetos, visando melhorias para a aplicação. Observa-se, assim, que muitos sistemas vem sendo afetado por esta deficiência (Butler,1995).

Na tentativa de minimizar este problema, muitos autores vêm se interessando pela distribuição de objetos. As crescentes pesquisas nesta área, já demonstram que muitos pesquisadores estão reconhecendo os benefícios trazidos pelo paradigma da orientação a objetos no campo da computação distribuída. Os resultados destas pesquisas, já começam a surgir, refletindo em uma maior segurança para se trabalhar nestes ambientes. Assim, vem

crescendo, entre os pesquisadores um sentimento de diferentes opiniões sobre distribuição, e como os objetos devem ser empregados neste campo. Estas diferentes idéias sobre a melhor forma de se utilizar os objetos na computação distribuída, também vem contribuindo para o crescimento dos estudos sobre objetos distribuídos (Bakker et al.,1997).

Neste capítulo é apresentada uma pesquisa que relaciona alguns destes estudos sobre objetos, em sistemas distribuídos, nos mais diversos aspectos. Cada trabalho difere-se em suas implementações, cada qual apresenta como enfoque principal, objetivos distintos, observando formas diferentes de se tratar os objetos em seus algoritmos. Para a distribuição, todos os trabalhos se assemelham em suas conclusões, onde buscam melhorar suas aplicações, explorando as características apresentadas ao se utilizar a tecnologia de objetos distribuídos como: desempenho, disponibilidade de sistema, tolerância a falhas e etc.

Percebe-se assim, que muitos trabalhos estão sendo realizados na área de objetos distribuídos. Porém, a escassez de técnicas para melhor distribuir os objetos ainda prevalece.

Detalham-se algumas experiências com diversos propósitos. Estes trabalhos variam desde regras básicas para se distribuir, passando por modelagens para replicações até checagem de *deadlocks*.

2.8.1 – Regras Básicas para Distribuição

Sessions (Sessions, 1996) apresenta regras para distribuição. As regras aqui apresentadas são básicas e iniciais, mais de grande importância para se iniciar um projeto que se espera ser bem sucedido.

Algumas regras para se trabalhar com objetos distribuídos:

- Conhecer a tecnologia: entender o que são objetos distribuídos, qual é seu propósito, avaliar se devem, realmente, ser utilizados em determinados projetos;

- Analisar a complexidade envolvida: o código de um ambiente distribuído é composto de muitas linhas e interfaces, sendo mais complexo que um ambiente centralizado;

- Analisar o desempenho desejado: como as atividades são remotas, um tempo mais significativo deve ser considerado;

- Conhecer padrões: atualmente existem no mercado padrões abrangentes e muito eficientes, distintos para cada aplicação em que se deseja executá-los, mais simples como o Java RMI, mais abrangente, complexo e multiplataforma como o CORBA;

- Definir um plano de projeto: traçar um plano de projeto tentando idealizar onde os objetos serão criados, observar o número de sistemas que devem ser preparados para acomodá-los, como instanciá-los e desabilitá-los, como os objetos participarão das transações, tentar criar expectativas reais do sistema;

- Definir mecanismos de localização dos objetos: como os clientes encontram estes objetos, são registrados no *naming service* ou outro mecanismo;

- Criar um protótipo: tentar desenvolver um protótipo antes do desenvolvimento real, para se ter certeza que a distribuição poderá ser aplicada e trará resultados esperados;

- Testar: testar tudo que possível localmente, e depois distribuir, já que é mais fácil e rápido testar de depurar erros na mesma máquina.

2.8.2 – Um Modelo de Replicação de Objetos

Quando os recursos de computação são limitados, aumentar o grau de replicação de um objeto pode afetar desfavoravelmente a qualidade do serviço provido. Além disso, em um sistema distribuído complexo é difícil prever as necessidades das diferentes aplicações, pois provavelmente, estas variam dinamicamente enquanto o sistema está sendo executado.

Assim, manter o grau de replicação do objeto para alcançar um nível desejável de disponibilidade, enquanto se aloca recursos suficientes para os objetos a fim de garantir uma qualidade segura de serviço, é um problema desafiador.

Neste estudo, Kalogeraki (Kalogeraki et al., 1999) possui como motivação principal o fato que, em um sistema de objetos distribuídos, a aplicação de objetos pode ser replicada em processadores diferentes para prover tolerância a falhas e alta disponibilidade. Assim, constrói um algoritmo cuja meta é determinar o grau de replicação para cada objeto, maximizando a utilidade do sistema enquanto respeita os recursos disponíveis. O algoritmo determina um grau inicial de replicação para cada objeto e ajusta o grau de replicação dinamicamente em tempo real.

As características principais deste modelo são:

- O grau de replicação de objetos fornecido para a aplicação neste modelo depende da importância dos objetos na aplicação das tarefas;
- Os objetos mais replicados são aqueles de aplicações que possuem funcionalidades críticas, enquanto que, os objetos menos importantes são replicados menos vezes.
- Assume-se que réplicas individuais de um objeto falham independentemente com iguais probabilidades, a possibilidade de que todas as réplicas de um objeto falhem, diminuem com o aumento do número das réplicas;

- Consideram-se dois tipos de recursos: CPU e memória;
- Réplicas de um objeto devem ser independentes e prover a mesma funcionalidade que o objeto de origem;

Basicamente, considera-se um sistema composto de um conjunto de processadores que suportam execução de múltiplas tarefas. Assume-se que cada recurso a ser utilizado por estas múltiplas tarefas possui desempenho independente e que cada tarefa individualmente contribui somente em pouca proporção para sobrecarregar os recursos. O gerenciamento de recursos invoca o algoritmo quando os objetos são replicados inicialmente e quando as condições de sistema mudam consideravelmente.

Considera-se um conjunto de objetos e suas utilidades, que são avaliadas de forma a ordená-los pela importância. Por definição, cada objeto pode ter um máximo de quatro replicas. Usando fórmulas matemáticas desenvolvidas para tal modelo, computa-se a utilidade de cada réplica do objeto. Inicia-se utilizando as réplicas de mais alta utilidade. Quando o algoritmo é invocado, a réplica de importância mais elevada para cada objeto é alocada, após, o número de réplicas deste objeto é incrementado. Se um objeto adicional for introduzido, este deve ser acomodado de forma a manter a ordem por utilidade, quebra-se a seqüência e começa-se a considerar também a utilidade deste novo objeto. Réplicas de objetos podem ser removidas quando uma nova tarefa iniciar ou quando a capacidade dos recursos for excedida.

Determina-se o grau de replicação para aplicação de objetos e aloca os objetos para os vários processos. Intuitivamente, a utilidade do sistema é maximizada quando mais réplicas de objetos com utilidades mais altas são selecionadas.

Assim, Kalogeraki propõe em sua modelagem, um algoritmo que determina o grau de replicação dos objetos, a fim de alocar recursos de forma eficiente, melhorando desta forma, o desempenho do sistema. O algoritmo baseia-se na importância da replicação do objeto e recursos como memória e CPU, tendo

como meta evitar prejudicar a qualidade do serviço quando os recursos são limitados e o número de replicas de objetos tendem a aumentar.

2.8.3 - Um Modelo que Detecta Deadlock

Objetos que são distribuídos por diferentes máquinas podem ser regularmente executados concorrentemente. Um objeto cliente que reside em uma máquina executa concorrentemente com um objeto servidor que reside em uma máquina diferente. Muitas vezes, pode haver vários diferentes objetos clientes que podem fazer requisições de operações de um servidor concorrentemente. Esta execução concorrente traz uma interessante questão sobre como a sincronização entre cliente e objetos servidores é alcançada.

As execuções concorrentes e paralelas podem acarretar alguns problemas, como os *deadlocks*. A principal contribuição da pesquisa de Emmerich (Emmerich et al.,2000) é primeiramente identificar este importante grupo de problemas existentes em sistemas de objetos distribuídos. Explora-se o fato que o objeto *middleware* possui uma pequena política de sincronização e posteriormente, expressa-se esta sincronização em modelos dinâmicos da (UML) *Unified Modeling Language*. Definem-se as semânticas para se mapear estes modelos no padrão UML para processos algébricos. Finalmente, aplicam-se modelos de checagem nesta notação de processos algébricos e demonstra-se que se é capaz de detectar os possíveis *deadlocks* que foram relatados anteriormente, no modelo dinâmico da UML.

De acordo com Emmerich (Emmerich, 2000), o *middleware* suporta diferentes primitivas de sincronização e determina como clientes e objetos servidores se sincronizam durante a requisição. Requisições síncronas bloqueiam o cliente até que o servidor processe a requisição e retorne o resultado da operação. Esta é a primitiva de sincronização padrão não só em CORBA, mas também em RMI e COM. Nas requisições assíncronas o controle retorna para o cliente

assim que a invocação é realizada. Depois da invocação, o cliente está livre para outras tarefas, ou realizar novas requisições.

Deadlocks que envolvem mais de um objeto são difíceis de serem detectados manualmente. Existem desenvolvedores que necessitam de ferramentas automatizadas para prover suporte, já em um estágio inicial de desenvolvimento de seu projeto.

Discute-se a utilização da UML para expressar primitivas de sincronização nas requisições de objetos. Utiliza-se neste modelo, os diagramas de colaboração e seqüência, e é através destes diagramas que pode se perceber se existe uma possível situação em que uma interação conduz a um *deadlock*. Um *deadlock* pode não ser tão facilmente percebido, já que o comportamento do objeto distribuído foi determinado em um nível de abstração, ou seja, no diagrama de classes, e o comportamento da sincronização foi modelado em um segundo nível de abstração, que é o diagrama de interação. Porém, são extraídas informações suficientes dos diagramas de classes e interação para automaticamente derivar uma especificação formal de sincronização e comportamento. Após, utiliza-se uma notação algébrica para este propósito que é suportada, por sua vez, por um modelo de checagem.

Assim o processo de checagem procura por *deadlocks*, ao encontrá-los, estes podem ser resolvidos. Sugere-se ainda, nesse artigo, possibilidades de trabalhos futuros relacionados com esta idéia inicial.

CAPÍTULO 3

TRABALHO PROPOSTO

A arquitetura SICSD surge como uma nova abordagem para o software de controle de satélite do INPE. O SICSD explora os recursos da distribuição, e se apresenta como uma arquitetura flexível e dinâmica para objetos distribuídos.

Para a realização deste trabalho, o estudo de caso considerado é um subsistema pertencente à arquitetura SICSD, denominado Software Simulador de Satélites. Por estar atuando em ambiente centralizado, o Software Simulador de Satélites possui algumas limitações com relação aos aspectos de: disponibilidade, tolerância a falhas, flexibilidade em atender em diferentes situações, etc.

O trabalho proposto tenta explorar os benefícios da distribuição de objetos. Para tanto, aplica-se o paradigma de objetos distribuídos ao Simulador de Satélite. A tentativa de se importar as vantagens deste paradigma ao Software Simulador possui o objetivo de minimizar ou até mesmo eliminar algumas das limitações apresentadas por este Software, advindas do ambiente centralizado.

Além de explorar os benefícios da distribuição, o presente trabalho também tem como meta, levantar aspectos sobre técnicas de modelagens para a distribuição de objetos.

São idealizadas modelagens de objetos baseadas na análise do comportamento dos objetos, na aplicação e nas necessidades dos usuários. Através de critérios impostos por estas técnicas de modelagem da distribuição, pretende-se melhorar as características providas pela distribuição dos objetos.

Aplica-se aos objetos do Software Simulador de Satélites, as técnicas de modelagens aqui apresentadas. A finalidade é estabelecer uma organização da distribuição dos objetos, tentando assim obter melhorias às características proporcionadas pela distribuição empregada ao Software Simulador.

Para melhor explicar o trabalho proposto, faz-se necessário o detalhamento das duas arquiteturas. A primeira arquitetura é a SICSD e consiste em um ambiente dinâmico e distribuído para objetos distribuídos empregados no Software de Controle de Satélites. A segunda arquitetura é o Simulador de Satélites, software que compõe a arquitetura SICSD.

Apresentam-se, a seguir, estas arquiteturas para que seja possível a compreensão da proposta de trabalho aqui apresentada, posteriormente apresenta-se o trabalho proposto propriamente dito.

3.1 - A ARQUITETURA SICSD

A arquitetura SICSD consiste em apresentar uma aplicação distribuída ao software de controle de satélites, onde seus objetos se apresentam de forma dinâmica, ou seja, podem migrar de um nó para outro, se adaptando às solicitações de serviços, melhorando um conjunto de características, como desempenho, flexibilidade, confiabilidade e utilização dos recursos computacionais disponíveis.

A arquitetura SICSD apresenta seus objetos se comunicando através de um *middleware*, que implementa a especificação CORBA. Na Figura 3.1 apresentam-se os referidos objetos (Ferreira, 2001):

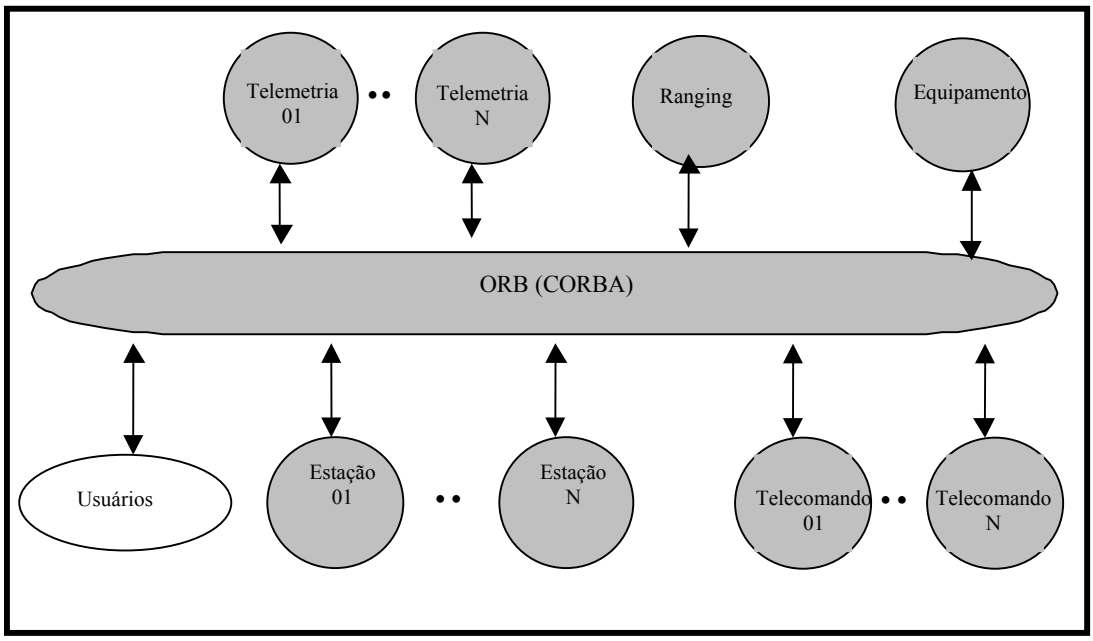

FIGURA 3.1 – Arquitetura SICSD.

FONTE :Adaptada de Ferreira (2001).

- ***Telemetria***: apresenta o estado interno do satélite, como por exemplo, a voltagem de um determinado circuito, o posicionamento de uma determinada chave (ligado ou desligado), ou seja, mostra as condições dos equipamentos do satélite e ainda verifica a execução dos telecomandos.

- ***Telecomando***: representa mensagens que podem ser enviadas para o satélite, com finalidade de corrigir ou mudar posições de chaves, ligar e desligar sensores, etc.

- ***Estação***: descreve as estações utilizadas para recepção dos dados dos satélites, que são controlados pelo INPE. Atualmente existem duas estações: Cuiabá e Alcântara.

- ***Ranging***: desempenha a função de medir a distância do satélite em relação à Terra, bem como os métodos responsáveis por esse cálculo.

- ***Equipamento***: descreve os equipamentos instalados para a recepção e transmissão de dados para o satélite.

Além destes objetos da aplicação para controle de satélites, a arquitetura SICSD compreende os serviços dos Agentes, Persistência, Segurança e Balanceamento.

A Figura 3.2 apresenta uma visão da arquitetura SICSD e seus serviços disponíveis:

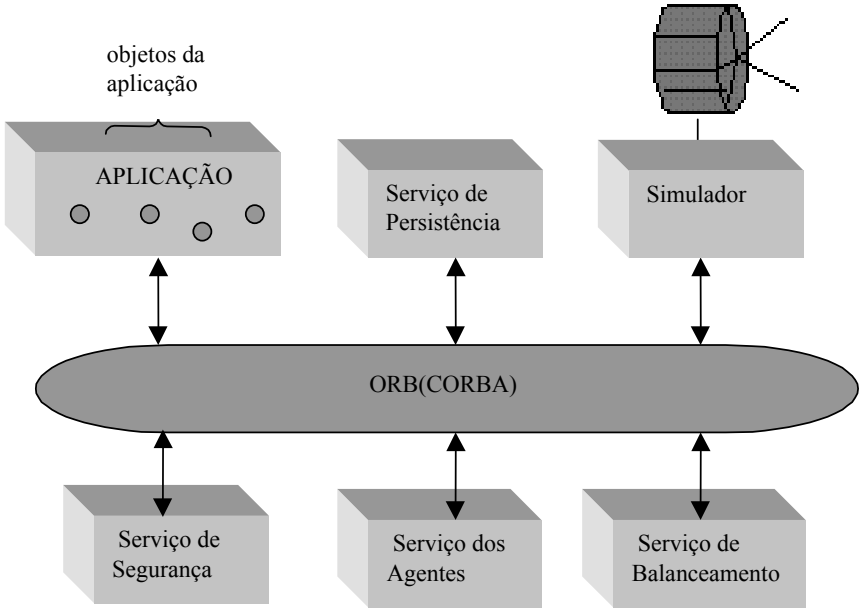

FIGURA 3.2 - Uma visão dos serviços da arquitetura SICSD.
FONTE: Ferreira (2001).

- Serviço dos agentes: a localização física dos objetos instanciados, o estado disponível ou não do objeto para receber solicitações de serviço do sistema, disponibilidade da CPU de um determinado nó, são algumas

das informações gerenciadas pelos serviços dos agentes, a fim de monitorar o estado dos objetos, bem como o status da rede, mantendo atualizadas e distribuídas as informações necessárias para se desenvolver uma aplicação dinâmica e flexível para controle de satélites.

- Serviço de Balanceamento: este serviço é capaz de levantar os pontos críticos do ambiente configurado para controle de satélites, ou seja, identificar os nós que contêm uma sobrecarga de trabalho, tomando decisões que podem resultar na migração ou na replicação dos objetos da aplicação para controle de satélites. Portanto, é responsabilidade deste serviço garantir os recursos de mobilidade, a flexibilidade e o dinamismo da arquitetura proposta. Por exemplo, se há um número elevado de objetos instanciados em um determinado nó, o serviço de balanceamento é responsável pela migração de objetos de máquinas saturadas para máquinas ociosas; ou ainda, se objetos instanciados em determinado nó estiverem recebendo várias solicitações de serviços de outros nós da rede, o serviço de balanceamento deve replicar os objetos de um nó saturado em nós ociosos.

- Serviço de Persistência: este serviço 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; eles simplesmente acionam o serviço de persistência para realizar essa operação.

- Serviço de Segurança: responsabiliza-se por garantir o acesso ao sistema somente de pessoas previamente autorizadas, restringindo-as às funções previamente definidas, de acordo com o seu perfil. Como a arquitetura SICSD permite que objetos migrem de uma máquina para outra; mecanismos de segurança também devem ser utilizados para garantir que somente objetos autorizados possam migrar para máquinas remotas.

- Simulador de Satélites: objeto responsável por simular os possíveis estados de um satélite, ou seja, as possíveis condições internas que ele apresenta em um determinado instante.

Sendo o foco deste trabalho apresentar uma proposta para o estudo de objetos distribuídos, utilizando como estudo de caso, o Software Simulador de Satélites, faz-se necessário uma introdução de como este software se apresenta atualmente.

3.2 – O SIMULADOR DE SATÉLITES

A finalidade principal do Simulador de Satélites é permitir treinar os operadores quanto ao controle e rastreio do mesmo, e prover um ambiente virtual que possa ser utilizado para elaboração dos testes de aceitação do software de sistema de controle de satélites (Yamaguti et al., 1994).

Assim, o Simulador de Satélites desempenha as seguintes funções:

- Simular características físicas necessárias à operação do satélite, como: órbita, atitude, campo magnético, incidência da luz solar;

- Simular todas as funções dos subsistemas do satélite real como: telemetria, telecomando, temperatura, medidas de distância, azimute, elevação, carga e descarga da bateria e ainda funções das estações de rastreio;

- Prover interface com o SICSD;

- Controlar todo o processo de simulação, ativando ou desativando as funções de acordo com as solicitações do operador, tais como:

    - Inicialização ou finalização do processo de simulação;

    - Recuperação de informações já armazenadas na Base de Configuração;

- Visualização e monitoração dos estados correntes do processo de simulação;
- Registro e emissão de relatórios que descrevem configurações de Base de Configuração, Log e estado corrente;
- Edição de arquivos de configuração;
- Inserção de falhas no funcionamento do satélite simulado.

3.2.1 - Arquitetura do Simulador de Satélites

A arquitetura do Simulador de Satélites, que pode ser observada na Figura 3.3, é dividida em dois principais subsistemas:

- Controle de Processo Modelado;
- Modelos de Satélite, Ambiente e Estações.

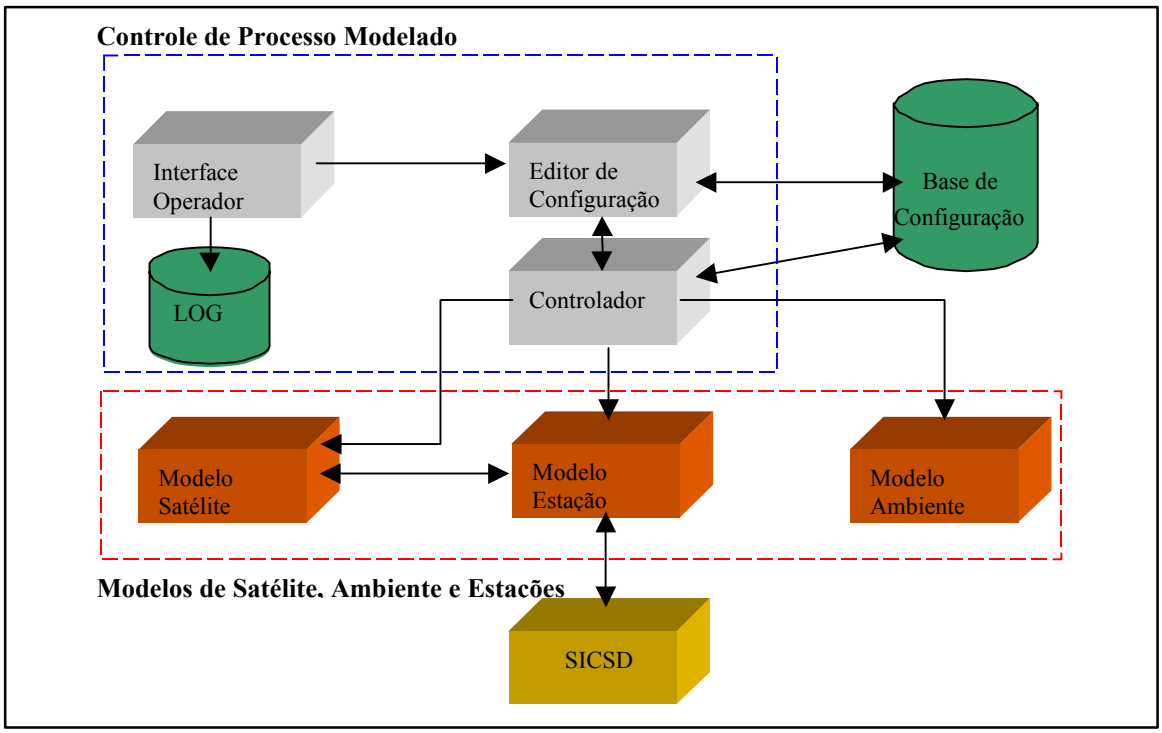

FIGURA 3.3 - Arquitetura do Simulador de Satélites.
FONTE: adaptada de Rozenfeld et al. (1990).

O subsistema denominado Controle do Processo Modelado, é responsável pela interface com operador de sistema, geração de relatórios, manutenção dos arquivos de configuração e log e pela ativação e desativação dos subsistemas que implementam os modelos do satélite. É constituído pelos seguintes subsistemas:

- Controlador do Simulador de Satélites;

- Editor de configuração;

- Gerenciador de log;

- Interface com operador.

A segunda parte, além de ser responsável pela interface com o SICSD, implementa os seguintes modelos:

- Ambiente no qual o satélite opera;

- Estações de rastreio e controle do satélite;

- Satélite.

O Simulador de Satélites é operado através da interface com operador em um terminal exclusivo onde o Simulador de Satélites é inicializado, e, a partir dele, o responsável pela simulação deve poder acompanhar o andamento do processo, monitorando seu estado e interagindo com o mesmo a fim de exercer ações de controle ou provocar falhas.

As falhas podem ser previamente programadas durante o processo de inicialização ou provocadas durante a simulação, alterando-se parâmetros necessários para este fim. A inserção de falhas, assim como os comandos para visualizar ou alterar parâmetros e congelar seção, são identificados pelo controlador de simulação, subsistema também responsável pela ativação ou não destas operações.

É através do editor de configuração que se edita a Base de Configuração do sistema, onde se encontram os itens de identificação, senha, e atributos do usuário, telas que permitem visualizar e monitorar estados correntes da simulação, parâmetros que definem este estado e as falhas simuladas.

As ocorrências de eventos importantes são registradas num arquivo de LOG, que pode ser consultado durante o processo de simulação. Este processo é monitorado através do gerenciador de log.

As trocas de mensagens com o SICSD ocorrem normalmente, como se fosse o sistema real, não havendo, por parte do controlador de satélites do SICSD como distinguir a operação simulada da real.

3.2.2 – O Processo de Simulação

O comportamento do processo de simulação é totalmente definido pelas condições iniciais (contidas na configuração inicial), pelos algoritmos que implementam os modelos do satélite e ambiente no qual o mesmo opera, pelos telecomandos recebidos e comandos de simulação emitidos pelo operador do Simulador de Satélites.

A seguir descreve-se resumidamente o processo de simulação ilustrado na Figura 3.4:

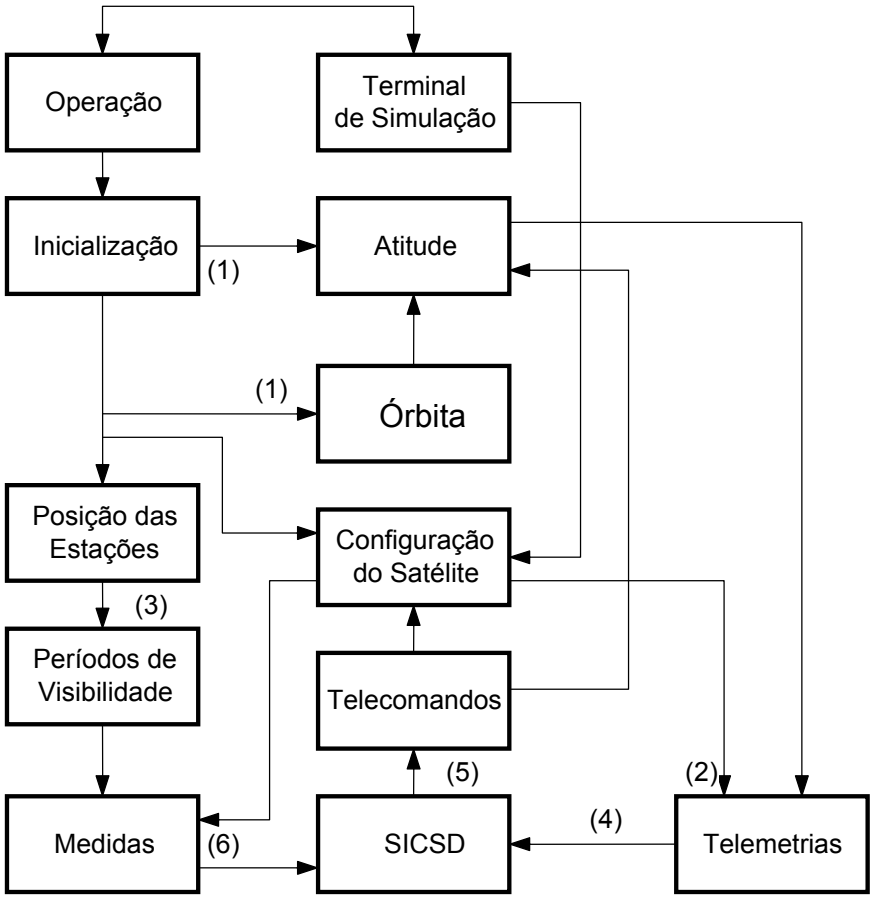

FIGURA 3.4 - Esquema simplificado do Simulador de Satélites.
FONTE Adaptada de Rozenfeld et al. (1990).

São inicialmente (1) obtidos os valores das variáveis estado da órbita (movimento de translação do satélite em relação à Terra) e atitude (movimento de rotação do satélite que determina sua orientação no espaço) do satélite, juntamente com o vetor posição do Sol num mesmo referencial para se verificar a condição de iluminação.

Através da verificação de iluminação do satélite e do ângulo de aspecto solar (ângulo entre o eixo de rotação de satélite e a direção de incidência de raios solares no mesmo), é simulada, com auxilio de um modelo térmico do satélite, a evolução da temperatura, nos pontos onde esta grandeza é monitorada por telemetria (2).

A simulação do processo de carga da bateria, também pode ser efetuada neste ponto, pois, depende também da condição de iluminação do satélite (que determina se os painéis solares estão gerando energia ou não). Pode-se verificar se está havendo excesso de energia, a qual é desviada para carregar a bateria. Se a corrente gerada pelos painéis solares for menos que o necessário, o processo de descarga da bateria passa a suprir parte do consumo.

Em cada passo da simulação deve ser verificada, para cada uma das estações de rastreio consideradas no processo, a condição de visibilidade (3), para que seja possível detectar a entrada e saída do satélite nesta região.

Devem ser observadas ainda, saídas das telemetrias que monitoram as variáveis dinâmicas do satélite, valores calculados através de modelos matemáticos ou tabelas contendo valores pré-avaliados através de modelagens dinâmicas, o que faz com que cada valor represente com realismo a evolução da respectiva grandeza.

As variáveis analógicas de telemetria mudam basicamente para refletir a atuação de algum telecomando enviado do solo, ou de circuitos temporizadores, como por exemplo, tensões de alimentações de equipamentos

de bordo ou chaves acionadas para energização de algum componente do satélite.

A simulação da operação de recepção dos quadros de telemetria da estação pelo Centro de Controle de Satélites (CCS) é simplificada (4), já que o grau de realismo é suficiente para se chegar à similaridade da operação real.

Em termos de telecomandos, a interface entre CCS e Simulador de Satélites (5) é idêntica ao satélite real, assim como a conexão entre o CCS e estações de rastreio e todas as mensagens entre os mesmos.

As medidas (6) de localização do satélite (distância, azimute e elevação) são simuladas a partir da posição orbital e das coordenadas da estação de rastreio cuja região de visibilidade se encontra.

3.3 - TRABALHO PROPOSTO

A necessidade de se aplicar as características inovadoras da distribuição de objetos ao Simulador de Satélites, é reforçada pela idealização da arquitetura SICSD. Sendo o Simulador um dos subsistemas desta arquitetura, é necessário acompanhar o desenvolvimento tecnológico do SICSD, para servi-lo apropriadamente no ambiente distribuído. Desta forma, selecionou-se o Simulador de Satélites como estudo de caso deste trabalho de dissertação.

O Simulador de Satélites, como já mencionado anteriormente, tem como objetivo simular o satélite, inclusive no que diz respeito aos efeitos das condições ambientais, as quais o mesmo está sujeito em órbita, se tornando realmente indistinguível do satélite real, permitindo treinar todo o pessoal envolvido na operação, atuando também como base para testes de aceitação e validação de procedimentos de operação e controle do satélite (Yamaguti et al., 1994).

Entretanto, algumas limitações podem ser observadas no sistema atual:

- Disponibilidade do sistema: No Simulador de Satélites, uma falha no subsistema, inviabiliza a execução de outra tarefa. Como exemplo, uma falha no subsistema que simula (TMS) telemetria impossibilita a visualização das telemetrias recebidas do satélite (simulador) e conseqüentemente impede a verificação da atuação de um determinado (TC) telecomando. Isto porque a verificação do *status* da execução de um telecomando se dá através de uma telemetria.

- Tolerância à falhas: A falha de um servidor em uma arquitetura centralizada compromete todos os subsistemas ali residentes e conseqüentemente toda a simulação da operação de controle de satélites.

- Flexibilidade para atender às diferentes situações de controle: 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. No caso específico do Simulador de Satélites, a fase considerada crítica é a anterior ao lançamento, onde verificações finais de softwares, validações de aplicativos e treinamentos da equipe de operação do satélite são constantes. Esta fase exige, portanto, uma necessidade maior de disponibilidade de máquinas a fim de atender a intensa demanda de usuários. A arquitetura centralizada não apresenta esta flexibilidade, ou seja, não possui disponibilidade de máquinas de acordo a necessidade, o que limita o atendimento do Simulador de Satélites a um pequeno grupo de usuários.

As limitações apresentadas pelo Simulador de Satélites, decorrentes do sistema centralizado, acabam por impor obstáculos ao desempenho de todo o processo. Percebe-se que estes obstáculos vão ao encontro das características apresentadas no paradigma de objetos distribuídos. Percebe-se

também, que trabalhar em ambiente distribuído, transparente, obtendo um melhor desempenho no resultado das aplicações, deixou de ser apenas uma tendência prevista no mundo da computação, passando à realidade (Mainetti,1997).

Assim, entende-se que existe a possibilidade de se reduzir ou até eliminar as limitações exibidas anteriormente, utilizando-se do recurso da computação distribuída, através de suas propriedades inovadoras. Desta forma, neste trabalho aplica-se objetos distribuídos ao Software Simulador de Satélites.

Como resultado, observa-se que muitas das limitações advindas do ambiente centralizado poderão ser minimizadas ou até mesmo solucionadas. São apresentadas nesta seção, sugestões de como se eliminar algumas destas limitações, já citadas, do Simulador Centralizado. O Capítulo 4 aborda a implementação das idéias propostas.

A necessidade de se analisar e organizar de forma adequada à distribuição dos objetos existentes, de acordo com as funções exercidas no processo de simulação também é um fato observado. Visando um melhor resultado da aplicação da distribuição ao Software Simulador, apresenta-se neste trabalho, um estudo focando a modelagem da distribuição. O objetivo, como já citado, é melhor distribuir os objetos nos subsistemas do Simulador de Satélites, tentando melhorar as características da distribuição apresentadas. Assim, detalha-se também nesta seção, as técnicas para se realizar a distribuição destes objetos.

3.3.1 - Eliminação das limitações do Simulador de Satélites

Como já citado, o Simulador de Satélites apresenta algumas limitações, pretende-se solucionar os problemas citados através das seguintes vantagens proporcionadas pela distribuição de objetos:

- **O aumento da disponibilidade de serviços**: A tecnologia de objetos distribuídos provê mecanismos que asseguram a disponibilidade dos objetos, independente de falhas nos computadores (Ferreira, 2001). Esta característica soluciona o problema de disponibilidade do Simulador de Satélites, elevando a disponibilidade de seus serviços.

- **Tolerância a falhas**: A falha de um computador ou objeto, em um ambiente distribuído representa apenas uma falha parcial do sistema, que pode ser superada: caso ocorra uma falha no nó, onde um determinado objeto se encontre instanciado, uma nova conexão pode ser estabelecida com outro objeto que presta serviço equivalente. Esta capacidade soluciona a limitação da versão atual do Simulador de Satélites de não apresentar tolerância a falhas.

- **O aumento da concorrência e desempenho**: A capacidade de instanciar cópias de um mesmo objeto, em máquinas distintas, proporciona um melhor desempenho ao atendimento a requisições de múltiplos usuários. 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.

- **Flexibilidade para atender às diferentes situações de controle:** Com a aplicação de objetos distribuídos no sistema do Simulador de Satélites, pode-se distribuir partes do processo de simulação, ou até mesmo replicá-lo totalmente em máquinas distintas. Desta forma, encontra-se uma maior disponibilidade de máquinas, aumentando o número limite de usuários que necessitem concorrentemente utilizar o Simulador de Satélites.

3.3.2 - Modelagens de distribuição propostas

Um sistema complexo de software, que apresenta qualidade, exibe uma harmonia que o torna flexível a modificações. Esta flexibilidade é conseguida, muitas vezes, através da utilização de algumas técnicas já consagradas, como por exemplo, uma modelagem.

Assim, constata-se que, a maioria dos projetos bem sucedidos são semelhantes em diversos aspectos, apresentando muitos elementos que contribuem para este sucesso, entre eles, a utilização de padrões e modelagens (Booch et al.,1999).

Um modelo é uma simplificação da realidade (Booch et al.,1999), sendo que um dos objetivos dos modelos é mapear o mundo real (Tamai,1999). Portanto, os melhores modelos estão relacionados à realidade (Booch et al.,1999).

O processo orientado a objeto possui os seguintes estágios (Sommerville, 2001):

- entender e definir o contexto e os modelos do sistema;
- projetar a arquitetura do sistema;
- identificar os principais objetos no sistema;
- desenvolver os modelos do projeto;
- especificar interfaces

Assim como a orientação a objetos, os objetos distribuídos também utiliza os modelos para representar o mundo real, possui ferramentas de distribuição, mas, como já citado, a modelagem enfocando a forma de distribuir, ainda não está bem definida.

No Capítulo 2, item 2.8 é apresentado algumas sugestões de alguns autores enfocando o processo de “como” distribuir objetos, mas percebe-se que ainda existem outras formas de distribuição que podem ser exploradas.

Neste contexto, observa-se a importância em se modelar técnicas de distribuição, ou seja, uma tentativa de se estabelecer critérios para melhor distribuir os objetos a fim de se acentuar os benefícios fornecidos pela distribuição.

Para buscar uma melhor organização dos objetos na aplicação, alguns aspectos importantes devem ser analisados, como por exemplo:

- A função exercida e o comportamento dos objetos diante da aplicação;
- Fatores individuais de cada objeto, ou quando relacionado a outros objetos;
- Aspectos determinantes da aplicação e ainda;
- As necessidades dos usuários.

A partir desta análise, idealiza-se o levantamento de algumas técnicas. Estas técnicas obedecerão a critérios impostos de acordo com as necessidades do desenvolvedor e características apresentadas pela aplicação. O objetivo é conseguir modelar mais adequadamente a distribuição de objetos, amenizando a deficiência existente no aspecto de como melhor organizar a distribuição dos mesmos.

As técnicas propostas, na próxima subseção, estabelecem critérios e diferentes formas de modelagens. A iniciativa não tem como objetivo apontar qual a melhor técnica a ser utilizada. Os esforços se concentram apenas em se levantar as vantagens e desvantagens das técnicas apresentadas, explorando fatores qualitativos e quantitativos das mesmas e classificando-as de acordo

com os resultados obtidos num estudo comparativo, que envolve desempenho e disponibilidade.

3.3.2.1 - Modelagens da distribuição baseada em casos de uso

No Simulador de Satélites, em determinados momentos, percebe-se uma interdependência existente entre objetos que não deve ser quebrada. É o que ocorre quando um objeto telecomando, recebido do cliente, ao ser tratado, reflete quase que imediatamente no frame de telemetria enviado ao SICSD. Observa-se então, que freqüentemente um objeto telecomando atua e afeta o objeto telemetria, assim, deixá-los trabalhando próximos ou até mesmo na mesma máquina, pode trazer algumas vantagens, se obedecendo a um determinado critério.

Este critério para distribuição de objetos pode ser conseguido analisando-se os subsistemas do Simulador de Satélites e fazendo com que objetos que se comunicam mais freqüentemente entre si, participem de um mesmo caso de uso. Estes objetos então, relacionados por caso de uso, serão criados numa mesma máquina. Com isso, se diminuem os relacionamentos entre os casos de uso e conseqüentemente entre as máquinas, proporcionando vantagens, tais como: redução do tráfego na rede e maior disponibilidade. Por exemplo, se existir uma falha em um determinado objeto de um caso de uso, este não prejudicará um segundo caso.

Na Figura 3.5, observa-se os objetos que se colaboram para a realização de um caso de uso. A elipse tracejada está delimitando esta colaboração, as cores equivalentes caracterizam instância de um mesmo objeto. Os objetos estão sendo representados por quadrados.

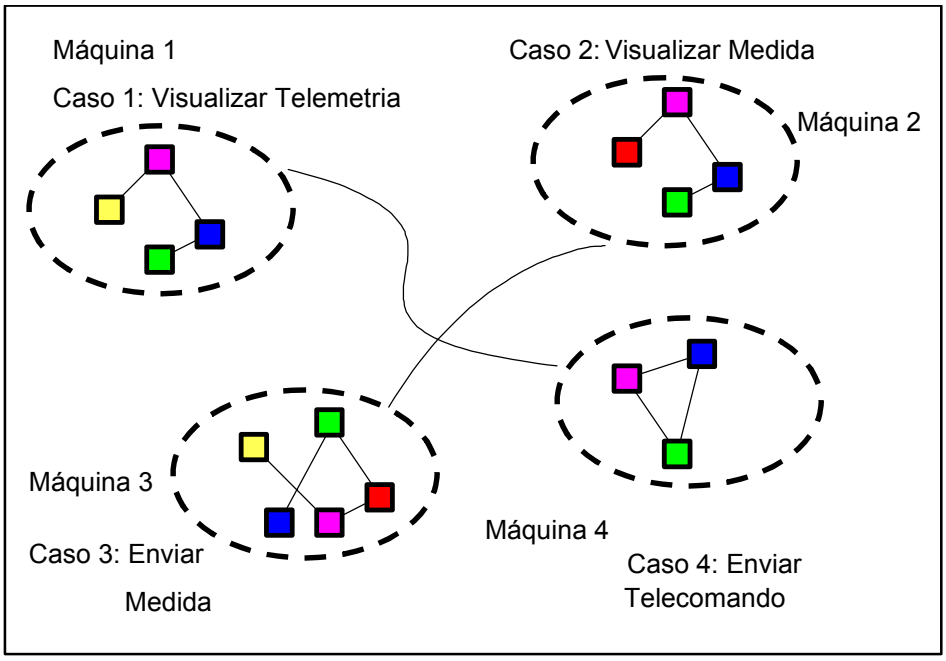

FIGURA 3.5 – Objetos distribuídos por modelagem baseada em casos de uso.

Não existe impedimento em se ter duplicidade de casos de uso em máquinas distintas, pois, sendo cópia de um caso de uso, o conjunto de objetos expressam o mesmo comportamento que o caso de uso de origem.

3.3.2.2 – Modelagem da Distribuição baseada em Tolerância a Falhas

Um segundo aspecto a ser abordado, na tentativa de se encontrar opções de modelagem para uma melhor distribuição, é proporcionar tolerância a falha de objetos. Focando nesta propriedade, o novo critério estabelecido, é replicar os objetos, independente do tipo de serviço executado, em todas as máquinas existentes no sistema. Neste caso, a sobrecarga das máquinas se torna inevitável, porém se assegura fortemente a disponibilidade de determinado objeto. A Figura 3.6 mostra como este sistema se apresenta:

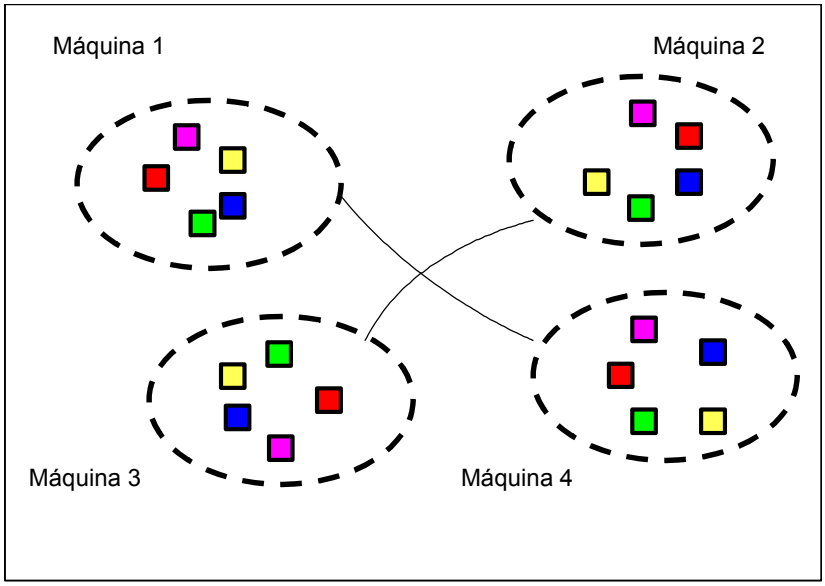

FIGURA 3.6 – Objetos distribuídos por modelagem baseada em tolerância a falhas

3.3.2.3 – Modelagem da Distribuição baseada na Forma Aleatória

A terceira forma apresentada consiste na modelagem aleatória.

No método tradicional, que é o atualmente utilizado, realiza-se a distribuição sem seguir qualquer critério previamente definido.

A modelagem aleatória é equivalente a este método tradicional, exceto por obedecer a um único critério imposto pela modelagem, que estabelece a existência de pelo menos uma cópia de um mesmo objeto no sistema. A Figura 3.7 a seguir ilustra esta distribuição:

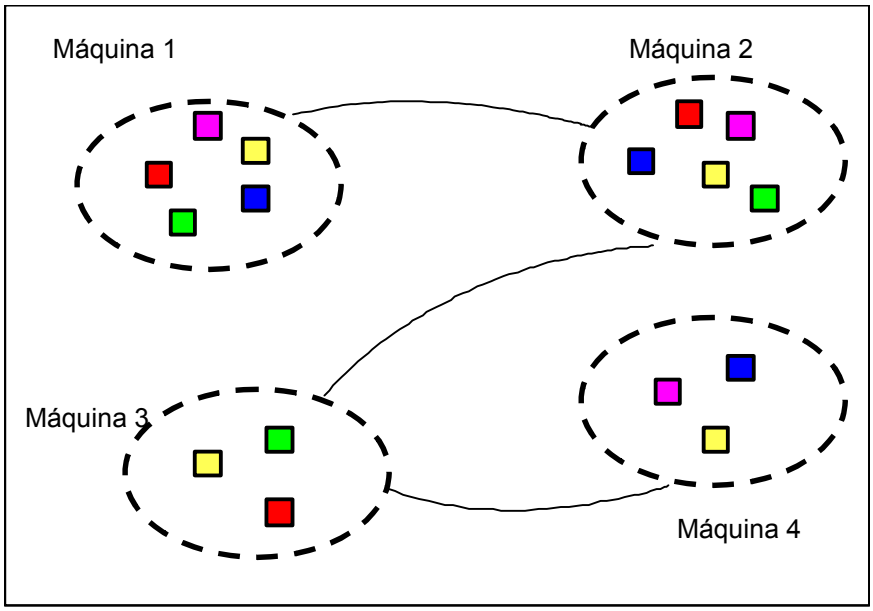

FIGURA 3.7 – Objetos distribuídos por modelagem baseada na forma aleatória.

Uma análise comparativa a respeito do desempenho das três formas de modelagem apresentadas também é desenvolvida. Os resultados são apresentados no Capítulo 6 do presente trabalho.

CAPÍTULO 4

DESENVOLVIMENTO DO TRABALHO PROPOSTO

A compreensão de um sistema pode ser melhor concretizada através da modelagem. A modelagem é uma técnica de engenharia aprovada e bem-aceita que simplifica a realidade (Booch et al.,1999).

Com relação ao software, a modelagem deve ser capaz de abranger as diferentes visões relacionadas à arquitetura do sistema e ainda, como esta arquitetura evolui ao longo do ciclo de vida de desenvolvimento do software. Assim, para auxiliar em uma melhor apresentação da modelagem do Software Simulador de Satélites Distribuído, utiliza-se do recurso dos diagramas da UML.

A UML é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de sistemas de software (Booch et al., 1999) e representa a coletânea das melhores práticas de engenharia que foram bem sucedidas na modelagem de um sistema complexo (OMG, 2001).

Utiliza-se a UML para:

- Mostrar as fronteiras de um sistema e suas funções principais;
- Ilustrar a realização destes casos de uso com diagramas de interação;
- Representar a estrutura estática de um sistema, empregando diagramas de classe;
- Modelar o comportamento de objetos através de diagramas de transição de estado;
- Revelar a arquitetura de implementação física com diagramas de componentes e de aplicação;

- Estender sua funcionalidade através de estereótipos.

Neste capítulo pretende-se elucidar a modelagem do software simulador, especificando seus requisitos e casos de uso. Apresenta-se também a estratégia de desenvolvimento do software, detalhando seus aplicativos auxiliares necessários e seu funcionamento.

4.1 - MODELAGEM DO SOFTWARE SIMULADOR DISTRIBUÍDO

Os casos de uso são importantes para visualizar, especificar e documentar o comportamento de um elemento. Pode-se aplicar o diagrama de caso de uso tipicamente de duas formas:

- ✓ Realizar a modelagem do contexto de um sistema;
- ✓ Realizar a modelagem dos requisitos de um sistema. (Booch et al., 1999).

Esta seção se restringe a elucidar as definições dos requisitos através do diagrama de caso de uso.

Como em todo projeto, um dos primeiros passos a ser tomado é definir as necessidades do sistema, delimitando suas fronteiras. Para tanto, é necessário estar ciente do que realmente o sistema realizará, definindo ainda em que ambiente o mesmo atuará (Sommerville, 2001).

Mas, um dos objetivos do caso de uso é especificar o comportamento desejado do sistema sem determinar como esse comportamento será executado. Assim, através do caso de uso se visualiza o sistema como uma caixa preta: é possível ver o que está fora do sistema e como o sistema reage a algo externo, mas não é possível ver como o sistema funciona internamente.

Desta forma, a modelagem dos requisitos envolve então, a especificação do que este sistema deve fazer, independente de como o sistema deve fazê-lo (Booch et al., 1999).

A análise é uma das fases que constitui o ciclo de vida do desenvolvimento de um sistema de software. Esta fase focaliza “o que”, ou seja, procura identificar os requisitos do sistema e do software, identificando quais as informações deverão ser processadas, qual função e desempenho desejados, quais interfaces deverão ser estabelecidas, quais as restrições do projeto e critérios de validação que são exigidos para se definir um sistema bem sucedido. Assim, o requisito é uma condição cuja exigência deve ser satisfeita (Sommerville, 2001).

Para especificar os requisitos a serem satisfeitos pelo Software Simulador de Satélites, deve-se considerar as aplicações previstas para a sua utilização. Estas aplicações se resumem em: auxiliar na aceitação do software aplicativo do CCS pelo CRC, auxiliar no treinamento do pessoal de operação e permitir realizar ensaios simulados da operação do satélite nas fases críticas (Rozenfeld et al., 1990).

Os requisitos se subdividem em grupos de acordo com suas características, podendo ser classificados como requisitos funcionais e requisitos não funcionais.

Como o caso estudado trata-se de um protótipo, estes requisitos foram resumidos e simplificados para atender apenas a necessidade do projeto em questão.

4.1.1 - Requisitos Funcionais do Software Simulador de Satélites Distribuído

O Software Simulador, no contexto deste trabalho, deve servir de base para que se possa atender principalmente as características da distribuição e estar disponível para a aplicação e testes das modelagens de distribuição já apresentadas. Assim, o Software Simulador Distribuído tem como objetivo:

- Prover um ambiente em que se possa aplicar a distribuição;
- Possibilitar a realização das medições necessárias para observar o comportamento dos objetos;
- Verificar como os modelos de distribuição podem alterar ou não nas características do mesmo.

Os requisitos funcionais descrevem a funcionalidade ou serviços que um sistema espera prover. As atividades a serem desenvolvidas neste protótipo foram selecionadas exclusivamente para atender aos objetivos do projeto. O Software Simulador de Satélites proposto apresenta várias funções, entretanto, dentre estas, selecionou-se algumas para a construção do protótipo. A seguir, observa-se as funcionalidades selecionadas que constituem o protótipo:

Requisitos de telemetria:

- *Frames* de telemetrias analógicas e digitais devem ser constantemente apresentadas na tela de telemetrias. As telemetrias analógicas simulam valores de sensores do satélite, as digitais simulam chaves ligadas ou desligadas;
- O simulador deve reproduzir as telemetrias analógicas e digitais do Satélite como saídas dos sensores, pontos de temperaturas internas do satélite, etc;

- As telemetrias poderão ser visualizadas nos períodos de visibilidade do satélite, ou seja, quando houver conexão de telemetria.

Requisitos de telecomando:

- Deve haver um procedimento para conexão de telecomandos;
- Os telecomandos recebidos podem ser visualizados pelo operador através das telemetrias;
- Deve-se preparar os telecomandos para serem enviados na configuração de telecomandos.

Requisitos de medidas de localização:

- Medidas devem ser geradas a fim de se obter a distância do satélite em relação à Terra;
- O envio de medidas de distância deve simular 32 medidas enviadas em grupos de 3 quadros;
- A leitura das medidas deve estar disponível para operadores;
- A conexão das medidas deve ser realizada.

Requisitos de processos auxiliares:

- O arquivo log, com todas as operações anteriormente realizadas, pode ser visualizado sempre que necessário.
- Permitir edição de configuração do telecomando a fim treinar operadores: a configuração de telecomando simula a preparação dos conjuntos de telecomandos a serem enviados pelo operador;
- Simular os equipamentos das estações, simulando conexões das estações de Cuiabá e Alcântara.

4.1.2 - Requisitos Não Funcionais

Os requisitos não funcionais, como o nome sugere, são os que não estão diretamente ligados com a funcionalidade do sistema, estão relacionados às características do sistema, propriedades reais como tempo, disponibilidade, capacidade de dispositivos de IO, etc. (Sommerville,2001).

- O simulador deve prover disponibilidade de serviço, principalmente no período determinado durante a passagem pela área de visibilidade da antena da estação, ou seja, se um nó deixar de funcionar, outro deve assumir o serviço.

- Nesta mesma fase deve prover tolerância a falhas, por exemplo, se um objeto telemetria não estiver apto a enviar telemetria, outro objeto telemetria deverá ser instanciado.

- A precisão é um fator fundamental em um procedimento de controle de satélites, já que a passagem pela área de visibilidade da antena ocorre em períodos determinados e existem intervalos longos entre uma passagem e outra. Um telecomando não enviado corretamente ou um erro do operador durante aquela passagem pode ocasionar danos irrecuperáveis no satélite. Desta forma o desempenho do sistema se torna de grande importância.

Com os requisitos definidos pode-se utilizar um diagrama de caso de uso para melhor expressá-los.

4.1.3 - Levantando Casos de Uso

Um caso de uso é uma interação típica entre um usuário e um sistema, um modo específico de utilização a partir de um ponto de vista segmentado de funcionalidade (Furlan, 1998). Como já citado, um caso de uso especifica o

comportamento do sistema ou parte do sistema e é uma descrição do conjunto de seqüência de ações, incluindo variantes que o sistema desempenha para produzir um resultado de valor para um ator (Booch et al., 1999). Na Figura 4.1, observa-se o diagrama de casos de uso do Software Simulador:

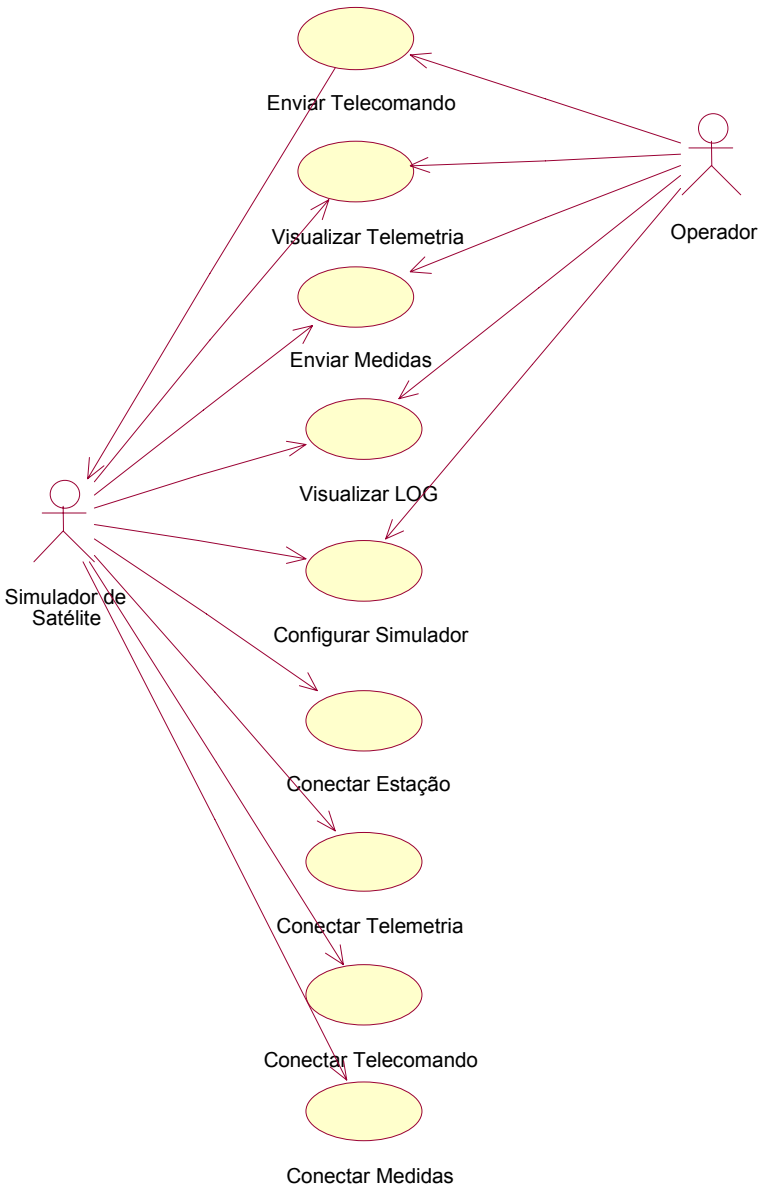

FIGURA 4.1 – Casos de uso do Software Simulador de Satélites

Para a compreensão, análise e realização destes relacionamentos é necessária a utilização de um segundo diagrama, denominado diagrama de colaboração. O diagrama de colaboração propõe uma visão do conjunto de elementos interligados da modelagem, exibindo uma interação organizada em torno dos objetos participantes e seus vínculos.

No próximo capítulo, na seção 5.2, detalha-se o diagrama de colaboração através de visões estruturais e dinâmicas representadas pelos diagramas da UML. A próxima seção destina-se a demonstrar como ocorrem as colaborações dos casos de uso.

4.1.4 – Diagrama de Colaborações

Um caso de uso, como citado anteriormente, captura o comportamento pretendido do sistema, sem ter que especificar “como” o comportamento é implementado. Assim, a análise do sistema, que especifica o comportamento, não é influenciada pela implementação, que especifica como o comportamento está sendo executado.

Contudo, normalmente, ao se implementar o caso de uso, necessita-se criar classes ou outros elementos para trabalhar juntamente para a implementação deste caso de uso.

Esta sociedade de classes e outros elementos, que trabalham em conjunto para especificar a realização de casos de uso, é denominada colaboração. Este diagrama pode ser observado na Figura 4.2:

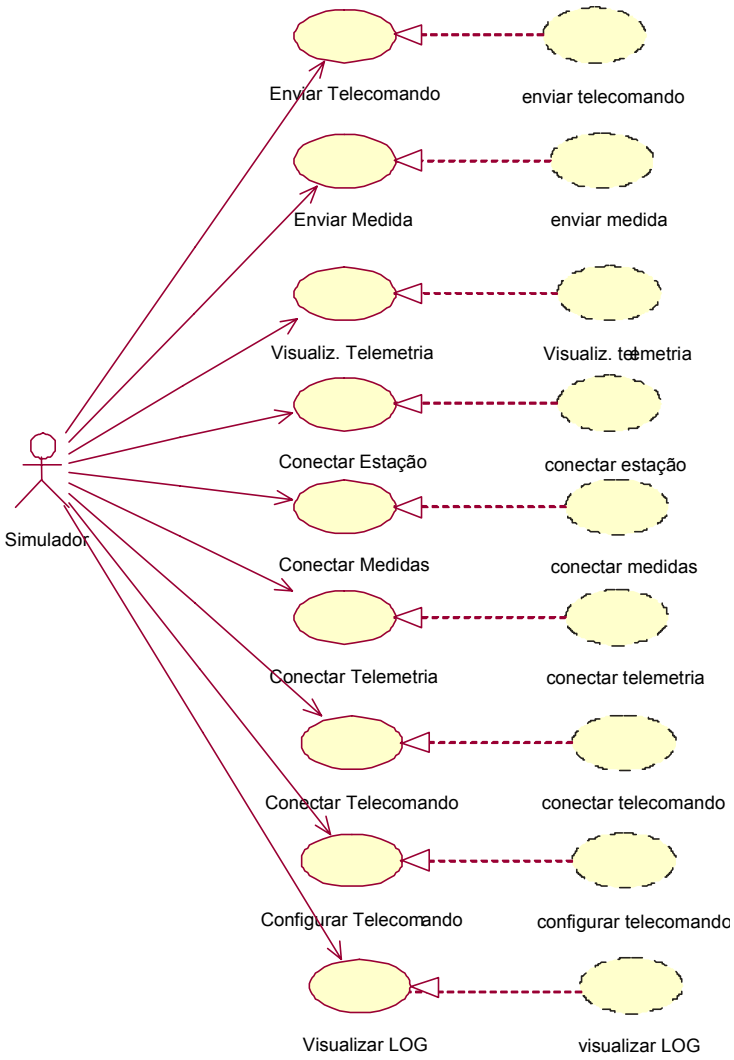

FIGURA 4.2 - Realização de caso de uso através do diagrama de colaboração.

Um diagrama de classes mostra um conjunto de classes, interfaces, colaborações e seus relacionamentos. Utiliza-se o diagrama de classes para modelar uma visão estática do sistema. São importantes não só para visualização, especificação e documentação do modelo estrutural, mas também para construção do sistema executável.

Na Figura 4.3, é apresentado o diagrama de classes da colaboração para realização do caso de uso “enviar telecomando” contendo seus atributos e

métodos.

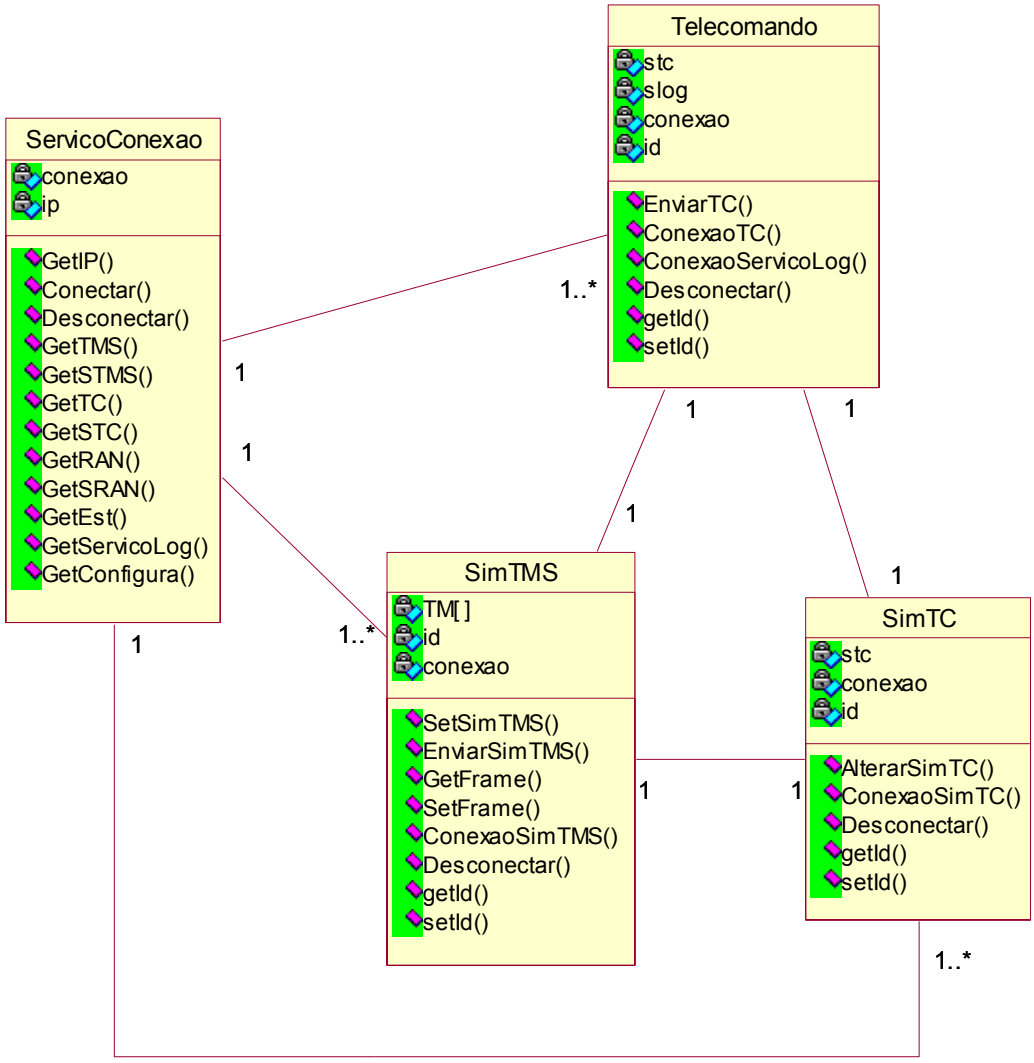

FIGURA 4.3 -Diagrama de classes do caso de uso:enviar telecomando.

O diagrama de classes da colaboração para a realização deste caso de uso contendo seus relacionamentos, atributos e métodos pode ser observado na Figura 4.4:

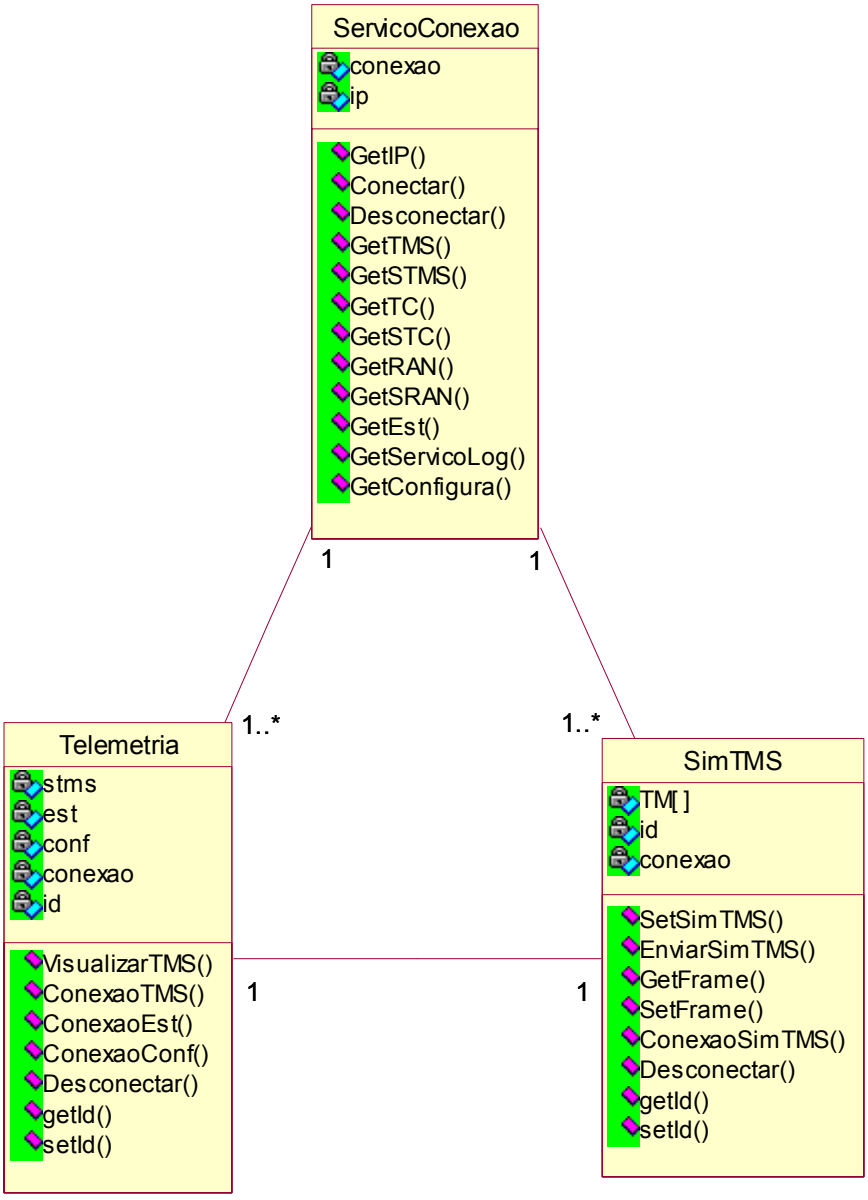

FIGURA 4.4 – Diagrama de classes: visualizar telemetria.

4.2–ESTRATÉGIAS DE DESENVOLVIMENTO DO SOFTWARE SIMULADOR DE SATÉLITES

Foi necessário utilizar, para o funcionamento do Software Simulador de Satélites, alguns outros serviços auxiliares que também foram desenvolvidos. É com auxílio destes serviços auxiliares que os objetos são distribuídos nos computadores da rede.

O ambiente de desenvolvimento do Software Simulador de Satélites se apresenta constituído pelos seguintes itens:

- 4 máquinas ligadas em rede, cuja descrição detalhada é apresentada no capítulo seguinte, na seção 5.1;

- Software Simulador de Satélites;

- software gerenciador de cenários;

- base de configuração;

- serviço de carga e conexão para cada máquina.

Na Figura 4.5, ilustra-se a interação dos itens que constituem o ambiente de desenvolvimento do Software Simulador de Satélites:

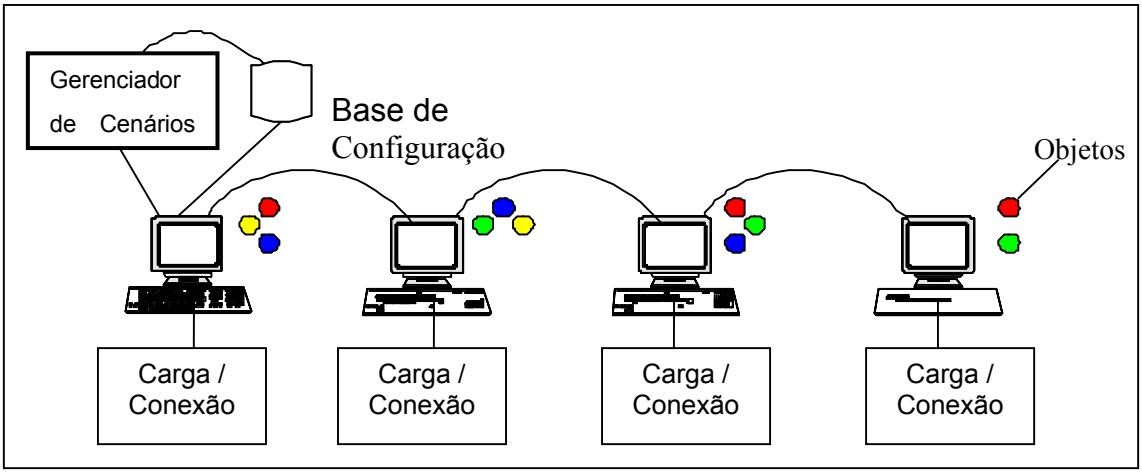

FIGURA 4.5 – Ambiente de desenvolvimento.

O Software Simulador de Satélites obedece a uma distribuição de objetos selecionada previamente. O responsável por determinar esta pré-seleção é o software gerenciador de cenários, que é um software que possui o objetivo de oferecer opções de cenários para a distribuição.

São aqui denominadas cenários, as várias configurações que podem ser formadas pelos objetos com relação a sua localização nas máquinas, ou seja, como os objetos podem ser encontrados nas quatro máquinas apresentadas na Figura 4.5.

Assim, o software gerenciador de cenários cria estes vários cenários, gerenciando as corretas configurações na base de configuração. A próxima subseção destina-se a apresentar detalhes sobre este software.

A base de configuração armazena as informações necessárias para que os objetos sejam criados nas máquinas disponíveis. O serviço de carga auxilia no processo de criação destes objetos e o serviço de conexão auxilia na utilização destes objetos criados. Estas atividades serão melhores detalhadas no decorrer desta seção.

Na Figura 4.6, pode-se observar o procedimento do protótipo:

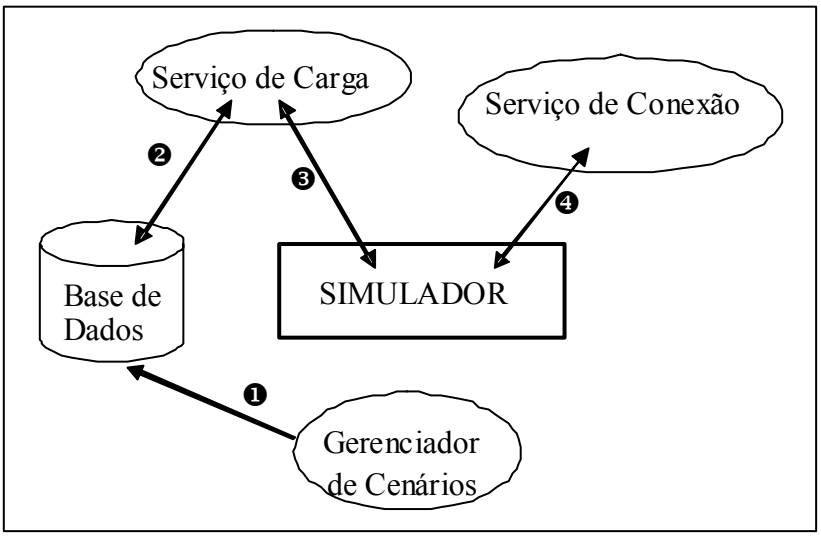

FIGURA 4.6 – Funcionamento do Software Simulador.

Com a finalidade de modelar a distribuição dos objetos, determina-se, o cenário mais adequado às necessidades do usuário ou da aplicação. Ou seja, pode-se previamente selecionar uma forma de distribuir os objetos de acordo com os possíveis cenários disponíveis. Estes cenários podem ser, por exemplo, todos os objetos em uma máquina, ou todos os objetos em todas as máquinas ou ainda algumas das três formas de modelagem de distribuição apresentadas na seção anterior, no item 3.3.2. As diversas possibilidades de cenários são melhores explicadas na seção destinada ao software gerenciador de cenários.

Estabelece-se, portanto, através do cenário a ser utilizado, em quais máquinas os objetos serão distribuídos. Através do software gerenciador de cenários, aplica-se o cenário já selecionado, ou seja, insere-se na base de configuração as informações sobre objetos relacionados com os nós. Assim, a base de configuração é alterada conforme solicitação do gerenciador de cenários ❶. Os objetos são então relacionados com os nós na base de configuração, ficando à disposição do Software Simulador de Satélites. Para que o Software Simulador de Satélites seja ativado, um dos primeiros procedimentos a ser realizado é

acionar, em todas as quatro máquinas participantes do projeto, a classe servidora denominada serviço de carga. A base de configuração servirá também como fonte de consulta a esta classe para que a mesma possa verificar quais os objetos devem ser criados nas máquinas destinadas aos mesmos.

O serviço de carga é responsável por esta verificação e criação dos objetos ❷.

Após criação de todos os objetos nas diferentes máquinas, conforme indicação da base de configuração, o simulador fica esperando que um comando seja ativado pelo operador, ou seja, o operador do Software Simulador já pode acionar funções como: visualizar telemetrias, enviar telecomandos etc, pois os objetos já foram criados e, portanto, já se encontram disponíveis ❸.

Ao ser selecionado um destes comandos, uma referência do objeto responsável por atender a este comando é obtida, através da classe serviço de conexão. O serviço de conexão localizará o nó em que se encontra este objeto com auxílio da base de configuração e solicitará a referência do objeto ❹. Assim, o comando do Simulador já pode atuar.

Este procedimento ocorre para todos os demais comandos. Percebe-se que os objetos se distribuem obedecendo ao critério regido pelo cenário, ou seja, os objetos podem, agora, ser distribuídos de acordo com os critérios de modelagem apresentados neste trabalho.

Na Figura 4.7, observa-se a tela do Software Simulador de Satélites, ilustrando-se seus comandos:

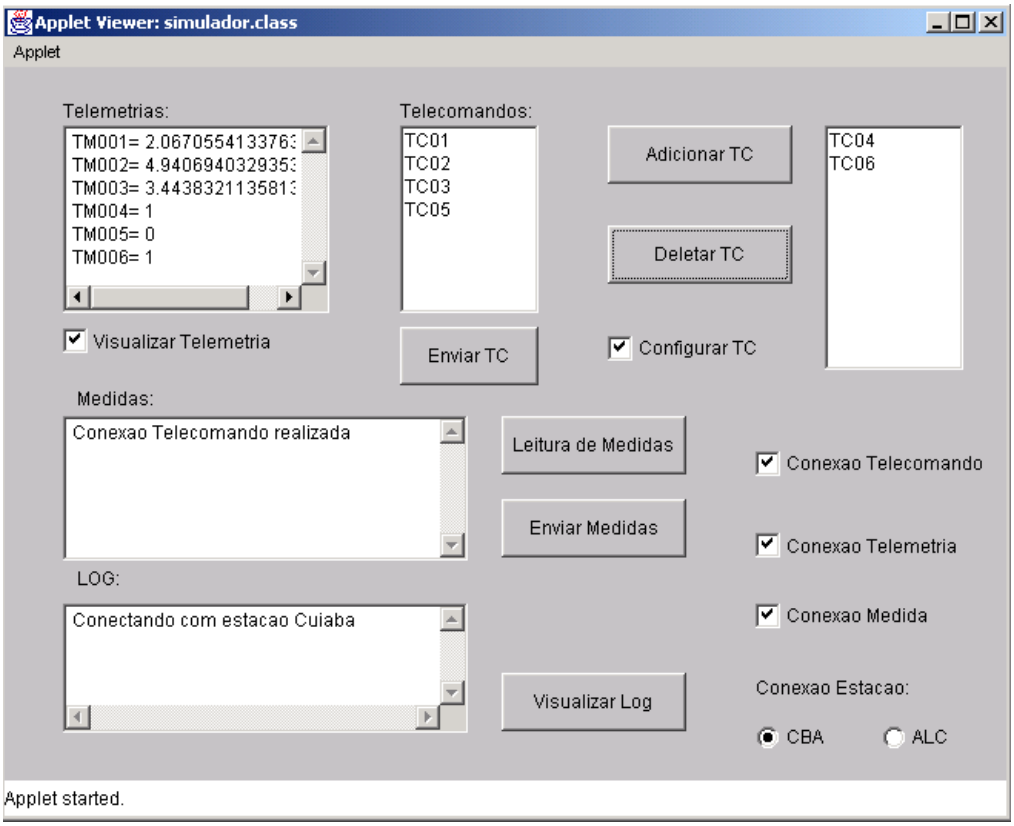

FIGURA 4.7 - Tela do protótipo do Software Simulador.

Estes comandos correspondem aos subsistemas do protótipo do Software Simulador e se resumem em:

- Visualizar Telemetrias: quando este comando é ativado *frames* de telemetrias analógicas e digitais são constantemente apresentadas na tela de telemetrias. Uma condição para que este comando possa ser acionado é que a conexão de telemetria deve estar realizada.

- Enviar Telecomandos: seleciona-se um telecomando da tela de telecomandos, para que o mesmo possa ser enviado, no momento de atuação do telecomando enviado, automaticamente percebe-se que a

telemetria digital apresentará novo valor, a condição de conexão de telecomando também deve ser satisfeita.

- Enviar Medidas: após a conexão de medidas, pode-se acionar este comando e então observar o envio de medidas de distância. As 32 medidas enviadas em grupos de 3 quadros que podem ser observados na tela de medidas.

- Configurar Telecomandos: permite-se configurar a tela telecomandos adicionando ou removendo telecomandos pré-estabelecidos. Basta selecioná-los na tela de telecomando e acionar os botões adicionar ou deletar.

- Conexão de Estação: apresenta-se as opções para que sejam selecionadas as estações de Cuiabá ou Alcântara.

- Visualizar Log: as operações já realizadas podem ser observadas quando necessário na tela de log.

4.2.1 - O Software Gerenciador de Cenários

Como já mencionado anteriormente, as inúmeras formas de se distribuir os objetos são gerenciadas através de um software, denominado gerenciador de cenários, construído especificamente para esta finalidade.

O gerenciador de cenários consiste em um aplicativo, desenvolvido em Visual Basic 6, da Microsoft, cuja função principal é inserir ou excluir na base de configuração os objetos envolvidos em cada cenário, obedecendo a prévia seleção do usuário que determinará quais objetos deverão ser criados e em quais máquinas. Do resultado da seleção de um determinado cenário, os objetos constituintes deste cenário são inseridos na base de configuração. É

então, permitida a criação ou destruição de instâncias destes objetos, através do Software Simulador de Satélites, nas máquinas disponíveis.

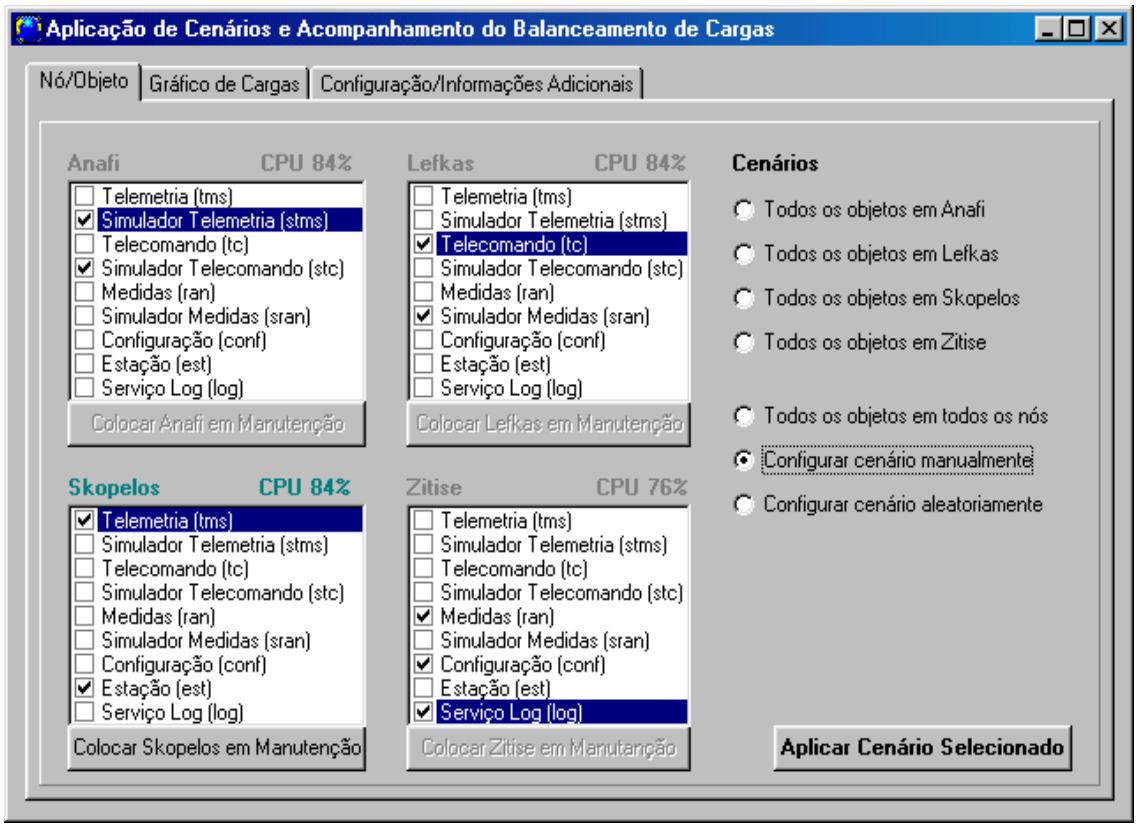

FIGURA 4.8 - Tela de seleção de cenários do software gerenciador de cenários.

Na Figura 4.8, pode ser observada a tela “Nós/Objetos”. Esta tela apresenta 4 janelas que possuem o nome das máquinas utilizadas no protótipo. Em todas estas quatro janelas, estão disponíveis opções para possível seleção de todos os objetos. Para realizar a seleção basta escolher um dos cenários disponíveis do lado direito da tela, ou realizar uma seleção personalizada, selecionando-se manualmente cada objeto desejado. Na Figura 4.8, selecionou-se manualmente a distribuição dos objetos nas máquinas constituindo o cenário como mostra a Tabela 4.1:

TABELA 4.1 – Cenário escolhido.

Máquinas	Objetos
Anafi	simulador de telemetria, simulador de telecomando
Lefkas	telecomando, simulador de medidas
Skopelos	telemetria, estação
Zitise	medidas, configuração e serviço de log

Os cenários são pré-estabelecidos de acordo com a necessidade do Software Simulador em relação à modelagem dos objetos. Ao aplicar o cenário selecionado, o gerenciador de cenários atualiza automaticamente a base de configuração, alterando os objetos na tabela “NoObjeto”, como mostra a tabela da Figura 4.9:

FIGURA 4.9 – Tabela “NoObjeto” da base de configuração.

A Figura 4.9 apresenta um exemplo de como se apresenta a tabela “NoObjeto” da base de configuração, segundo cenário criado pela distribuição aleatória gerada na tela da Figura 4.8 apresentada anteriormente.

Assim, a base de configuração se encontra preparada para receber as futuras solicitações de informações do Software Simulador de Satélites, para que o mesmo consiga criar os objetos nos nós estabelecidos.

Atualmente, este gerenciador pode criar automaticamente, na base de configuração, os seguintes cenários:

- Todos os objetos em Zitise;
- Todos os objetos em Anafi;
- Todos os objetos em Lefkas;
- Todos os objetos em Skopelos;
- Objetos distribuídos por casos de uso – esta opção trata-se do modelo de distribuição baseada em casos de uso, apresentada em 3.3.2.1.
- Todos os objetos em todos os nós - esta opção trata-se do modelo de distribuição apresentado em 3.3.2.2, para se realizar uma distribuição baseada em Tolerância a Falhas;
- Objetos distribuídos aleatoriamente – é a opção de não se seguir nenhum critério previamente definido, apresentado em 3.3.2.3, para se realizar uma distribuição na forma aleatória.

Uma outra característica importante do Software Gerenciador de Cenários é a possibilidade de apresentar informações como: disponibilidade da CPU e o número de conexões entre máquina e objetos. Este software gera também um gráfico que informa evolução das disponibilidades de CPU das máquinas

utilizadas no decorrer do tempo, permitindo assim o acompanhamento de cada situação apresentada.

O Software Gerenciador de Cenários necessita residir em apenas uma das máquinas do ambiente de desenvolvimento apresentado.

4.2.2 - Base de Configuração

Como já citado, é na base de configuração que ficam armazenadas todas as informações necessárias para que os objetos possam ser criados nas máquinas disponíveis.

A base de configuração possui em suas tabelas a configuração da rede do ambiente de desenvolvimento do protótipo, ou seja, tabelas que relacionam máquinas e objetos, suportando assim a idéia dos cenários. A seguir, na Figura 4.10 observa-se o diagrama de entidade-relacionamento:

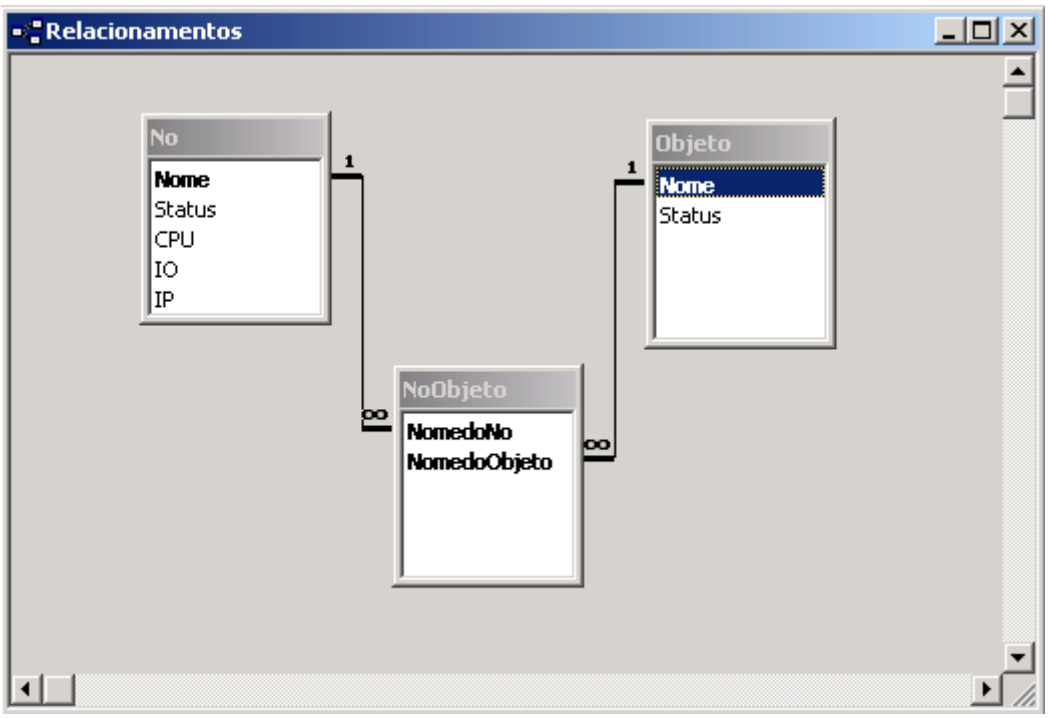

FIGURA 4.10 – Modelo entidade relacionamento.

De acordo com a necessidade do usuário ou da aplicação, é selecionado um cenário através do software gerenciador de cenários. Após realizada esta seleção, este software tem como responsabilidade inserir as informações relativas a este cenário selecionado na base de configuração.

Esta inserção se resume em atualizar a tabela “NoObjeto” com os nomes do objetos relacionando-os com os nomes dos nós em que os mesmos devem ser criados.

Como exemplo, selecionou-se o cenário “todos os objetos na máquina Zitise”. A tabela resultante desta seleção pode ser observada na Figura 4.11.

FIGURA 4.11 – Tabela da base de configuração “NoObjeto”.

A atualização da informação relativa à disponibilidade da CPU também é função do software gerenciador de cenários. É na tabela “No” que esta informação se encontra juntamente com: número IP, estado e I/O.

Na Figura 4.12, pode-se observar estes dados. Ainda considerando-se o cenário escolhido “Todos os objetos em Zitise”, é possível verificar que a disponibilidade da CPU se apresenta em 100% nas três máquinas ociosas, já em Zitise este valor se reduz para 28%.

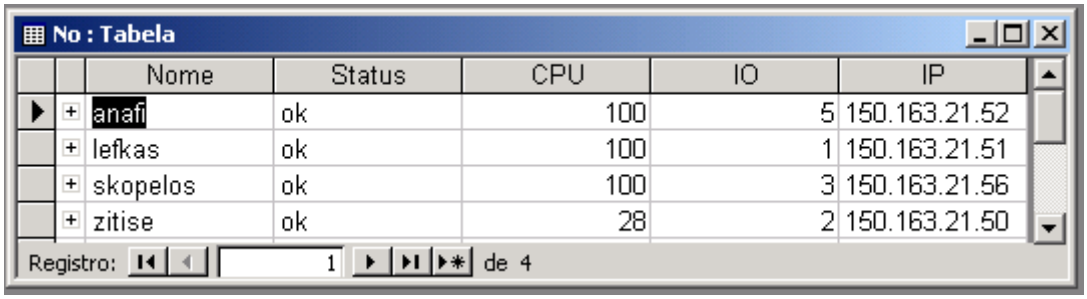

FIGURA 4.12 – Tabela da base de configuração “No”.

4.2.3 - Serviço de Carga

O serviço de carga, responsável pela criação dos objetos, também utiliza as informações da base de configuração. Sempre que ativado, sua primeira providência é acessar esta base a fim de verificar quais objetos devem ser criados.

Como citado anteriormente, o serviço de carga é responsável por esta verificação e criação dos objetos. Para cada máquina, este serviço verifica a situação da base de configuração, observando quais os objetos devem ser criados no nó local, em função do cenário selecionado. Após obter esta informação, o serviço de carga está apto a criar localmente somente os objetos destinados à própria máquina. Assim, o serviço de carga de uma determinada máquina é responsável pela carga dos objetos instanciados neste nó, ou seja, assim, cada máquina é responsável por criar seus objetos.

Tomemos como exemplo a máquina “Zitise”, o serviço de carga desta máquina será responsável por verificar a tabela “NoObjeto” na base de configuração, observando quais os objetos devem ser criados na máquina “Zitise”.

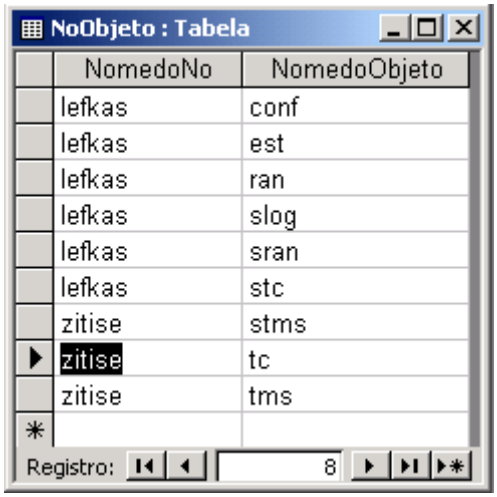

FIGURA 4.13 – Tabela “NoObjeto” .

Desta forma, o serviço de carga da máquina Zitise é responsável pela carga dos seguintes objetos: stms,tc e tms, que são os objetos destinados a serem criados neste nó, como ilustra a Figura 4.13.

4.2.4 - Serviço de Conexão

O serviço de conexão é responsável por obter a referência dos objetos necessários a atender as solicitações do Software Simulador de Satélites, quando um comando é acionado na interface deste Software.

O serviço de conexão localizará o nó em que se encontra este objeto acionando a base de configuração e solicitará a referência do mesmo disponibilizando-o para utilização.

O serviço de conexão também utiliza a base de configuração a fim de contabilizar as conexões iniciadas ou finalizadas entre objetos e máquinas na execução do Software Simulador. A tabela “Conexões”, mostrada na Figura 4.14, é responsável por auxiliar este serviço. Os campos origem e destino devem conter, respectivamente, o nó que solicitou o serviço e o nó onde o objeto se encontra instanciado.

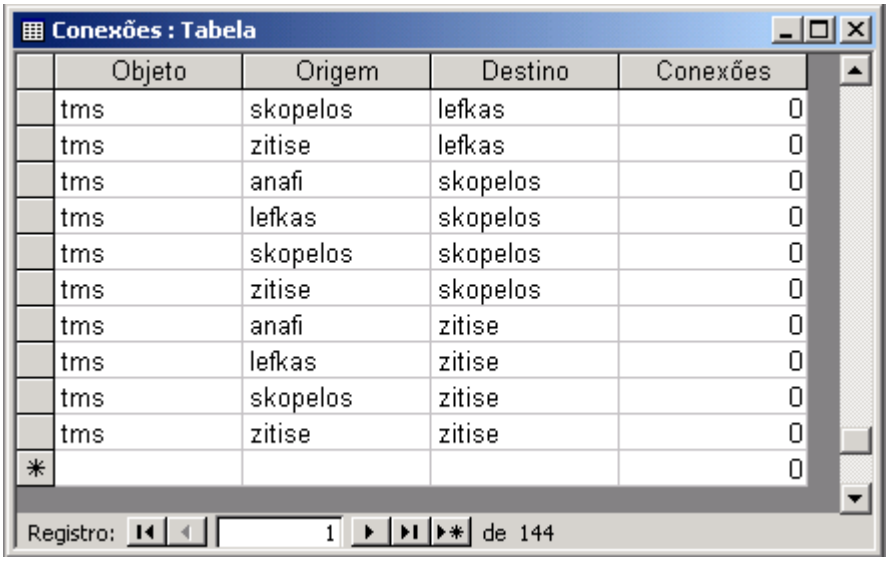

FIGURA 4.14 – Tabela da base de configuração “Conexões”.

A contabilização das conexões é necessária para se garantir o controle dos objetos criados e destruídos.

CAPÍTULO 5

IMPLEMENTAÇÃO DO SIMULADOR PROPOSTO

Para que fosse possível analisar, obter resultados e conclusões sobre os aspectos relacionados à distribuição e modelagens de objetos, já citados, optou-se por implementar um protótipo do Software Simulador de Satélites.

Para o desenvolvimento deste protótipo, foram selecionados alguns itens do Software Simulador de Satélites. Esta seleção se restringiu apenas às atividades consideradas relevantes ao objetivo do projeto. Desta forma, foram reproduzidos para o Software do Simulador de Satélites Distribuído somente os subsistemas fundamentais para que fosse possível:

- empregar a distribuição dos objetos;
- tentar eliminar as limitações do Software Simulador, advindas do ambiente centralizado;
- aplicar as modelagens apresentadas na Seção 3.3.2;
- realizar as observações necessárias à análise do comportamento dos objetos com relação à modelagem aplicada;
- realizar medições relacionadas a desempenho.

Neste Capítulo, são descritos os detalhes do ambiente em que foi desenvolvido o protótipo do Software Simulador de Satélites. Utiliza-se novamente dos recursos da UML a fim de complementar a modelagem dos aspectos estruturais e dinâmicos do sistema, já iniciados no capítulo anterior. Os aspectos principais da implementação do protótipo são explicados através de partes do código que auxiliam a sua compreensão.

5.1 - AMBIENTE DE DESENVOLVIMENTO

Para promover a distribuição dos objetos do protótipo do Software Simulador de Satélites, foi utilizado o padrão Java RMI, da SunMicrosystem, já apresentado na Seção 2.7 Para o desenvolvimento da interface gráfica do projeto em questão, utilizou-se o ambiente de desenvolvimento denominado PowerJ. O PowerJ é um aplicativo Java, que provê todos os recursos necessários para a criação das interfaces visuais. Como sistema de gerenciamento da base de configuração foi utilizado o Microsoft Access.

O ambiente utilizado constitui-se de 4 computadores interligados entre si, conectados à rede de 100Mbits do já citado Centro de Controle de Satélites do INPE.

Na Tabela 5.1 apresenta-se uma breve descrição da capacidade em termos de Hardware das máquinas utilizadas na implementação do projeto:

TABELA 5.1: Características das máquinas utilizadas no desenvolvimento.

Máquinas	Disco	Memória	Processador
Lefkas	20G	256M	AMD 1.4 GHz
Anafi	20G	256M	AMD 1.4 GHz
Zitise	20G	256M	AMD 1.4 GHz
Skopelos	20G	256M	AMD 1.4 GHz

Nestas máquinas os objetos são distribuídos de acordo com os vários cenários disponíveis que são, por sua vez, selecionados pelo usuário.

Na Tabela 5.2 apresenta-se uma contabilização dos números de classes e linhas de códigos necessários para a implementação do protótipo do Software Simulador de Satélites:

TABELA 5.2 – Contabilização dos códigos desenvolvidos.

Atividade	Número de Classes Desenvolvidas	Número de Linhas de Código
Simulador	10	1412
Serviço de Conexão	1	1056
Serviço de Carga	1	246
Gerenciador de Cenários	-	597
Total	**12**	**3311**

5.2 - REALIZAÇÃO DOS CASOS DE USO DO SIMULADOR

O objetivo deste tópico é apresentar a realização dos casos de uso apresentados na Seção 4.1.3. Representa-se assim, em nível dinâmico o comportamento dos elementos do Software Simulador de Satélites através do diagrama de colaboração. Este diagrama pode ser observado na Figura 4.2 na Seção 4.1.4.

As colaborações possuem dois aspectos: uma parte estrutural que especifica as classes, interfaces e outros elementos que trabalham em conjunto para executar a colaboração e a parte comportamental que especifica a dinâmica de como estes elementos interagem. (Booch et al.,1999)

Assim, através do diagrama de classes, que representa o aspecto estrutural, e do diagrama de seqüência, que apresenta a parte comportamental das colaborações, pode-se apresentar exemplos da realização de algumas das

colaborações apresentadas. Para fins ilustrativos selecionou-se uma das duas funções mais importantes do Software Simulador: enviar telecomando e visualizar telemetrias. O diagrama de classes foi apresentado na Seção 4.1.4 do capítulo anterior.

5.2.1 - Realização do Caso de Uso - Enviar Telecomando

O operador deve selecionar um telecomando específico para ser enviado, este telecomando, por sua vez, está relacionado a uma determinada telemetria. Um telecomando pode ser enviado para ligar uma chave, desligar um sensor, enfim, alterar o estado de algum equipamento. Qualquer alteração realizada no satélite irá refletir automaticamente nas telemetrias. Na Tabela 5.3, observa-se o relacionamento existente entre telecomandos, telemetrias e resultados de suas ações:

TABELA 5.3 - Valores de Telemetrias.

Telecomandos	Telemetrias/Valores	Ação
TC01	TMS04 = “1”	Ligada
TC02	TMS04 = “0”	Desligada
TC03	TMS05 = “1”	Ligada
TC04	TMS05 = “0”	Desligada
TC05	TMS06 = “1”	Ligada
TC06	TMS06 = “0”	Desligada

O diagrama de seqüência é um diagrama de interação que enfatiza o tempo, ordenando as mensagens; o diagrama de colaboração é um diagrama de interação que enfatiza uma organização estrutural dos objetos que enviam e recebem mensagens. Os diagramas de interação são utilizados para modelar aspectos dinâmicos do sistema (Booch et al., 1999).

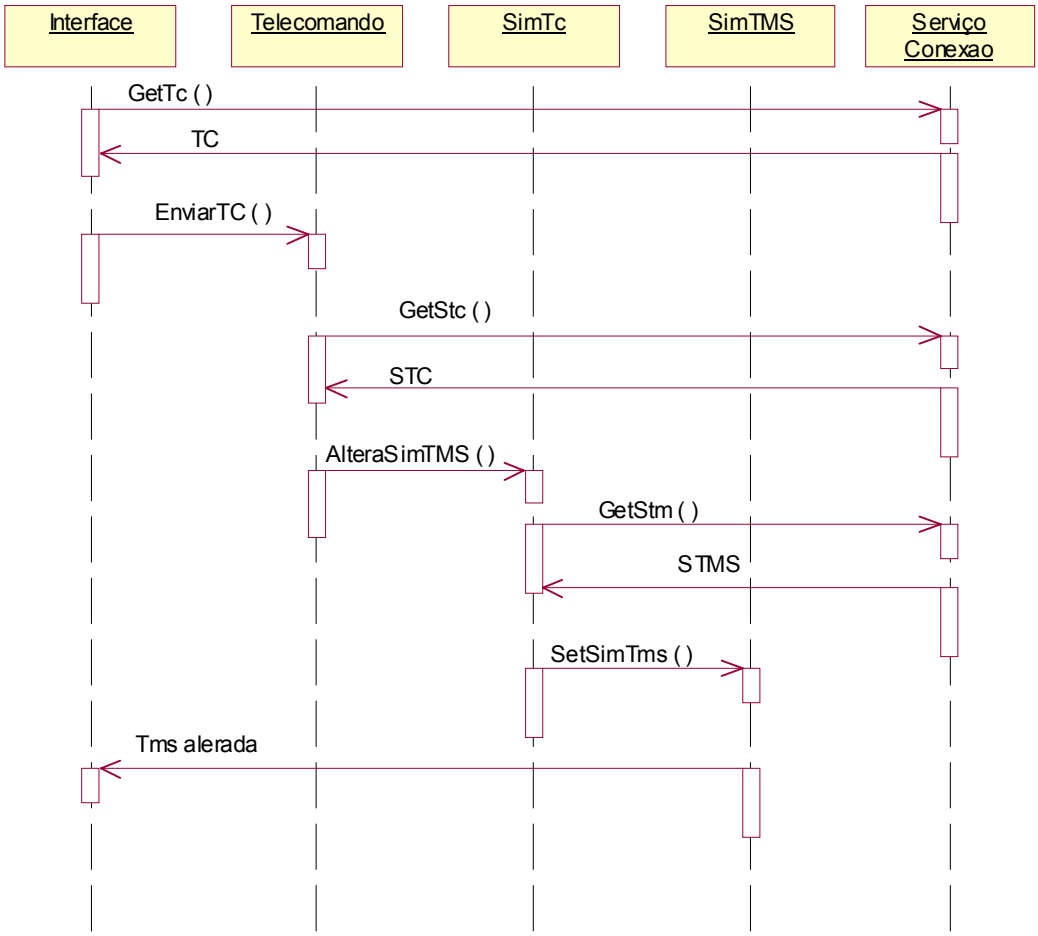

FIGURA 5.1 – Diagrama de seqüência: enviar telecomando.

Como já mencionado no capítulo anterior, quando o caso de uso “enviar telecomando” é ativado, o telecomando selecionado é enviado para atuação. Neste momento, o valor da telemetria digital relacionada a este telecomando, se altera automaticamente, indicando a atuação do mesmo.

Na interface humana do Software Simulador de Satélites, ao se selecionar qualquer um dos comandos disponíveis, deve-se primeiramente localizar os objetos envolvidos no caso de uso referido. Assim, quando se aciona o comando “Enviar Telecomando” deve-se acessar o serviço de conexão, que é o serviço responsável por localizar a referência do objeto telecomando “tc”.

O serviço de conexão consegue a referência do objeto telecomando “tc”, através do método “getTc()”.

```
....
        bc = (IServicoConexao)Naming.lookup("//localhost/ServicoConexao");
        tc1=bc.getTc(System.getProperty("user.name"),"",0);
....
```

No método “getTc()” adquire-se a referência do objeto telecomando “tc” através do método “lookup()”, que retorna a referência ao objeto que está associado a um nome especificado, no caso, este nome é “Telecomando”:

```
....
tc = (ITelecomando) Naming.lookup ("//" + ip +  "/Telecomando");
....
```

Assim, já é possível acessar o método “enviarTC ()” do objeto TC:

```
....
tc1.enviarTC(lb_tc.getSelectedItem());
 ....
```

Neste método acessa-se novamente o serviço de conexão, agora, para se obter a referência do objeto “stc”.

```
...
        bc = (IServicoConexao)Naming.lookup("//localhost/ServicoConexao");
        stc = bc.getStc(System.getProperty("user.name"), "tc", id);
....
```

Realiza-se uma conexão com este objeto para acessar o método “alterarSimTms()” da classe “Simulador Telecomando”.

```
...
        String conexao = stc.conexaoSimTC();
        stc.alterarSimTms(valor);
...
```

No método “alterarSimTms”, através do serviço de conexão, consegue-se a referência do objeto “stms”:

```
....
        bc = (IServicoConexao)Naming.lookup("//localhost/ServicoConexao");
        stms = bc.getStms(System.getProperty("user.name"), "stc", id);
....
```

E finalmente, através do método “setSimTms()”, altera-se o valor da telemetria:

```
...
        stms.setSimTms(valor);
....
...
    //Para TC's digitais
        if (valor.equals("TC01")) TM[04] = "1";
        if (valor.equals("TC02")) TM[04] = "0";
        if (valor.equals("TC03")) TM[05] = "1";
```

```
        if (valor.equals("TC04")) TM[05] = "0";

        if (valor.equals("TC05")) TM[06] = "1";

        if (valor.equals("TC06")) TM[06] = "0";
...
```

5.2.2 - Realização do Caso de Uso - Visualizar Telemetria

Mais simples que o método “enviar telecomando”, o método “visualizar telemetria” é responsável por apresentar os valores das telemetrias analógicas e digitais do Simulador de Satélites. Os valores das telemetrias são atualizados automaticamente sempre que um telecomando atuar.

Na Figura 5.2, é apresentado o diagrama de seqüência do método visualizar telemetria:

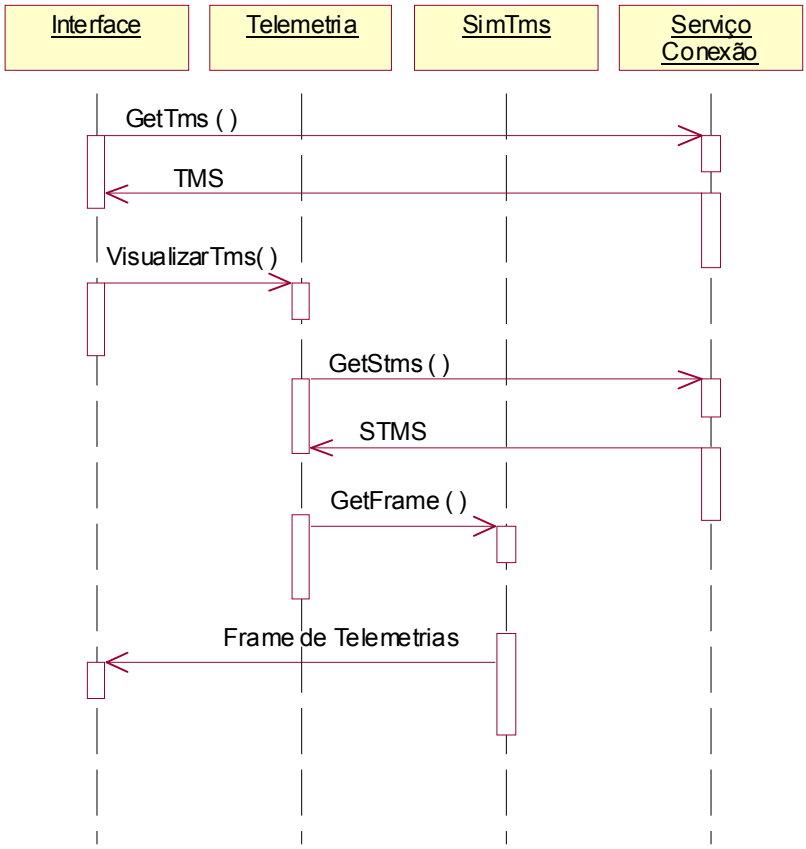

FIGURA 5.2 – Diagrama de seqüência: visualizar telemetria.

A interface disponibiliza recursos para que o usuário possa acionar o comando “Visualizar Telemetria”. Da mesma forma que no método “enviar telecomando”, deve-se primeiramente haver uma chamada ao serviço de conexão para se obter a referência do objeto “tms”. O acionamento do método “visualizar telemetria” é realizado pela chamada demonstrada a seguir:

....

```
String message;

tms1 = bc.getTms(System.getProperty("user.name"), "",0);

message=tms1.visualizarTMS();
```

Posteriormente, é necessário utilizar novamente o serviço de conexão para se obter a referência do objeto simulador telemetria “stms”, invocando para tanto o método “getStms()”:

....

```
bc = (IServicoConexao)Naming.lookup("//localhost/ServicoConexao");

stms = bc.getStms(System.getProperty("user.name"), "tms", id);
```

....

De posse da referência do objeto, já é possível acionar o método responsável por obter as telemetrias:

....

```
String f[ ] = new String[7];

f = stms.getFrame();
```

....

5.3 - DIAGRAMA DE CLASSES DO SIMULADOR

Na Seção anterior detalhou-se dois exemplos de casos de uso do protótipo do Software Simulador: “enviar telecomando” e “visualizar telemetria”, que foram selecionados apenas para fins explicativos. As classes que são utilizadas para a colaboração dos outros casos de uso são apresentadas nesta seção, através do diagrama de classes.

O diagrama de classes envolve a identificação dos itens considerados importantes para o sistema. Assim, para se ter uma idéia geral do Simulador, ilustrando todas as classes consideradas importantes para o projeto,

apresenta-se o seu diagrama de classes completo, com todos os atributos e métodos e seus relacionamentos:

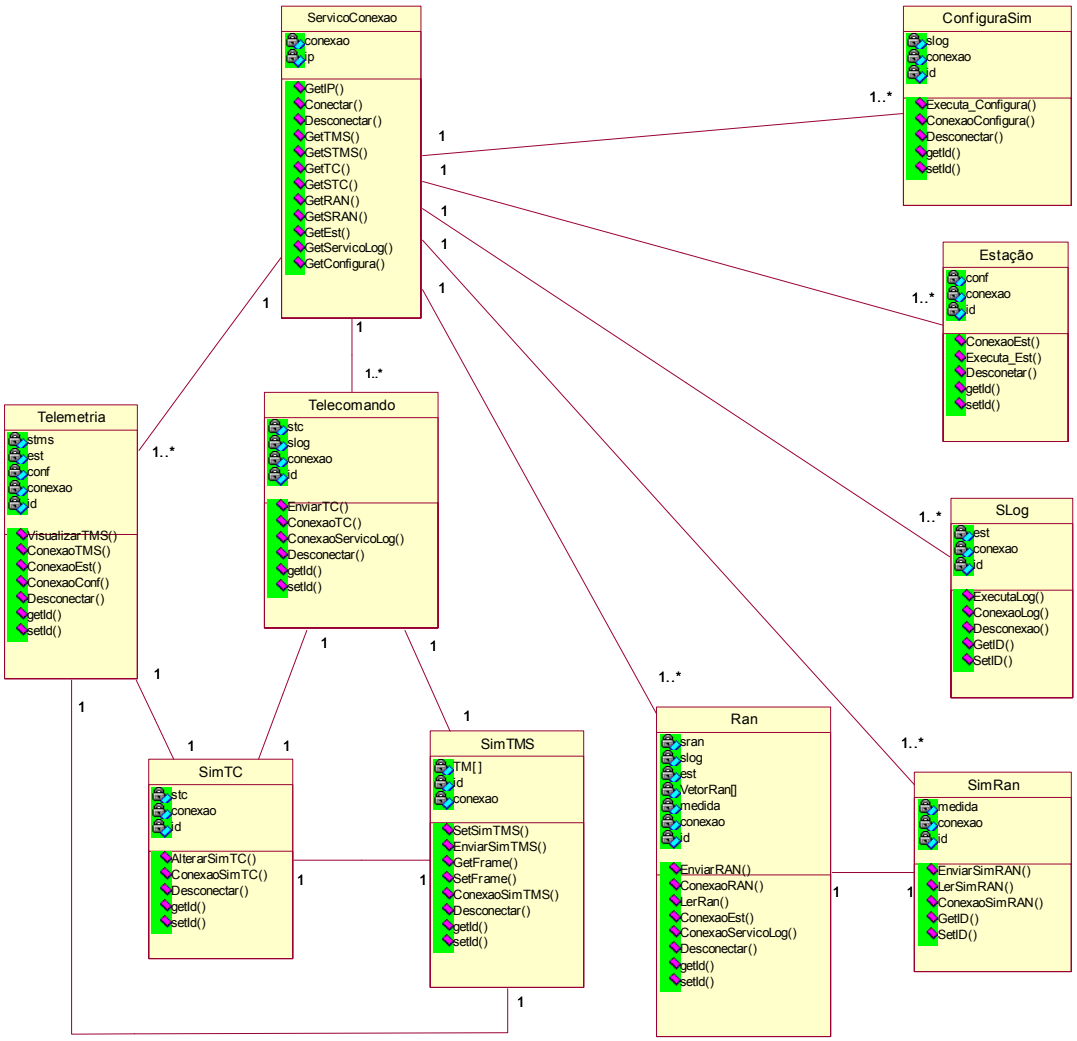

Fig. 5.3 – Diagrama de classes do Software Simulador de Satélites.

5.4 - IMPLEMENTAÇÃO DO PROTÓTIPO DO SOFTWARE SIMULADOR DE SATÉLITES

Para se trabalhar com o padrão RMI, alguns procedimentos devem ser obedecidos. Alguns destes procedimentos merecem uma explicação mais detalhada, como é o caso da criação de interfaces, criação de objetos e invocação dos métodos remotos.

Esta seção destina-se a explicar a implementação do projeto, expondo detalhes importantes do código e a utilização do padrão de distribuição utilizado.

O *middleware*, como já citado na seção 2.3, possibilita objetos serem acessados remotamente por outros serviços ou aplicações. Uma aplicação distribuída utilizando Java RMI é constituída de interfaces e classes. As interfaces definem as características e o comportamento dos objetos, incluindo as operações que podem ser requisitadas a esses objetos (Ferreira, 2001). As classes implementam os métodos definidos na interface e também podem definir métodos adicionais.

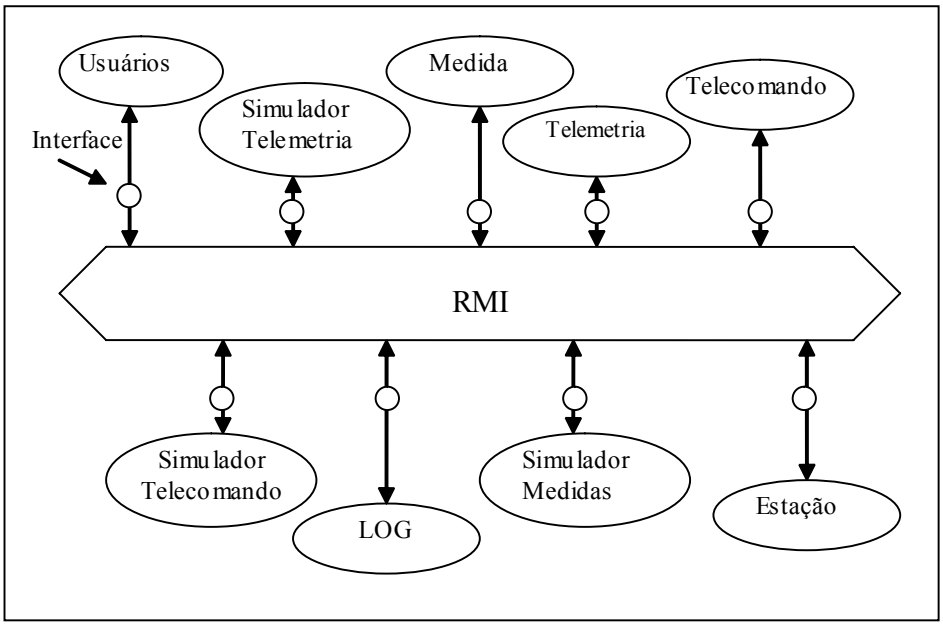

FIGURA 5.4 – A comunicação dos objetos na arquitetura do Software Simulador.

Na Figura 5.4 pode se observar os objetos que compõem o Software Simulador se comunicando através do *middleware* RMI da arquitetura Java. O usuário pode solicitar serviços desta arquitetura, sendo que os objetos são responsáveis por atender a estes serviços.

No caso do Software Simulador, os objetos estão intimamente ligados entre si. Por exemplo, quando um serviço “Visualizar Telemetria” é solicitado pelo cliente, o objeto “Telemetria” é acionado, porém ele se encarrega de solicitar ao objeto “Simulador de Telemetrias” os *frames* de telemetrias necessários à visualização. Toda esta interação entre objetos é permitida através das informações contidas na interface. Através da interface, o cliente pode conhecer quais operações e respectivos argumentos estão disponíveis para serem solicitados.

A distribuição destes objetos, exibidos na Figura 5.4, entre os 4 nós pré destinados para o ambiente depende dos cenários que por sua vez é

selecionado pelo desenvolvedor. Por exemplo, se for escolhido o modelo de distribuição na forma aleatória, então, deve-se distribuir aleatoriamente entre as quatro máquinas, todos os objetos apresentados na Figura 5.4.

Apesar da Figura 5.4 não explicitar, pode existir mais de uma cópia de um objeto.

5.4.1 - Interfaces

Utilizando-se o padrão RMI, antes de se criar um objeto remoto deve-se primeiro definir uma interface que estenda a interface “java.rmi.Remote”. Um objeto de uma classe que implementa essa interface é um objeto que pode ser acessado remotamente (Bernadino, 2001). Assim, a interface especifica quais métodos o objeto apresenta para serem invocados.

Desta forma, uma das primeiras providências para o desenvolvimento do Software Simulador foi criar as interfaces para todos os objetos envolvidos. Para fins ilustrativos, a seguir, apresenta-se o código da interface do objeto telemetria:

```
public interface ITelemetria extends Remote
{
    public String visualizarTMS() throws RemoteException;
    public String conexaoTMS() throws RemoteException;
    // metodo de apoio para realizar medições
    public String conexaoConf()throws RemoteException;
    public String conexaoEst()throws RemoteException;
    //Metodos de gerenciamento das conexoes entre objetos
```

```
        public void Desconectar()throws RemoteException;

        public long getId() throws RemoteException;

        public void setId(long i) throws RemoteException;

    }
```

Com a interface criada, pode-se então partir para a criação dos objetos em questão.

5.4.2 - Criando os objetos

Como já citado, conectando-se a base de configuração, o serviço de carga verifica quais os objetos foram selecionados para serem criados localmente. Sua próxima função é então, criar estes objetos nas respectivas máquinas. Para esta finalidade dispõe-se de uma facilidade do RMI. O RMI utiliza uma ferramenta de nomeação chamada “rmiregistry” que roda na máquina servidora e implementa um “Serviço de Nomes”, que mantém registrados todos os objetos localizados no nó com seus respectivos nomes, tornando acessível a referência destes objetos ao cliente. Assim, toda vez que o servidor é inicializado, ele chama o sistema de registro para associar nomes com os objetos, utilizando o método “bind()” da classe “Naming” do pacote “java.rmi” para associar um nome no “registry”. Na Figura 5.5, observa-se a interação entre cliente, servidor e registro:

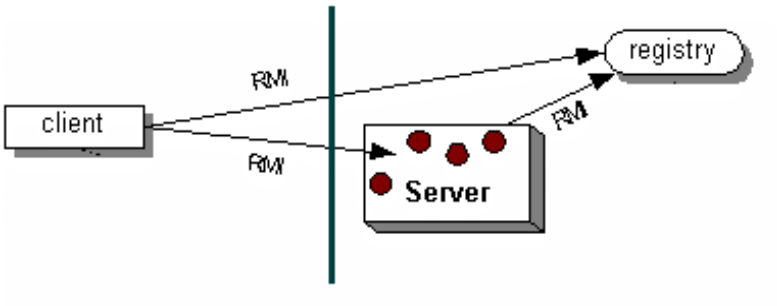

FIGURA 5.5 - Interação entre cliente, servidor e registro no RMI.

O método “bind()” associa o nome ao objeto remoto; a seguir o código desta operação é apresentado. Se o nome já está associado com o objeto, uma exceção é lançada. Já o método “rebind()” também associa o nome com o objeto remoto, porém substitui qualquer associação já existente a este nome, ou seja qualquer “bind()” anterior é descartado:

...

```
TelemetriaImpl tms = new TelemetriaImpl();

Naming.rebind("Telemetria", tms);

System.out.println("Objeto Telemetria adicionado ao registro");
```

...

Assim, foi criada uma instância do objeto “tms”. Esta instância está ligada ao objeto telemetria no registro de nomes, através do nome “Telemetria”, como pode ser observado no código apresentado.

5.4.3 - Invocando os objetos

Após a criação dos objetos, os métodos do Software Simulador de Satélites já podem ser ativados. Estes métodos, já citados na Seção 4.2, são acionados pelos comandos através da interface.

Quando um destes comandos é acionado, o RMI atua novamente, agora para se localizar o objeto. O cliente Java RMI adquire uma referência do objeto através do método “lookup()”, que retorna a referência ao objeto que está associado ao nome especificado. O método “lookup()” está localizado na classe serviço de conexão. Como já citado anteriormente, a classe serviço de conexão é a classe responsável por obter as referências de todos os objeto. A seguir, a obtenção da referência do objeto “stms” pode ser observada:

```
...

        String f[ ] = new String[7];

        stms = (ISimTms)Naming.lookup("//" + ip + "/STelemetria");

        f = stms.getFrame();

...
```

No código apresentado, percebe-se a chamada ao método “lookup()” da classe “Naming” do pacote “java.RMI”. Obteve-se então, uma referência remota a este objeto, cujo nome é “STelemetria”, que está armazenado no “registry”. Esta referência é armazenada no objeto “stms”, a partir deste momento, pode-se executar o método “getFrame()”.

CAPÍTULO 6

TESTES E RESULTADOS

O objetivo deste Capítulo é exibir os resultados obtidos através de testes realizados para que seja possível levantar as conclusões necessárias sobre a dissertação aqui apresentada. Primeiramente, exibem-se os resultados sobre as eliminações das limitações do Software Simulador. Os demais resultados, correspondentes às modelagens da distribuição também podem ser apreciados neste capítulo. Medições foram executadas a fim de realizar um estudo comparativo com relação a desempenho e disponibilidade apresentados pela modelagem de distribuição.

6.1 – RESULTADOS RELATIVOS À ELIMINAÇÃO DAS LIMITAÇÕES DO SOFTWARE SIMULADOR DE SATÉLITES

Com a finalidade de eliminar as limitações citadas na seção 3.3 do presente trabalho, aplicando-se a distribuição de objetos, pode-se observar os seguintes resultados, com relação à:

Disponibilidade e Tolerância a Falhas: verificou-se um aumento na disponibilidade no protótipo do Software Simulado distribuído, como pode se observar na experiência ilustrada na Figura 6.1 a seguir:

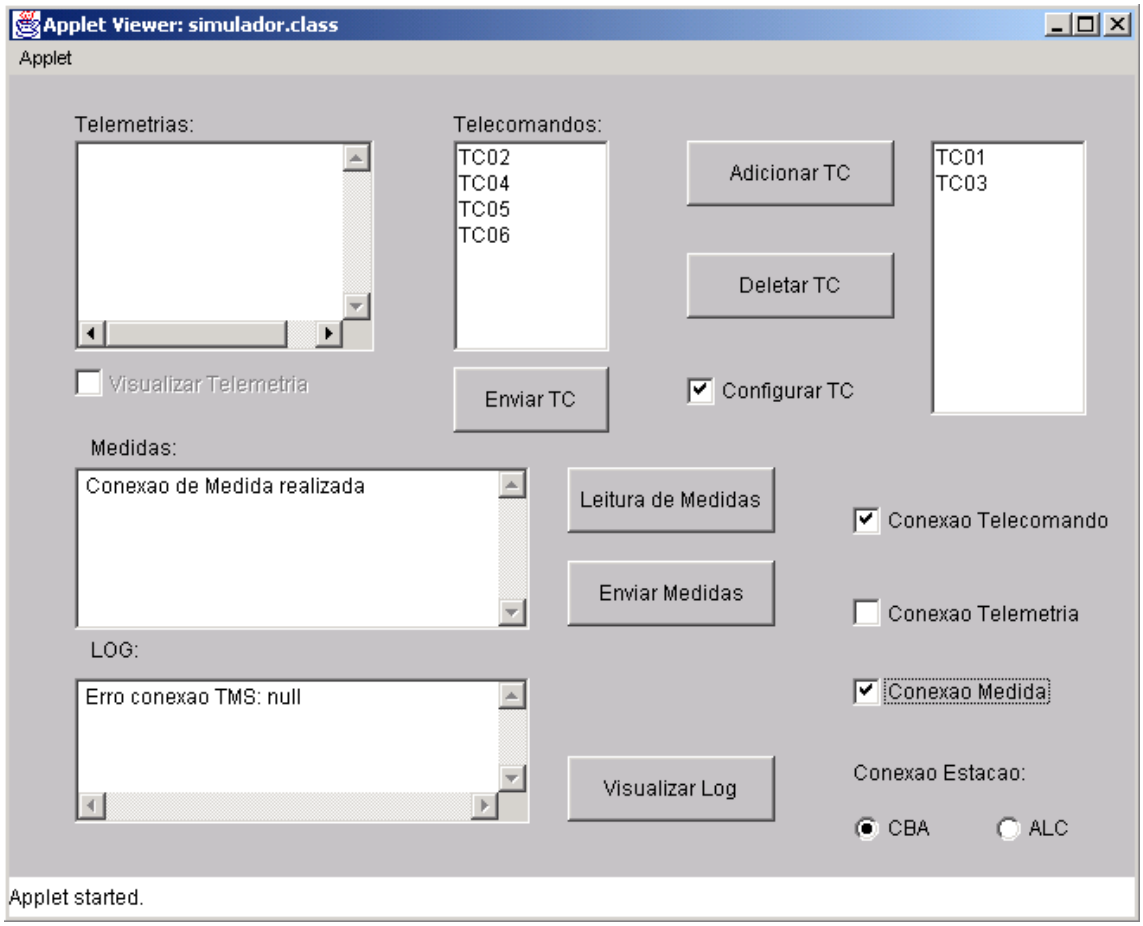

FIGURA 6.1 – Melhoria na disponibilidade do Simulador.

Nesta experiência observa-se que provocou-se uma falha no objeto Telemetria. Uma mensagem de erro pode ser observada na janela “LOG”. Observa-se que os outros subsistemas do Software Simulador, relacionados a outros objetos como, por exemplo, medidas e telecomando continuam disponíveis. O que mostra que diante de uma falha de um determinado objeto do Software, esta falha não acarreta a indisponibilidade de outros serviços.

Observou-se que o sistema realmente provê tolerância a falhas como pode ser observado no seguinte experimento ilustrado pela Figura 6.2 a 6.6:

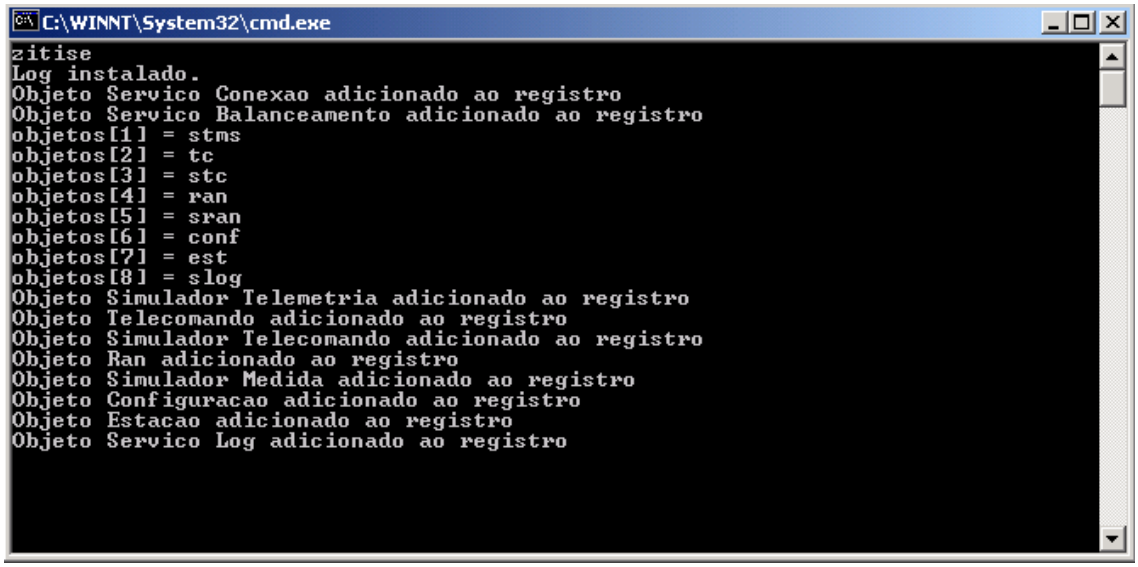

FIGURA 6.2 – Objetos criados na máquina Zitise.

A Figura 6.2 ilustra todos os objetos do simulador criados na máquina Zitise (a primeira linha da tela indica o nome da máquina em que os objetos estão sendo instanciados): stms, tc, stc, ran, sran,conf, est, slog. O objeto tms não foi criado nesta máquina. Na Figura 6.3 observa-se falha provocada em Lefkas, onde o objeto tms encontra-se instanciado.

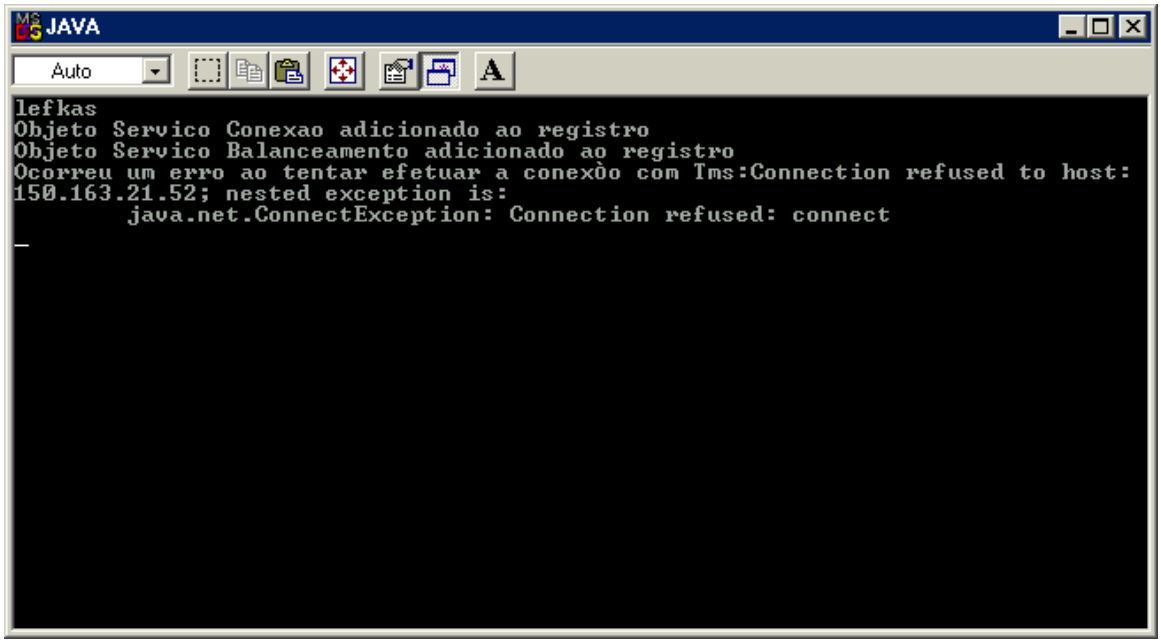

FIGURA 6.3 – Falha provocada em Lefkas: objeto tms instanciado.

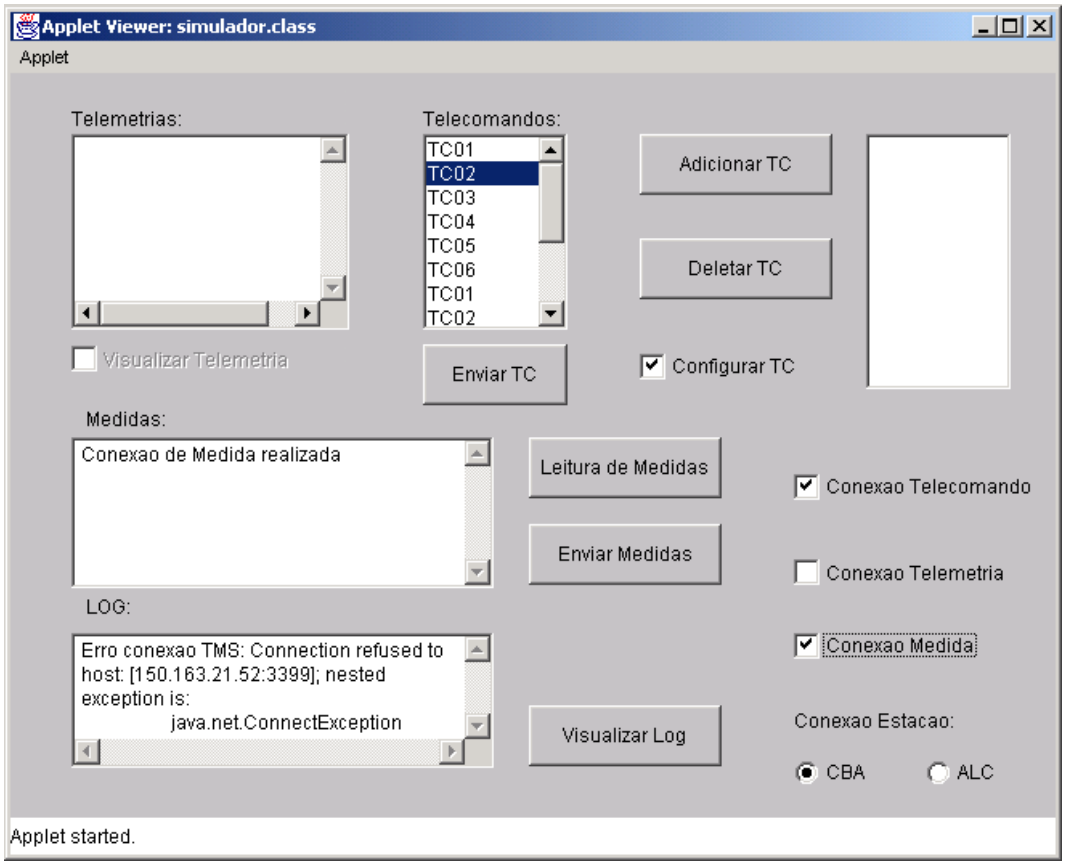

FIGURA 6.4 – Falha provocada.

A Figura 6.4 ilustra a conseqüência no Simulador da falha provocada no nó Lefkas com objeto tms. Observa-se a mensagem de erro de conexão na janela de “LOG”. Não é possível utilizar nenhum comando relacionado ao objeto tms.

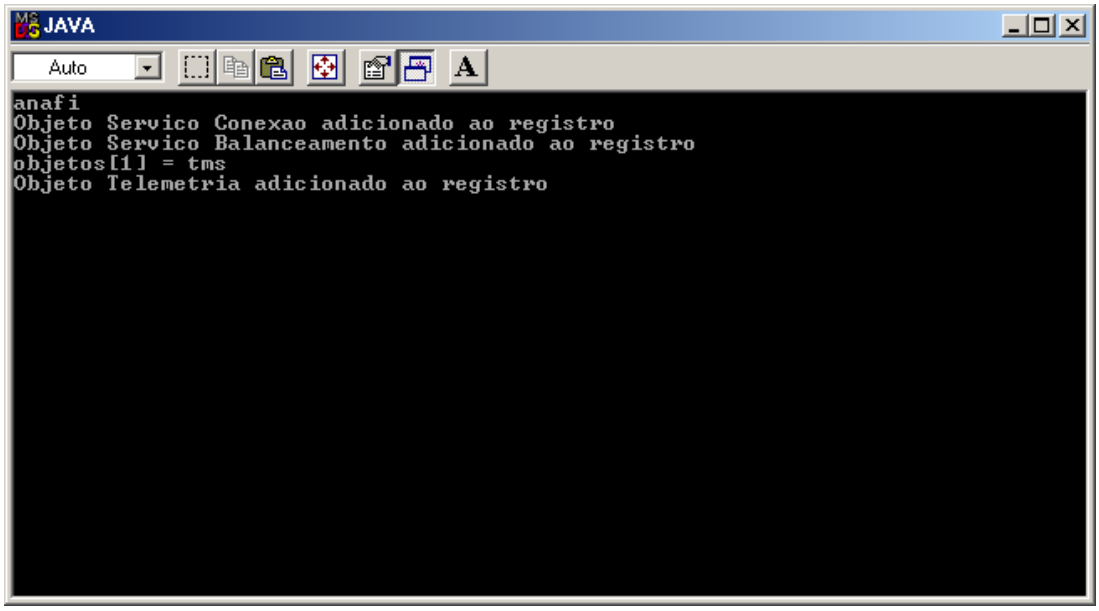

FIGURA 6.5 – Objeto criado na máquina Anafi.

Já na Figura-se 6.5 pode-se perceber uma nova conexão estabelecida com outro objeto equivalente na máquina Anafi. Observa-se que o objeto tms foi instanciado na máquina Anafi.

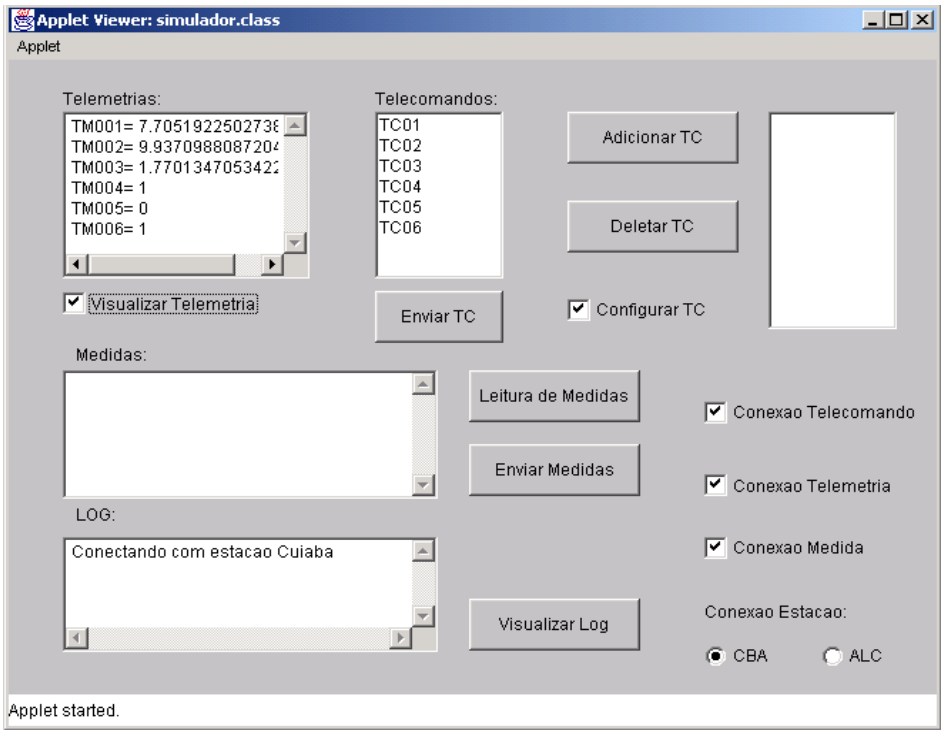

FIGURA 6.6 – Conexão restabelecida.

Observa-se a conexão de telemetria restabelecida na Figura 6.6, disponibilizando novamente a conexão e visualização de telemetria. Observa-se assim a existência de tolerância a falhas no Software Simulador Distribuído.

Concorrência: Dois ou mais usuários pode solicitar o mesmo serviço ao sistema, mas sendo atendidos por objetos instanciados em nós diferentes. Pode-se simultaneamente instanciar os objetos em várias máquinas como observado nas Figuras 6.7 e 6.8:

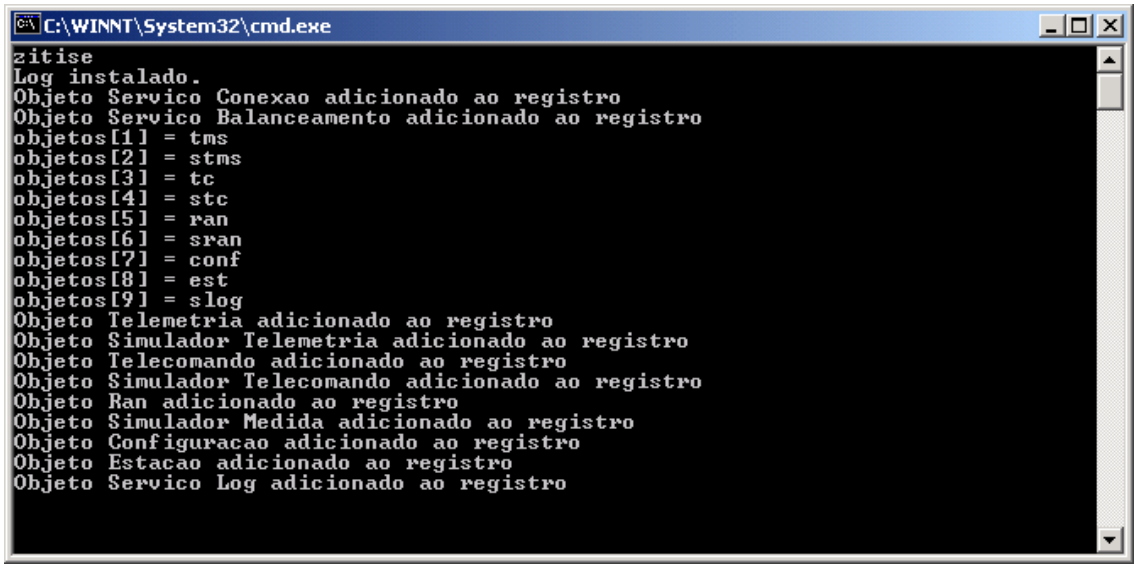

FIGURA 6.7 – Objetos instanciados em Zitise.

Na Figura 6.7 observa-se que os objetos: tms,stms,tc,stc,ran,sran,conf, est e log foram criados na máquina Zitise, como indica a primeira linha da tela que mostra o nome da máquina onde os objetos devem ser criados. O mesmo pode ser observado na Figura 6.8, onde os objetos são criados na máquina Anafi.

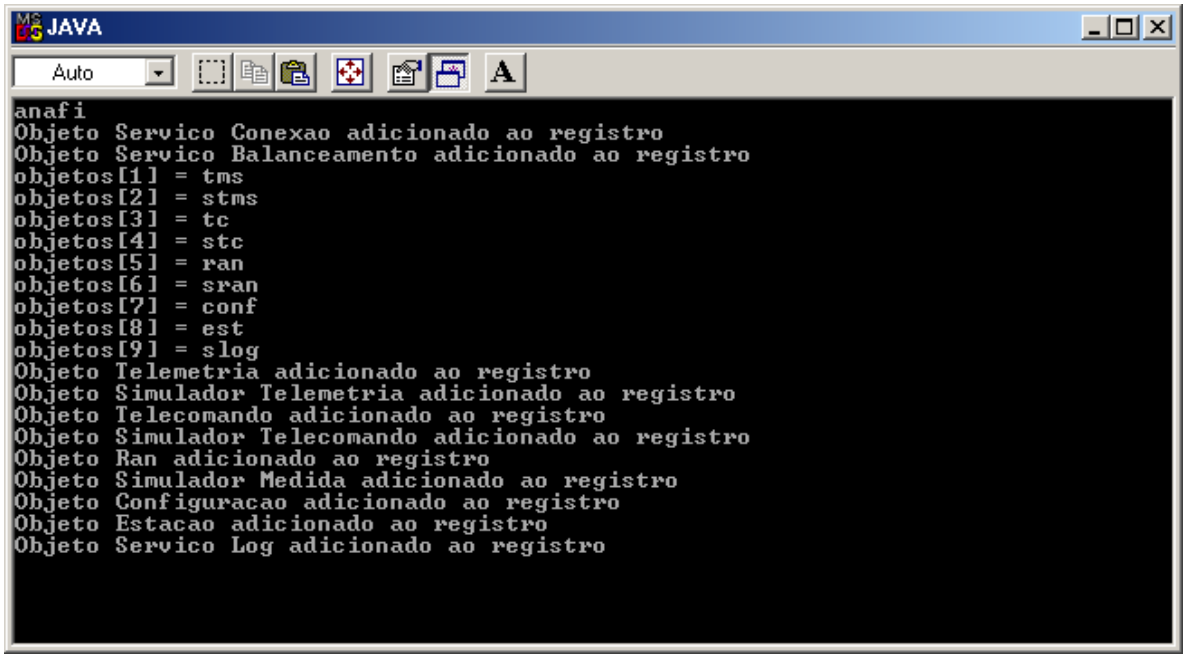

FIGURA 6.8 – Objetos instanciados em Anafi.

Flexibilidade: Observa-se uma maior flexibilidade para atender as diferentes situações de controle. É possível replicar totalmente o Software Simulador em uma outra máquina atendendo a possíveis necessidades de mais de um usuário, como observa-se na Figura 6.7 e 6.8 onde é possível instanciar todos os objetos simultaneamente em mais de uma máquina, podendo então replicar totalmente o Software Simulador.

6.2 – RESULTADOS RELATIVOS À MODELAGEM DA DISTRIBUIÇÃO

Várias medições foram realizadas a fim de se obter informações suficientes para que fosse possível realizar uma análise quantitativa, expressando através de gráficos, os resultado obtidos.

Como já citado, os resultados não têm como finalidade apontar qual é a melhor técnica de distribuição, pretende-se analisar os resultados referentes às características de desempenho e disponibilidade do sistema.

Para facilitar as medições, relacionou-se cada atividade, desenvolvida no protótipo, a ser medida a um número de experimento e assim, montou-se a tabela apresentada a seguir:

TABELA 6.1 - Experimentos.

Experimento	Atividade
1	Enviar Telecomando
2	Visualizar Telemetria
3	Enviar Medidas
4	Conectar Medidas
5	Conectar Telemetria
6	Conectar Telecomando

6.2.1 - Procedimento para Realização das Medidas de Desempenho

Antes de se detalhar o procedimento realizado para se obter os valores das medidas de desempenho, faz-se necessário uma breve explicação sobre a granularidade.

Segundo o dicionário da língua portuguesa Aurélio, granular significa ‘semelhante na forma ao grão’.

Assim, em relação a objetos, granularizar significa quebrar os objetos em objetos menores, de forma que a união destes objetos menores componha o objeto que os originou. Este conjunto de objetos menores deve ainda manter a mesma funcionalidade apresentada pelo objeto original.

A necessidade de se granularizar os objetos, neste trabalho, surgiu na realização dos testes práticos no Software Simulador. Observando-se os resultados, percebeu-se que os objetos que se colaboravam para realizar um caso de uso apresentavam desempenhos diferentes quando alocados localmente e remotamente, e este desempenho variava ainda mais quando se utilizava a técnica de modelagem aleatória, onde os objetos se encontravam dispersos pela rede. Surgiu então, o interesse em se observar como se apresentaria o comportamento destes objetos aumentando-se a granularidade dos mesmos, e distribuindo-os segundo as técnicas de modelagem aqui abordadas.

A Figura 6.9 ilustra a idéia anteriormente citada:

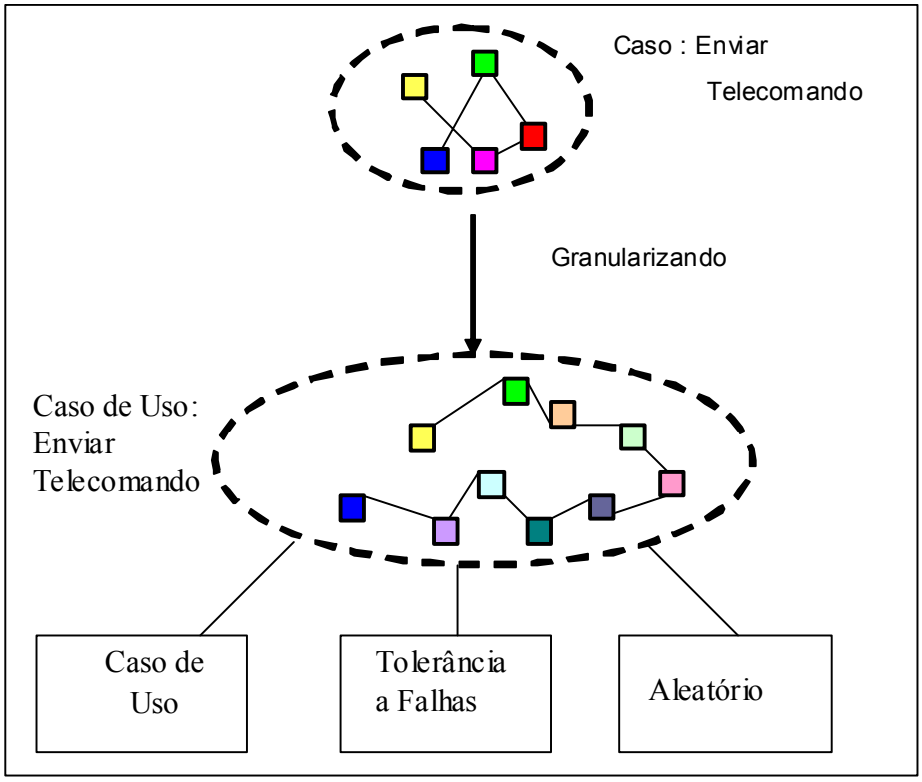

FIGURA 6.9 – Granularizando os objetos.

A Figura 6.9 ilustra objetos que se colaboram para realizar o caso de uso ”Enviar Telecomando”, a estes objetos pode-se aplicar a granularidade, ou

seja, quebrar estes objetos em objetos menores, porém mantendo-se a mesma funcionalidade. Assim, tem-se um conjunto maior de objetos para se colaborarem na realização do mesmo caso de uso “enviar telecomando”. O objetivo é observar como se comporta este novo conjunto de objetos, aplicando-se a eles, as técnicas de distribuição apresentadas neste trabalho, ou seja, como a granularidade do objeto pode interferir no desempenho de uma determinada tarefa.

Um exemplo pode ser observado na Figura 6.10, onde tem-se o diagrama de classes do caso de uso “visualizar telemetria”, antes e após a granularização. Observa-se que novos objetos surgem da associação entre os objetos Telemetria e Serviço Conexão:

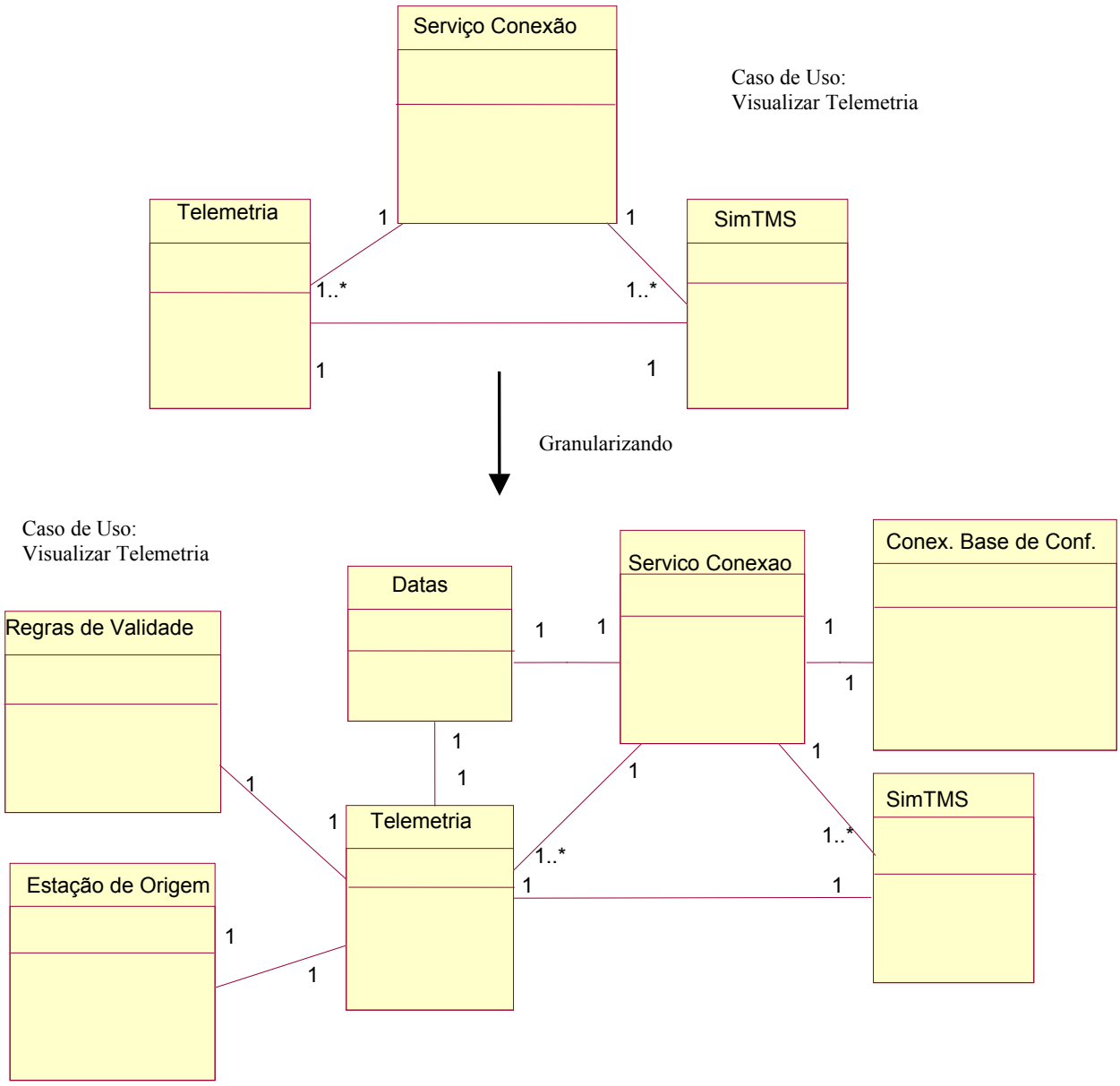

FIGURA 6.10 – Exemplo de Granularidade.

Na Figura 6.10 pode-se observar a aplicação da granularidade para os objetos Serviço Conexão e Telemetria. O objeto Serviço Conexão foi granularizado em objetos: Conex. Base Conf. (Conexão com Base de Configuração) e Datas. O

Objeto Telemetria foi granularizado em objetos: Regras de Validade, Estação de Origem e Datas.

Para realizar a conexão entre a base de configuração e o objeto ServicoConexão ou para obtenção de informações sobre datas deste objeto, antes da granularização, era necessário acessar métodos contidos na classe ServicoConexão. Da mesma forma, com relação ao objeto Telemetria, antes da granularização, as informações sobre validação, estação e datas poderiam ser obtidas através de métodos contidos na classe Telemetria.

Com a granularidade, estas atividades continuam sendo realizadas, porém agora, através de novos objetos específicos que foram criados para realizar estas funções. Por exemplo, para obtenção de uma data em que a telemetria foi recebida, pode-se acessar diretamente o novo objeto Datas, não precisando acessar um método do objeto Telemetria para acionar a obtenção de datas.

As medidas de desempenho foram realizadas da seguinte forma:

1) Para cada método do software simulador, coletou-se o tempo do sistema antes e depois do acionamento do método, inicialmente as medidas foram coletadas sem envolver a granularidade.

2) Com a finalidade se garantir uma melhor precisão nas medidas, optou-se por realizar uma média em um conjunto de 50 (número empírico) medidas, evitando-se imprecisões ocasionadas por perturbações momentâneas nas máquinas. Um exemplo de como se realizou as medições pode ser observado na chamada do método “enviarTC()”:

```
...
    String inicio=System.out.TimeMillis(); //Colhendo tempo inicial  em milisegundos
    For ( int i=0, i<j<i++)
            tc1.enviarTC();
```

```
String fim=System.out.TimeMillis();     //Colhendo tempo final em milisegundos

String dif= (fim-inicio)/j;             //Obtendo o intervalo

System.out.println(" Tempo medido = "+dif );

...
```

3) Utilizou-se os cenários: Casos de uso, tolerância a falhas e distribuição no modo aleatório.

4) Para cada cenário, executou-se todos os experimentos do simulador em cada máquina participante do projeto. Obteve-se as seguintes medidas de tempo, em milisegundos:

TABELA 6.2 – Tabela de medidas de tempo.

Simulador	Experimento	Cenários		
		Casos de Uso	Tolerância a Falhas	Aleatório
Zitise	1	186	199	238
	2	189	186	226
	3	185	186	265
	4	317	311	432
	5	194	197	211
	6	257	253	293
Anafi	1	182	198	250
	2	185	200	234
	3	182	189	293
	4	311	310	424
	5	192	200	196
	6	251	250	276
Lefkas	1	183	199	225
	2	184	209	216
	3	181	198	270
	4	308	334	416
	5	190	206	193
	6	248	270	304

(continua)

TABELA 6.2 – (conclusão).

Skopelos	1	183	200	227
	2	184	186	214
	3	180	185	288
	4	307	310	475
	5	186	196	192
	6	246	252	272

O gráfico da Figura 6.11 ilustra os resultado obtidos somente para os objetos originais, ou seja sem granularidade. A média pode ser verificada logo a seguir na Figura 6.12:

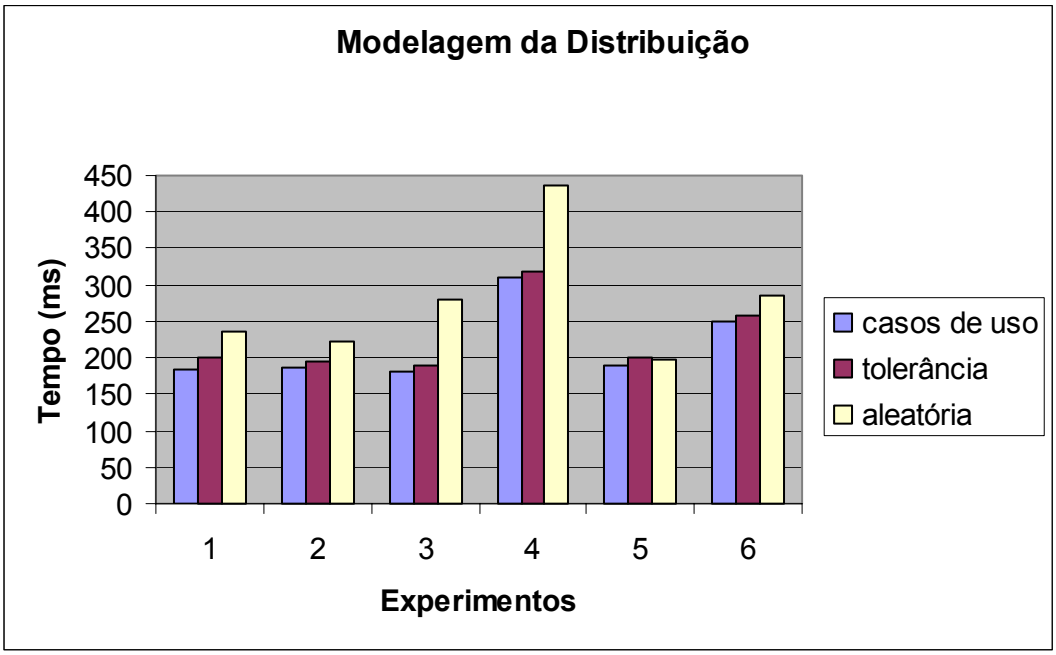

FIGURA 6.11 – Gráfico modelagem da distribuição.

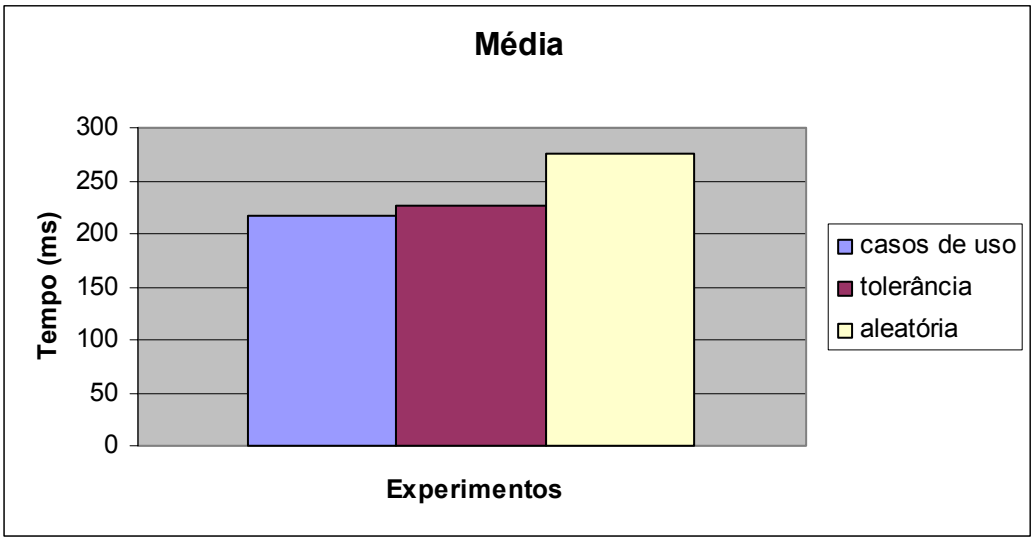

FIGURA 6.12 – Gráfico da média.

5) Repetiu-se as medidas aumentado-se a granularidade dos objetos em 5 e posteriormente em 10 objetos. O número de objetos originais que formam os casos de uso do Simulador varia de 3 a 5 objetos. Neste experimento, decidiu-se granularizar em 5 e 10 objetos. Neste contexto, granularizar em 5 objetos significa quebrar os objetos que estão envolvidos em um caso de uso do Software Simulador em mais 5 objetos. No caso de granularizar em 10 objetos significa quebrar os objetos envolvidos em um caso de uso do Software Simulador em mais 10 objetos. Este procedimento foi realizado para todos os casos de uso do software Simulador de Satélites. Salienta-se que, 5 e 10, foram valores escolhidos empiricamente a fim de satisfazer os experimentos.

Na Tabela 6.2, com o intuito de facilitar a observação e de realizar uma comparação entre os valores coletados, repete-se os valores apresentados anteriormente sem granularidade, especificados com a letra ‘S’. Os valores medidos aumentando-se a granularidade em 5 objetos são especificados em ‘+5’ e os com granularidade 10, em ‘+10’:

TABELA 6.3 – Tabela de medidas de tempo.

Simulador	Experimento	Cenários								
		Casos de Uso			Tolerância a Falhas			Aleatória		
		S	+5	+10	S	+5	+10	S	+5	+10
Zitise	1	186	290	294	199	327	325	238	273	712
	2	189	255	289	186	271	299	226	292	333
	3	185	252	280	186	291	416	265	495	674
	4	317	438	437	311	415	416	432	672	836
	5	194	284	324	197	301	326	211	288	359
	6	257	363	374	253	365	367	293	333	711
Anafi	1	182	268	292	198	330	474	250	305	836
	2	185	275	278	200	280	364	234	292	342
	3	182	291	279	189	290	379	293	504	784
	4	311	418	406	310	415	451	424	595	886
	5	192	285	295	200	310	321	196	267	335
	6	251	364	349	250	350	408	276	293	889
Lefkas	1	183	261	293	199	435	548	225	341	836
	2	184	284	281	209	301	362	216	353	365
	3	181	262	277	198	301	399	270	493	778
	4	308	387	404	334	420	491	416	591	983
	5	190	269	285	206	298	363	193	287	339
	6	248	356	343	270	357	425	304	385	773
Skopelos	1	183	281	293	200	278	338	227	267	860
	2	184	271	280	186	273	294	214	301	392
	3	180	238	279	185	254	293	288	493	797
	4	307	404	405	310	405	420	475	569	1025
	5	186	252	285	196	295	327	192	279	334
	6	246	340	350	252	350	365	272	335	867

A seguir, observa-se na Figura 6.13, o gráfico dos resultados granularizando-se os objetos originais em 5 objetos. O gráfico da média pode ser verificado logo a seguir na Figura 6.14:

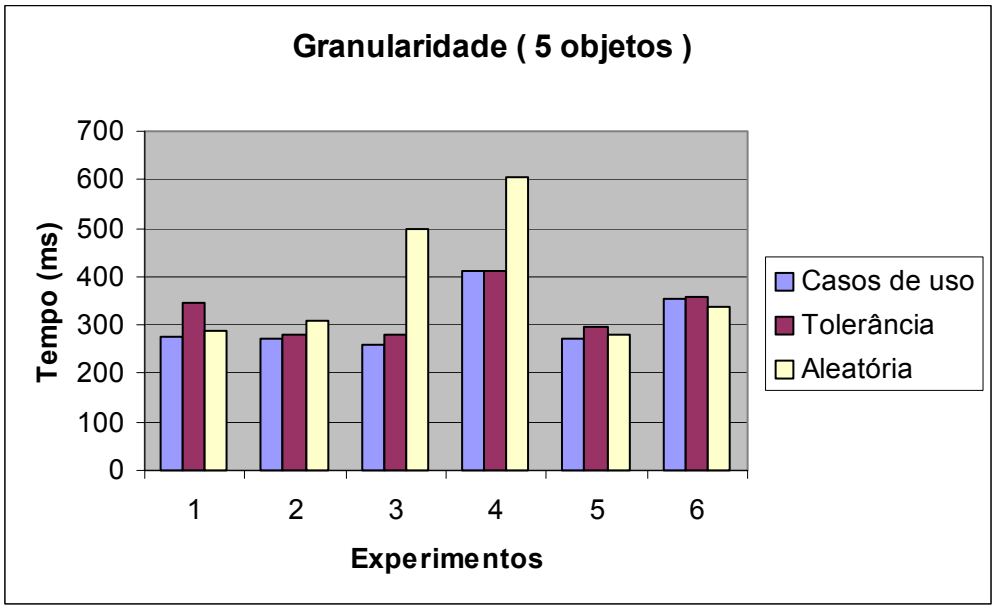

FIGURA 6.13– Granularidade em 5 objetos.

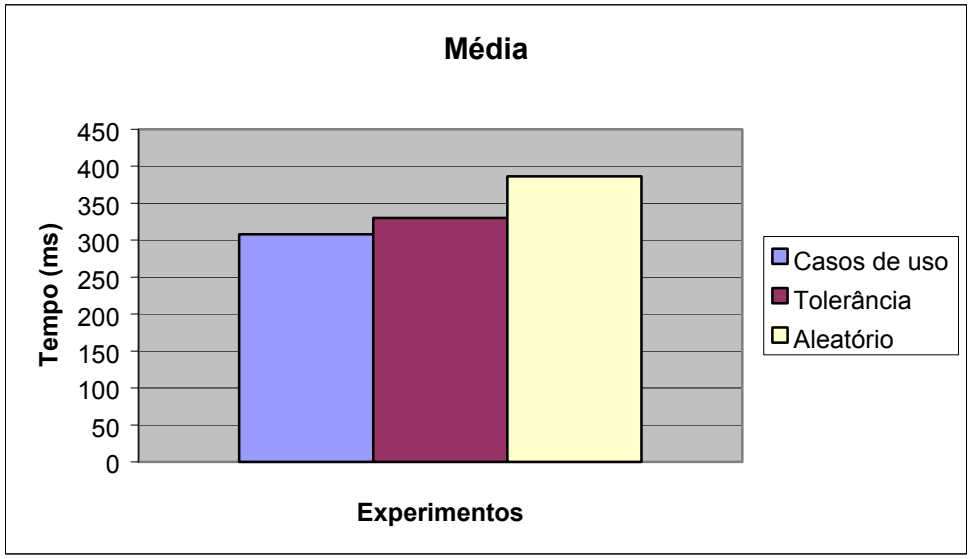

FIGURA 6.14 – Gráfico da média – Granularidade em 5 objetos.

A seguir, observa-se na Figura 6.15, o gráfico dos resultados granularizando-se os objetos em 10 objetos. O gráfico da média pode ser verificado logo a seguir, na Figura 6.16:

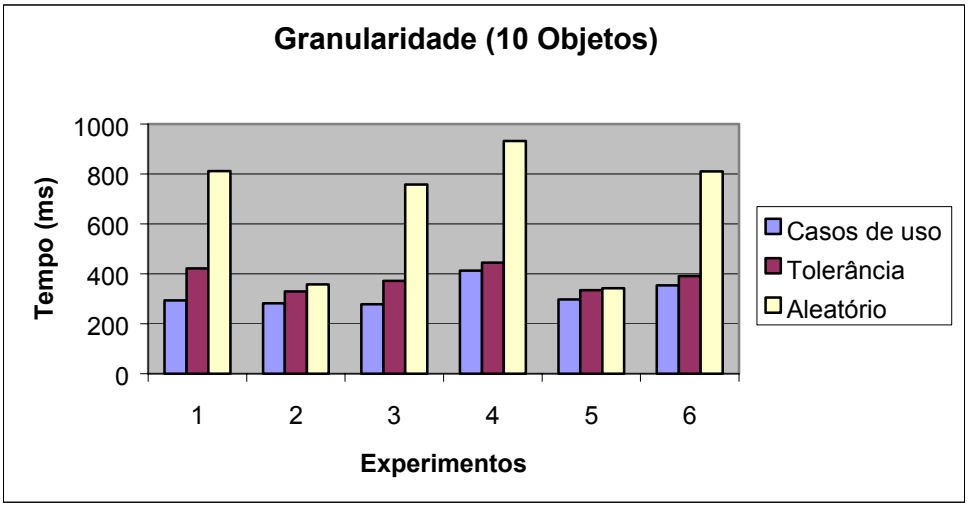

FIGURA 6.15 – Granularidade em 10 objetos.

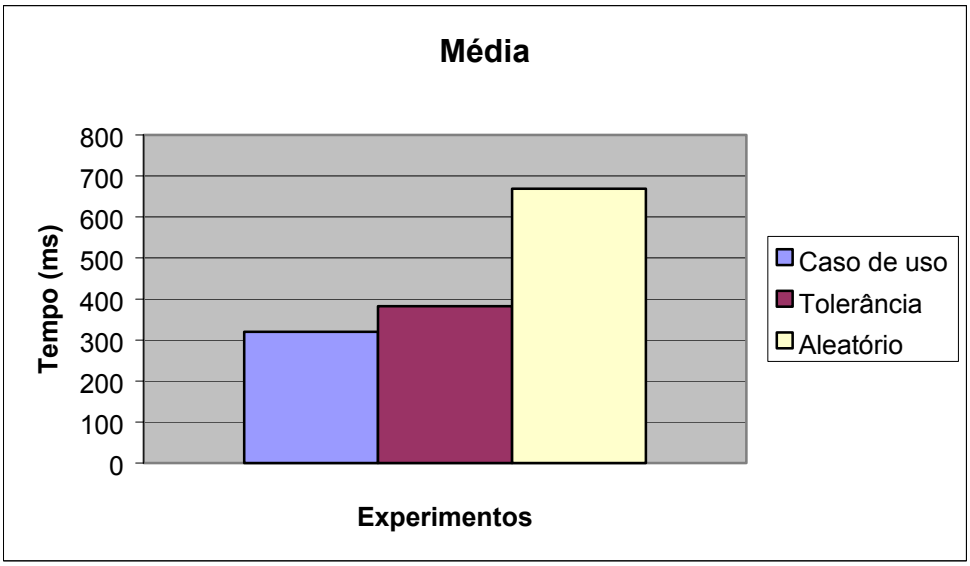

FIGURA 6.16 – Gráfico da média – Granularidade em 10 objetos.

No gráfico da Figura 6.17, observa-se a reunião das médias com objetos originais, ou seja, sem granularidade e granularidade em 5 objetos e em 10 objetos.

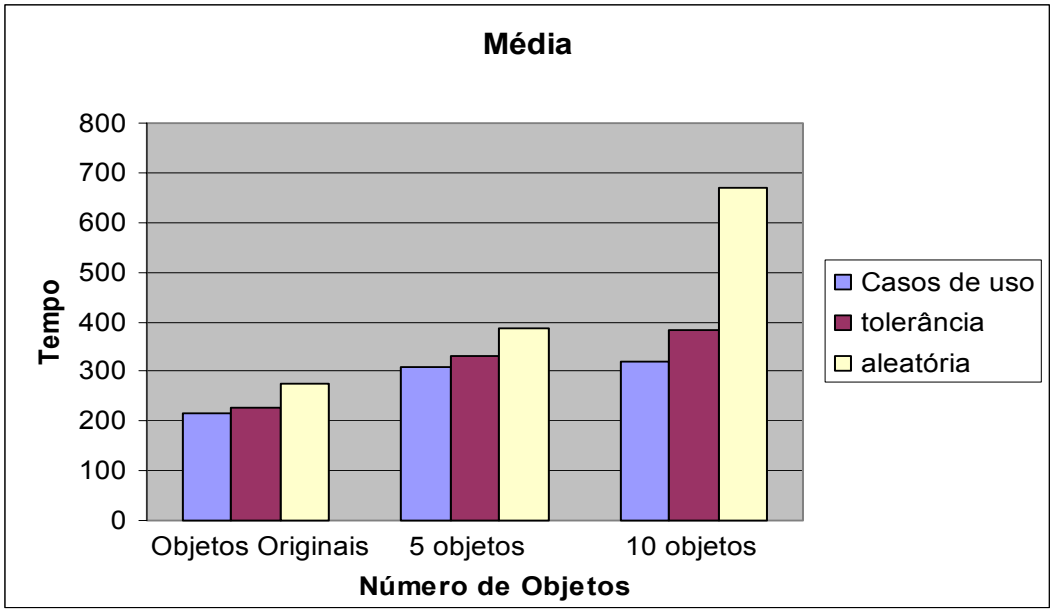

FIGURA 6.17 – Gráfico da média com objetos originais, 5 e 10 objetos.

Percebe-se na Figura 6.17 que, na média, a distribuição onde objetos se colaboram por casos de uso apresentou melhor desempenho em todos os experimentos. Objetos acoplados mais fortemente, ou seja, mais próximos uns dos outros possuem comunicação mais rápida, muitas vezes não necessitando utilizarem a rede.

Percebe-se também que a distribuição por tolerância a falhas inicialmente, com objetos originais apresentou-se uma diferença sutil em relação aos casos de uso, diferença esta que foi aumentando conforme aumentava-se o grau de granularidade. Os novos objetos sobrecarregam as máquinas, piorando o desempenho das mesmas.

A distribuição aleatória apresentou o pior desempenho em todos os casos. Percebe-se que o desempenho prejudicado é mais expressivo na granularidade em 10 objetos, isto pode ser explicado pelo fato de quebrando-se em mais objetos, estes ficam mais dispersos pela rede e surge uma latência maior na comunicação entre eles.

6) Para uma observação de como o tráfego influencia no desempenho, repetiu-se as medidas com aumento de tráfego na rede utilizada. O tráfego foi gerado da seguinte forma:

a) utilizou-se o comando de rede ‘ping’ para disparar pacotes (ICMP) *Internet Control Message Protocol* entre as máquinas no laboratório:

C:\ ping –l 65500 –t

Onde: - l envia pacotes de tamanhos estabelecidos, no caso 65500 bytes.

- t dispara contra o host especificado até ser interrompido.

b) transferiu-se um arquivo criado com tamanho de 380 Mbytes, através da rede, entre as máquinas utilizadas para distribuir os objetos.

c) observou-se o recebimento destes pacotes através da ferramenta administrativa de monitoramento de desempenho denominada Monitor do Sistema, encontrada na versão Windows 2000 Professional. O parâmetro selecionado para análise foi ‘Pacotes recebidos por segundo’. Um exemplo de configuração desta ferramenta, a fim de se observar pacotes recebidos, pode ser observado na Figura 6.18:

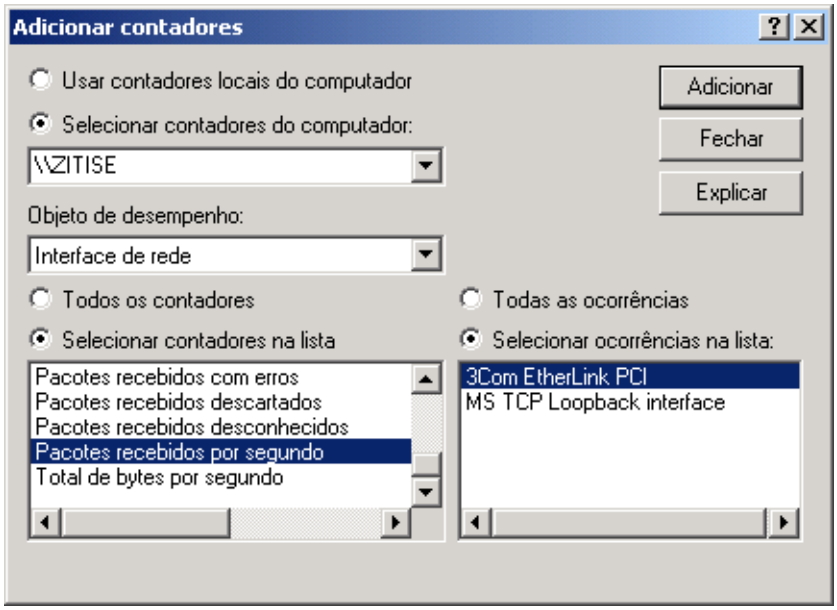

FIGURA 6.18 – Configuração da ferramenta de desempenho .

Na Figura 6.19, observa-se um exemplo do monitoramento destes pacotes na máquina Zitise, um aumento expressivo do número de pacotes recebidos pode ser percebido no gráfico desta figura:

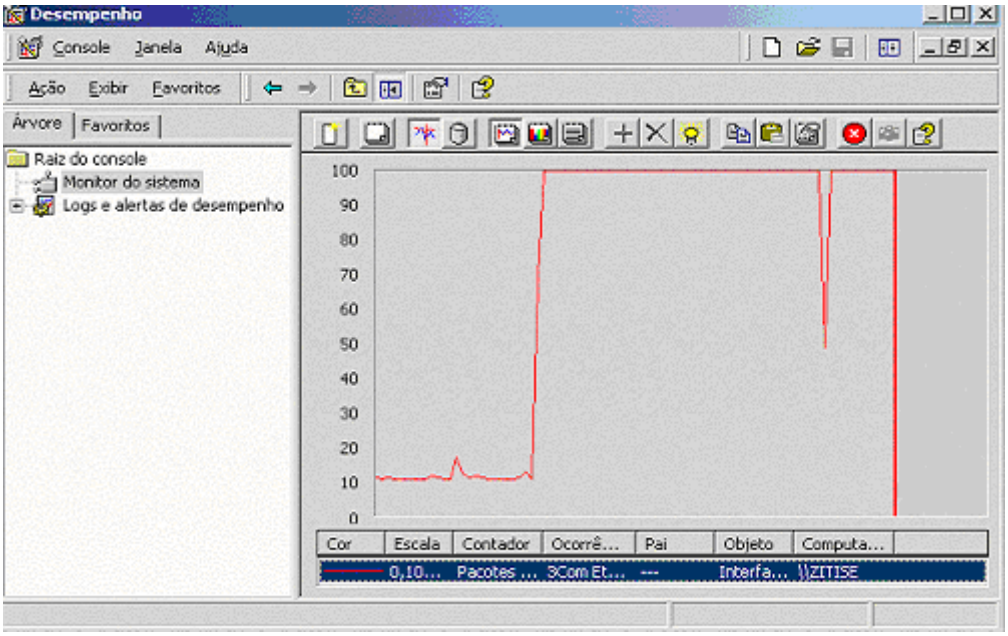

FIGURA 6.19 – Monitorando pacotes recebidos.

Na Figura 6.19, o eixo X representa a porcentagem dos pacotes recebidos, o eixo Y representa o tempo decorrido, o gráfico indica que os pacotes recebidos por segundo passam de 10% para 100% ao aumentarmos o tráfego.

Observa-se na Tabela 6.4 os valores coletados para cada experimento, sem granularidade e com aumento do tráfego.

TABELA 6.4 – Tabela de medidas de tempo com tráfego.

Simulador	Experimento	Cenários		
		Casos de Uso	Tolerância a Falhas	Aleatório
Zitise	1	221	199	276
	2	225	186	385
	3	221	186	274
	4	366	311	542
	5	222	197	333
	6	289	253	418
Anafi	1	219	198	318
	2	212	200	362
	3	213	189	285
	4	365	310	558
	5	227	200	338
	6	288	250	420
Lefkas	1	215	199	301
	2	233	209	622
	3	213	198	273
	4	360	334	622
	5	240	206	513
	6	301	270	462
Skopelos	1	211	200	253
	2	205	186	709
	3	212	185	273
	4	351	310	603
	5	240	196	495
	6	279	252	474

O gráfico da Figura 6.20 ilustra os resultado obtidos, com aumento de tráfego, somente para os objetos originais, ou seja, sem granularidade. A média pode ser verificada logo a seguir na Figura 6.21:

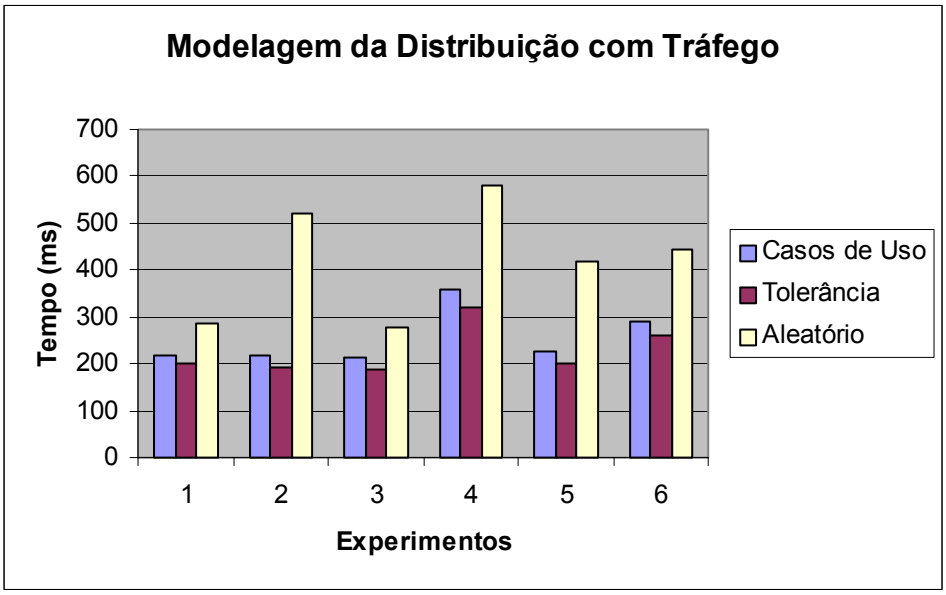

FIGURA 6.20 – Gráfico modelagem da distribuição.

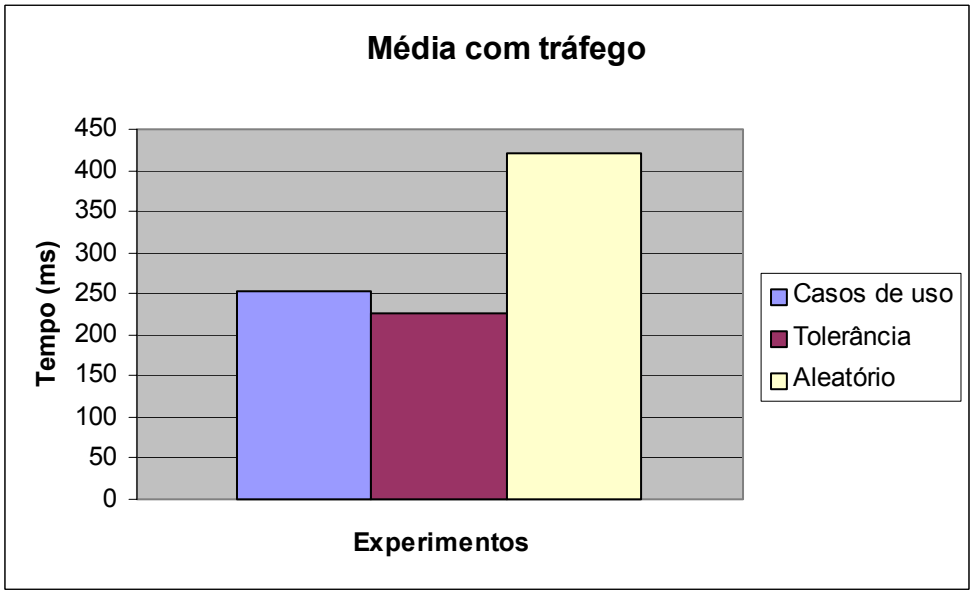

FIGURA 6.21 – Gráfico da média.

7) Repetiu-se as medidas aumentando-se a granularidade dos objetos em 5 e posteriormente em 10 objetos, com aumento de tráfego.

A Tabela 6.5 exibe os valores das medidas coletadas:

TABELA 6.5– Tabela de medidas de tempo com tráfego.

Simulador	Experimento	Cenários								
		Casos de Uso			Tolerância a Falhas			Aleatória		
			+5	+10		+5	+10		+5	+10
Zitise	1	221	316	449	199	327	325	276	313	708
	2	225	321	418	186	271	299	385	710	1066
	3	221	314	437	186	291	416	274	617	889
	4	366	506	597	311	415	416	542	845	1049
	5	222	336	468	197	301	326	333	748	1178
	6	289	425	549	253	365	367	418	445	774
Anafi	1	219	292	414	198	330	474	318	348	726
	2	212	301	398	200	280	364	362	747	1031
	3	213	320	440	189	290	379	285	619	786
	4	365	457	592	310	415	451	558	751	1174
	5	227	328	439	200	310	321	338	721	1074
	6	288	418	490	250	350	408	420	483	806
Lefkas	1	215	321	403	199	435	548	301	336	662
	2	233	305	385	209	301	362	622	1444	2115
	3	213	310	429	198	301	399	273	561	860
	4	360	474	575	334	420	491	622	888	1138
	5	240	319	437	206	298	363	513	1113	1089
	6	301	420	525	270	357	425	462	527	877
Skopelos	1	211	303	442	200	278	338	253	310	705
	2	205	319	408	186	273	294	709	1408	1115
	3	212	307	452	185	254	293	273	564	808
	4	351	437	538	310	405	420	603	953	1206
	5	240	319	437	196	295	327	495	1214	962
	6	279	368	468	252	350	365	474	558	776

A seguir, observa-se na Figura 6.22, o gráfico dos resultados, com aumento de tráfego, granularizando-se os objetos em 5 objetos. O gráfico da média pode ser verificado logo a seguir na Figura 6.23:

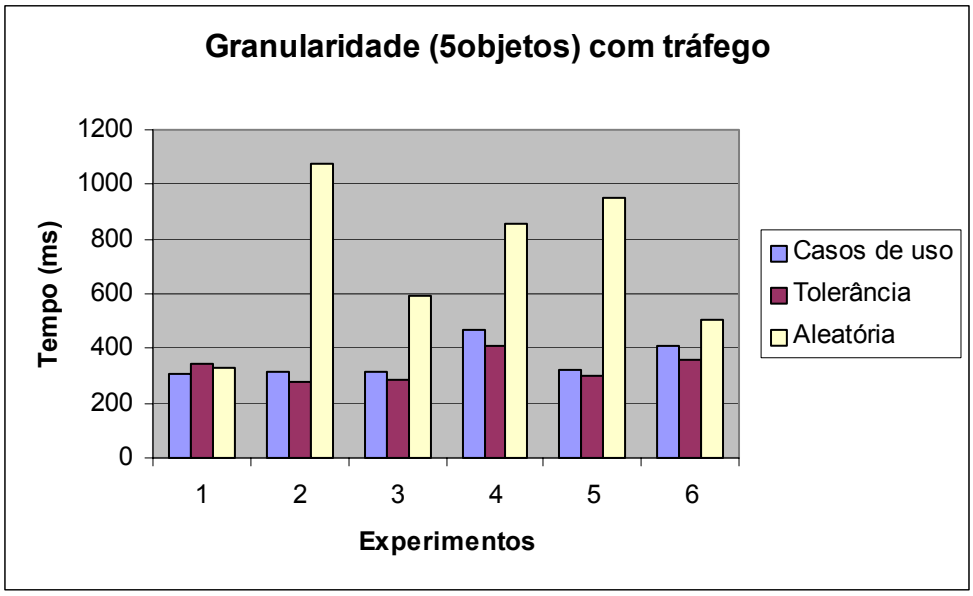

FIGURA 6.22 – Granularidade em 5 objetos com tráfego.

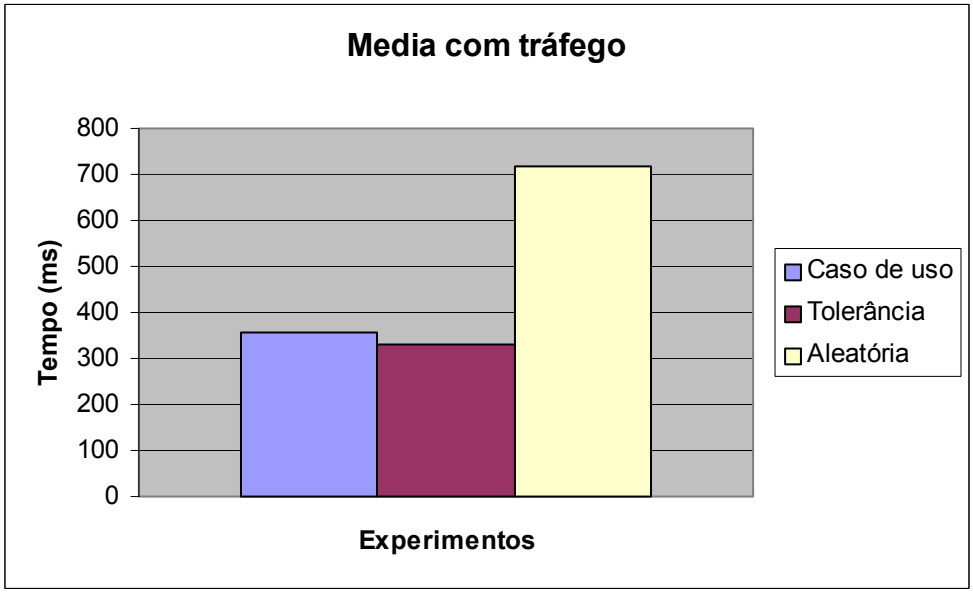

FIGURA 6.23 – Gráfico da média - Granularidade em 5 objetos com tráfego.

A seguir, observa-se na Figura 6.24, o gráfico dos resultados, com aumento de tráfego, granularizando-se os objetos em 10 objetos. O gráfico da média pode ser verificado logo a seguir na Figura 6.25:

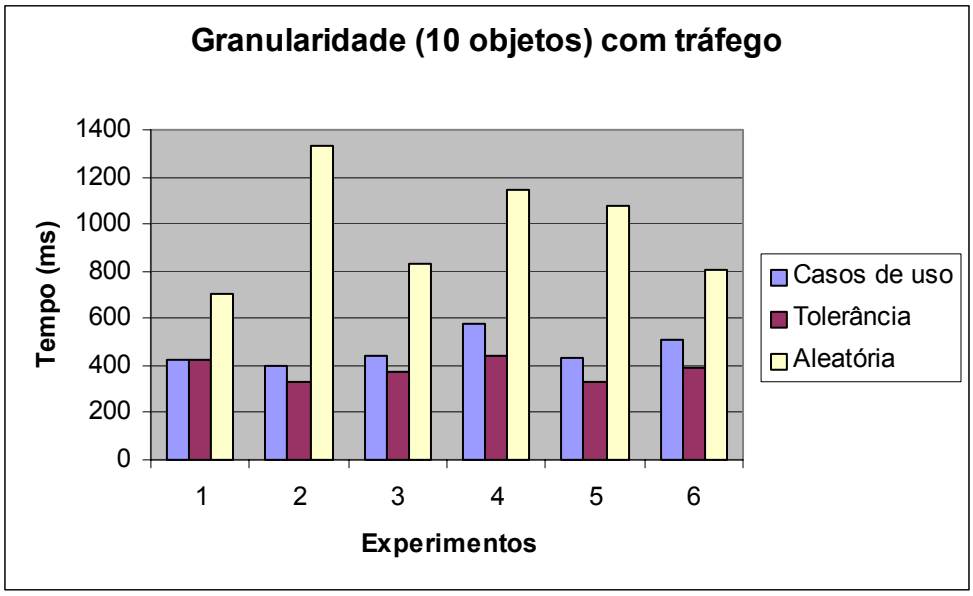

FIGURA 6.24 – Granularidade em 10 objetos com tráfego.

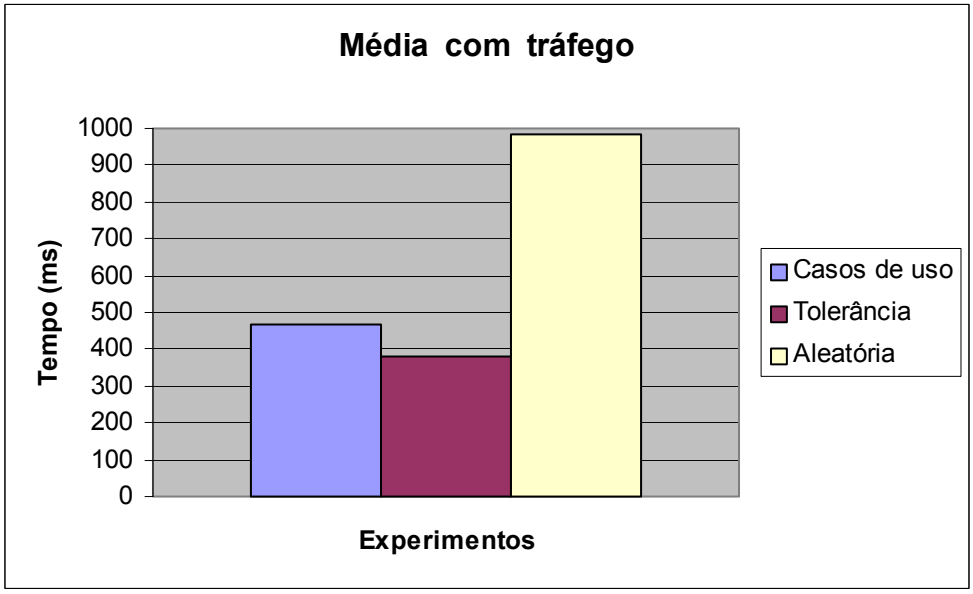

FIGURA 6.25 – Gráfico da média - Granularidade em 10 objetos com tráfego.

No gráfico da Figura 6.26, observa-se a reunião das médias, com aumento de tráfego, dos objetos originais (sem granularidade), granularidade em 5 objetos e em 10 objetos:

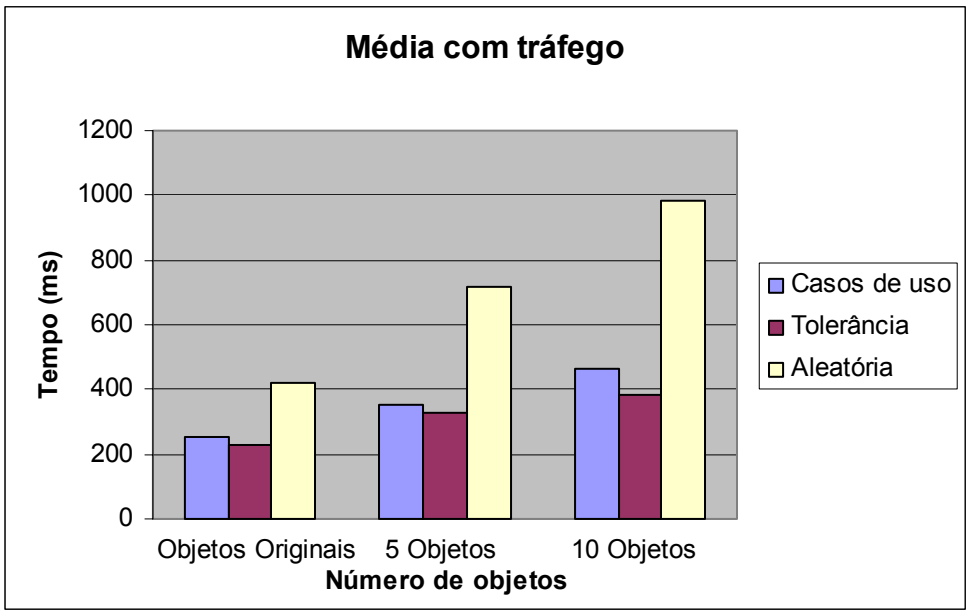

FIGURA 6.26 – Gráfico da Média com objetos originais, 5 e 10 objetos com tráfego.

No gráfico da Figura 6.26 observa-se que os objetos distribuídos por casos de uso mostram, em média, um desempenho pior que dos objetos distribuídos por tolerância a falhas. Isto ocorre devido ao fato de que, os objetos distribuídos por casos de usos podem estar instanciados localmente ou remotamente e muitas vezes, o caso de uso pode estar totalmente remoto, prejudicando assim o desempenho devido ao tráfego da rede. Na distribuição de objetos por tolerância a falhas, como todos os objetos estão instanciados em todas as máquinas, pode-se muitas vezes, utilizar objetos locais. Nestas condições, a distribuição por tolerância a falhas pode, muitas vezes, se apresentar melhor em desempenho que a distribuição por casos de uso.

O tráfego interferiu de forma mais expressiva na distribuição de forma aleatória, já que neste modelo, os objetos ficam mais dispersos pela rede. Com o aumento da granularidade o desempenho piorou ainda mais que nos casos anteriores, pois a rede se apresenta sobrecarregada e existem mais objetos se comunicando remotamente para realizar uma determinada funcionalidade.

CAPÍTULO 7

CONCLUSÕES

Pelas propriedades inovadoras que apresenta, a técnica de objetos distribuídos é considerada uma tecnologia promissora. Porém, ao se decidir utilizar esta tecnologia, deve-se observar algumas regras a fim de se evitar o insucesso do projeto. Como por exemplo, deve-se ter em mente a complexidade para se trabalhar com a distribuição: o aumento do número de interfaces e de linhas de códigos, observar o desempenho, já que a execução se realizará em ambientes remotos, etc. Conhecer estas regras, e tê-las como questões importantes sobre as propriedades da distribuição, deve ser uma das primeiras providências a serem tomadas ao se decidir trabalhar com objetos distribuídos.

Um outro fator determinante é a escolha entre os padrões disponíveis. Deve-se analisar as necessidades do projeto juntamente com os pontos oferecidos pelos padrões de distribuição. Características singulares são encontradas em cada tecnologia, por exemplo:

- flexibilidade de linguagens heterogêneas propiciada pelo padrão CORBA ou as várias funcionalidades disponíveis pelos seus serviços;
- facilidade na programação encontrada no RMI, etc.

No caso deste trabalho, optou-se pelo padrão RMI. Após estudos, constatou-se que as funcionalidades apresentadas por esta tecnologia se mostravam suficientes para a aplicação. A facilidade de programação já citada, também foi considerada como um fator decisivo.

Com relação a limitações do Software Simulador, pode-se perceber que as mesmas foram satisfatoriamente eliminadas com a aplicação dos objetos distribuídos ao sistema. Questões como disponibilidade, tolerância a falhas,

concorrência e flexibilidade foram resolvidas com o emprego da tecnologia da distribuição.

A deficiência encontrada em como distribuir os objetos de uma aplicação, motivou a criação de técnicas de modelagem de distribuição. Estas podem ser consideradas como um auxílio para se conseguir melhorias nas características da distribuição, já que resultados apresentados pelas experiências realizadas comprovam que tal modelagem influencia diretamente nas características do sistema, como desempenho e disponibilidade.

Como o propósito do trabalho não era apresentar qual a melhor técnica de modelagem de distribuição e sim comparar os resultados mostrando vantagens e desvantagens de cada técnica, lista-se aqui as conclusões obtidas através da observação dos resultados que o sistema apresentou ao utilizar cada técnica de modelagem.

Percebe-se, através dos resultados apresentados que, quando o projeto necessita de melhor desempenho, uma boa escolha pode ser a modelagem dos objetos que se colaboram para realizar os casos de Uso. Esta técnica apresentou melhores tempos na maioria os experimentos realizados. Percebe-se assim, que a utilização desta técnica, permite diminuir a latência de comunicação entre os objetos que estão interagindo com maior freqüência, já que cria um forte acoplamento entre eles, deixando-os mais próximos uns dos outros, evitando assim acessos remotos excessivos.

Porém, quando a preocupação é disponibilidade, a modelagem por tolerância a falhas se apresenta como a melhor escolha. Esta técnica garante a disponibilidade dos objetos, já que estes se apresentam instanciados em todas as máquinas. Percebe-se, através dos resultados obtidos, que com esta técnica, o desempenho das máquinas pode ser afetado devido à sobrecarga de objetos instanciados nas máquinas, mas mesmo prejudicado, o desempenho se mostra melhor que se utilizando a técnica por modelagem aleatória.

Já a modelagem aleatória, se apresentou como a pior opção em todos os casos. Como os objetos estão mais dispersos pela rede, são necessários mais acessos remotos que pelas outras técnicas de modelagem, aumentando-se, assim, o tráfego da rede, prejudicando inevitavelmente o desempenho das máquinas envolvidas.

Através dos gráficos obtidos, observa-se que os resultados seguem estas mesmas tendências também quando se aumenta a granularidade dos objetos, ou seja, na média, a técnica de objetos que se colaboram para realizar casos de uso continuam apresentando os melhores desempenhos. A modelagem aleatória apresenta resultados ainda piores quando se realiza a quebra dos objetos em objetos menores, pois a quantidade de objetos distribuídos pela rede aumenta. Assim, as conseqüências do aumento da granularidade refletem mais expressivamente quando os acessos são remotos.

A granularidade influencia no desempenho dos objetos, ou seja, a quebra dos objetos modifica o comportamento dos mesmos, assim, quanto maior a granularidade maior a latência da comunicação entre os objetos, pois estes estão ainda mais dispersos pela rede.

A necessidade da granularidade surgiu na realização dos testes práticos no Software Simulador. Desta forma, este tópico não foi exaustivamente pesquisado, não mencionado assim como estratégia de desenvolvimento. Este assunto merece maior atenção, já que ‘como quebrar os objetos’ é uma tarefa complexa que deve ser considerada na aplicação distribuída. Assim, a granularidade é um assunto interessante e que pode ser melhor explorado em trabalhos futuros.

Esta dissertação de mestrado apresentou técnicas de como distribuir os objetos, tentando minimizar a carência encontrada na literatura de técnicas que tratam do assunto. Concluiu-se que o fato de se organizar os objetos através das regras apresentadas por este trabalho, pode melhorar algumas características da distribuição, como desempenho e disponibilidade, trazendo

vantagens à execução do projeto. Percebe-se então que 'como distribuir' os objetos de uma aplicação é uma atividade que deve ser considerada ao se decidir trabalhar com objetos distribuídos.

Este trabalho foi apresentado no Congresso Internacional voltado para tecnologia espacial 'SpaceOps 2002' realizado em Houston, EUA, em outubro de 2002.

O mesmo foi também apresentado no 'Congresso Nacional da Tecnologia da Informação e Comunicação - SUCESU 2003' realizado em Salvador, BA em abril de 2003.

Um artigo originado deste trabalho foi submetido, para publicação, à revista especializada em tecnologia de objetos 'Object Magazine'.

REFERÊNCIAS BIBLIOGRÁFICAS

Ahuja, S.; Quintao, R. Performance evaluation of Java RMI: a distributed object architecture for internet based application. In: International Symposium On Modeling, Analysis And Simulation Of Computer And Telecommunication Systems, 8., 2000, San Francisco, California. **Eletronic Proceedings.** California: IEEE, 2000. Disponível em: <http://computer.org/proceedings/mascots/0728/07280565 abs.htm>. Acesso em: 12 maio 2001.

Bakker, A.; Kuz, I.; Steen, M. S. Towards a taxonomy of distributed-object models. In: AnnuaL ASCI Conference, 3., 1997, Heijen, The Netherlands. **Eletronic Proceedings.** Heijen: NEC Research Institute, 1997, p. 22-27. Disponível em:<http://citeseer.nj.nec.com/bakker97toward.html>. Acesso em: 24 ago. 2002.

Bernardino, F. P. **Tutorial sobre uso de RMI em aplicativos Java**. Rio de Janeiro: Instituto de Matemática da Universidade Federal do Rio de Janeiro, 2001. Disponível em: <http://tedmos.nce.ufrj.br/java/sood/fabio RMI.html>. Acesso em: 03 maio 2001.

Bernstein, P.A. Middleware: a model for distributed system services, **Communications of the ACM**, v.39, n. 2, 1996.

Bray, M. **Middleware**, 1998. Disponível em: <http://www.sei.cmu.edu/str/descriptions/middleware.html>. Acesso em: 24 out. 2002.

Booch, G.; Rumbaugh, J.; Jacobson, I. **The Unified Modeling Language User Guide.** Reading, Massachusetts: Rational Software,1999.

Butler, J. M. Quantum modeling of distributed object computing, In: Annual Simulation Symposium, 28, 1995, Santa Barbara, California. **Eletronic Proceedings.** Santa Brabara: IEEE,1995. p. 25 - 28. Disponível em:< http://computer.org/proceedings/ss/7091/70910175_abs.htm>. Acesso em: 28 set. 2001.

Chappell, D. **Understanding activeX and OLE.** Redmond, Washington: Microsoft Press,1996. ISBN 88-7131-779-3.

Chen, C. **Designing a fault tolerant model for distributed object using Java ORBs**, Essex, UK:, University of Essex, 1998. (Department of Computer Science)

Chin, R. S.; Chanson, S.T. Distributed object-based programming system, **ACM Computing Surveys**, v. 23, n. 1, Mar. 1991.

Costa, S. R. **Objetos Distribuídos:** conceitos e padrões. 2000. 230 p. (INPE-7939-TDI/744). Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2000.

Deitel, H.M.; Deitel, P.J. **Java como programar**. Porto Alegre: Editora Bookman, 2001.

Emmerich, W.; Kaveh, N. **Model checking distributed objects**. London: University College London, 2000. Dept. of Computer Science.

Ferreira, M. G. V. **Uma arquitetura flexível e dinâmica para objetos distribuídos aplicada ao Software de Controle de Satélites.** 2001. 244 p. (INPE-8602-TDI/787) Tese (Doutorado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2001.

Furlan, J. D. **Modelagem de objetos através da UML.** São Paulo: Makron Books, 1998.

Grimes, R. **Professional DCOM Programming.** Ontario,Canada: Wrox Press, 1997.

Instituto Nacional De Pesquisas Espaciais (INPE), **História**. São José dos Campos: INPE, 2003. Disponível em: <http://www.inpe.br/sobre_o_inpe/historia.htm>. Acesso em: 13 fev. 2003.

Kalogeraki, V.; Moser, L. E.; Melliar-Smith, P. M. Dynamic modeling of replicated objects for dependable soft real-time distributed object system, In: International Workshop on Object-Oriented Real-Time Dependable System, 4., Jan. 1999, Santa Barbara, California. **Eletronic Proceedings**. Santa Bárbara: IEEE,1999 27 - 29. Disponível em: <http://computer.org/proceedings/words/0101/ 01010048 abs.htm>. Acesso em: 15 set. 2000.

Katsaros, P.; Lazos, C. Structured performance modeling and analysis for object based distributed software system, In: International Conference On Parallel And Distributed Computing Systems, 15., 2002, Louisville, USA. **Eletronic Proceedings**. Louisville: NEC Research Institute, 2002

Mainetti, S. Jr. **Objetos distribuídos**. Curitiba: Visionnaire, jun. 1997.

Matthews, C. **Introduction to Java remote method invocation (RMI)**, 1998. Disponível em: <http://www.edm2.com/0601/rmi1.html>. Acesso em: 08 maio 2001.

Mowbray, T. J.; Ruh, W. A. **Inside CORBA - distributed object Standards and applications**. New York: Addison Wesley, 1997.

Object Management Group (OMG), **What is OMG-UML and Why is it important?**, Jun. 2001. Disponível em: <http://www.omg.org/gettingstarted/what_is_uml.htm>. Acesso em: 27 nov. 2001.

Orfali, R. Harquey, D.; Edwards, J. **The essential distributed objects survival guide**. New York: John Wiley & Sons,1996.

Pereira, P. M. **Serviço de Persistência para Ambientes Distribuídos explorando os recursos do repositório de interfaces.** 104 p. (INPE-9254-TDI/816). Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2002.

Purao, S.; Jain, H.; Nazareth, D. Effective distribution of object oriented applications. **Communications of ACM**, v.41, n. 8, Aug. 1998.

Rozenfeld, P.; Miguez, R.; Orlando, V. **Proposta de um simulador para o satélite SCD1.** São José dos Campos: INPE, 1990.

Saleh, K.; Probert, R.; Khanafer, H. The distributed objects computing paradigm: concepts and applications. **The Journal of system and Software**, v. 47, p. 125- 131, 1999.

Sessions, R. Ten rules for distributed object systems. **OS/2 Magazine**, Oct.1996. Disponível em: <http://www.objectwatch.com/articles_by_staff.htm>. Acesso em: 10 jun. 1999.

Sommerville, I. **Software engineering.** Harlow: Addison Wesley, 2001.

SUN MICROSYSTEMS. **An overview of RMI applications**. Sun Microsystems, 2003. Disponível m:<http://java.sun.com/docs/books/tutorial/rmi/overview.html> Acesso em: 02 fev. 2003.

Tamai,T. Objects and roles: modeling based on the dualistic view, **Information and Software Technology**, v. 41, p. 1005 – 1010, 1999.

TANENBAUM, A. S.; VAN STEEN, M., **Distributed system principles and paradigms** New Jersey: Prentice Hall, 2002.

Yamaguti, W.; Orlandi, E.F.E; Orlando, V., **Documento de Projeto Preliminar do Sistema Simulador do Satélite SCD1 – SIMS**. São José dos Campos: INPE, 1994.

Zhou, W.; Wang, L. **Distributed Object Replication in a Cluster of Workstation.** Australia: School of Computing and Mathematics – Deakin University, 1999.

BIBLIOGRAFIA COMPLEMENTAR

Ashok, S.; Bansal, V. Java: network-centric enterprise computing, **Computer Communications**, v.20, p.1467-1480, 1998.

Bitar, L.;Krupskaia, K. **Objetos Distribuídos:** um estudo sobre CORBA e DCOM. Belém, PA:, jan. 2000. Disponível em: <http://planeta.terra.com.br/informatica/Krups> . Acesso em:10 out. 2000.

Felber, P.; Jai, B.; Rastogi, R.; Smith, M. Using semantic knowledge of distributed objects to increase reliability and availability. In: IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS01), 6., Jan. 2001, Rome, Italy. **Eletronic Proceedings.** Rome: IEEE,2001. p. 153-160. Disponível em: <http://citeseer.nj.nec.com/felber01using.html>. Acesso em: 17 mar. 2002.

Homburg, P.; Doorn, L.; Steen, M.; Tanenbaum, A.S.; Jonge, W. **An object model for flexible distributed systems**. Amsterdam: Vrije Universiteit. Mar. 1995.

Jansen, M.; Klaver, E.; Verkaik, P.; Van Steen, M.; Tanenbaum, A.S. Encapsulating distribution by remote objects, **Information and Software Technology**, v.43, p. 353-363, 2001.

Juric, M.B.; Rozman, I.; Stevens, A. P.; Hericko, M.; Nah, S. Java2 Distributed Object Models Performance Analysis, comparison and Optimization**,** In: International Conference On Parallel And Distributed Systems - ICPADS'00, 7., Jul. 04 - 07, 2000,Iwate, Japan. **Eletronic Proceedings.** Iwate:IEEE, 2000. Disponível em: <http://computer.org/proceedings/icpads/0568/05680239abs.htm>. Acesso em: 23 mar. 2002.

Kono, K.; Masuda, T. Efficient RMI: dynamic of object serialization. In: International Conference On Distributed Computing System (ICDCS 2000), 20., Apr., 10 - 13, 2000,Taipei, Taiwan. **Eletronic Proceedings.** Taipei: IEEE, 2000. Disponível em: < http://computer.org/proceedings/icdcs/0601/06010308abs.htm Acesso em: 27 mar. 2001.

Mcpherson, S. **Java servlets and serialization with RMI.** Oct. 1999. Disponível em: <http://developer.java.sun.com/developer/technicalArticles/RMI/rmi>. Acesso em: 10 jan. 2002.

Morisawa, Y.; Okada, H.; Iwata, H.; Toyama, H. A computing model for distributed processing system and Its application. In: Asia Pacific Software Engineering Conference, Dec. 02 – 04, 1998,Taipei, Taiwan. **Proceedings.** Taipei: IEEE, 1998. p. 314 - 320.

Mos, A.; Murphy, J. Performance monitoring of Java component-oriented distributed applications. In: IEEE International Conference On Software, Telecommunications And Computer Networks (SoftCOM), 9., 2001. Dubrovnik, Croatia. **Eletronic Proceedings.** Dubrovnik: NEC Research Institute, 2001.Disponível em: http://citeseer.nj.nec.com/mos01 performance.html>. Acesso em: 10 jan. 2003.

Narasimhan, N.; Moser, L. E.; Melliar-Smith, P.M. Transparent Consistent Replication of Java RMI Objects. In: Symposium, Distributed Objects & Applications (DOA 2000), 2., 2000, University of California at Santa Barbara: Department of Electrical and Computer Engineering .**Proceedings.** Antwerp, Belgium: IEEE, 2000. p. 17-26.

Raj, G.S. **A Detailed Comparison of CORBA, DCOM and Java/RMI.** 1998. Disponível em: <http://www.execpc.com/~gopalan/misc/compare.html>. Acesso em: 04 maio 2001.

Reilly, D. **Introduction to Java RMI**. 1997. Disponível em : < http:// www. javacoffebreak.com/articles/ javarmi.html>. Acesso em: 07 maio 2001.

Ormerod, A.R.B.; Rocha, F.K.M. **RMI – Java** . Rio de Janeiro: Universidade do Estado do Rio de Janeiro, 1999. Disponível em: <http:// www.ime.uerj.br/~alexzt/cursos/topesp_inter/trabs>. Acesso em: 03 maio 2001.

Pooley, R.; King, P. **The unified modeling language and performance engineering.** Riccarton, Scotland: Department of Computing And Electrical Engineering, Heriot – Watt University, 1999.

Valente, M.T.O. **Aula prática sobre Java RMI** - sistemas distribuídos. PUC-MG: Instituto de Informática, Pontifícia Universidade Católica de Minas Gerais, 1999.

Wollrath, A.; Riggs, R.; Waldo, J. A Distributed object model for the java system. In: Usenix Conference on Object-Oriented Technologies. Toronto, Ontário. **Proceedings.** Toronto: Sun Microsystems, Jun. 1996.