MESTRADO EM ENGENHARIA ELÉTRICA
UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE COMUNICAÇÕES
DECOM – FEEC – UNICAMP

Fernando Luiz Prochnow Ramme

Uma Arquitetura Cliente/Servidor para Apoiar a Simulação de Redes em Ambiente de Simulação Orientado a Eventos Discretos

Campinas / SP

2004

UNICAMP

MESTRADO EM ENGENHARIA ELÉTRICA
UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE COMUNICAÇÕES
DECOM – FEEC – UNICAMP

Fernando Luiz Prochnow Ramme

Uma Arquitetura Cliente/Servidor para Apoiar a Simulação de Redes em Ambiente de Simulação Orientado a Eventos Discretos

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, Departamento de Comunicações, como parte dos requisitos exigidos para a obtenção do título de *Mestre em Engenharia Elétrica*.

Prof. Dr. Leonardo de Souza Mendes
Orientador

Prof. Dr. Antônio Marcos Alberti - INATEL
Prof. Dr. Léo Pini Magalhães - FEEC/UNICAMP
Prof. Dr. Maurício Ferreira Magalhães - FEEC/UNICAMP

Campinas / SP

2004

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

R285a

Ramme, Fernando Luiz Prochnow
Uma arquitetura cliente/servidor para apoiar a simulação de redes em ambiente de simulação orientado a eventos discretos / Fernando Luiz Prochnow Ramme. --Campinas, SP: [s.n.], 2004.

Orientador: Leonardo de Souza Mendes.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Simulação. 2. Redes de computação. I. Mendes, Leonardo de Souza. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

O objetivo deste trabalho é desenvolver uma arquitetura *cliente/servidor* em três camadas para o ambiente *Hydragyrum*, uma ferramenta de simulação orientada a eventos discretos, a fim de prover um procedimento padronizado para realizar a interoperabilidade entre diversos projetos de simulação criados sob este ambiente. Como conseqüência direta desta arquitetura, simulações distribuídas poderão ser realizadas onde modelos podem obter ganhos de processamento desde que haja um balanceamento cauteloso entre a quantidade de computação dos processos e a necessidade de comunicação entre processos alocados em máquinas distintas.

Abstract

The objective of this work is to develop a three-tier client/server architecture for the Hydragyrum environment, an event driven simulation tool, in order to provide a standardized procedure to accomplish the interoperability among several simulation projects created under this environment. As a direct consequence of this architecture, distributed simulations could be done where models can obtain processing gains if there is a cautious balance between the computation size of the processes and the communication needs among processes allocated in different machines.

Agradecimentos

Agradeço ao meu orientador, Leonardo de Souza Mendes pelo apoio dado no transcorrer do trabalho.

Agradeço aos meus pais todo apoio durante o período de dedicação deste trabalho.

Agradeço aos membros da comissão julgadora a aceitação do convite para participar da avaliação deste trabalho.

Agradeço em especial aos colegas *Ana Carolina Gondim* e *Antônio Marcos Alberti* pelo apoio durante o período de dedicação deste trabalho.

Gostaria também de agradecer a todos os colegas que participaram com discussões e contribuições durante o período de convivência na pós-graduação.

"Estudar é, realmente, um trabalho difícil. Exige de quem o faz uma postura crítica, sistemática. Exige uma disciplina intelectual que não se ganha a não ser praticando-a. Estudar não é um ato de consumir idéias, mas de criá-las e recriá-las."

PAULO FREIRE

Lista de Abreviaturas e Siglas

ANSI	American National Standards Institute
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BD	Banco de Dados
BOA	Basic Object Adapter
COM	Common Object Model
CORBA	Common Object Request Broker Architecture
DBMS	DataBase Management System
DC	Data Connection
DLL	Dynamic Linked Library
GIOP	General Inter-ORB Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IIOP	Internet Inter ORB Protocol
JDBC	Java Database Connectivity
JDK	Java Development Kit
LAN	Local Area Network
MER	Modelo Entidade-Relacionamento
NC	Network Connection
ODP	Open Distributed Processing

OMG	Object Management Group
ORB	Object Request Broker
OSI	Open Systems Interconnection
PC	Personal Computer
POA	Portable Object Adapter
QoS	Quality of Service
RDBMS	Relational Database Management Systems
RPC	Remote Procedure Call
RUP	Rational Unified Process
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol / Internet Protocol
UML	Unified Modeling Language
URL	Uniform Resource Locator
WWW	World Wide Web

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 O Trabalho

O ambiente de simulação *Hydragyrum*, e seus "antepassados" [1, 4], foi desenvolvido com o intuito de se ter a disposição um aplicativo que permitisse a simulação de sistemas de comunicações e ao mesmo tempo fornecesse recursos para o desenvolvimento de novos modelos de rede sem que fossem necessárias alterações no núcleo do simulador. No entanto, o ambiente *Hydragyrum* não contempla um procedimento que permita a interoperabilidade entre diferentes projetos de simulação desenvolvidos por equipes distintas de trabalho, que encontram-se distribuídas pela rede.

O objetivo deste trabalho é a implementação de uma extensão ao ambiente *Hydragyrum* visando disponibilizar uma arquitetura *cliente/servidor* que permita a interoperabilidade entre diversos projetos de simulação criados sob o *Hydragyrum*. Através desta arquitetura, projetos *Hydragyrum*, desenvolvidos nesta filosofia de modelagem, executando do lado cliente comunicam-se por intermédio da entidade servidor. Esta implementação poderá também ser útil na integração das atividades de outros ambientes hoje utilizados em redes de computadores e que poderão, assim, partilhar de uma ferramenta comum para processamento distribuído.

1.2 Técnica de Simulação Orientada a Eventos

Entende-se por simulação o uso de modelos para o estudo de problemas de natureza complexa através do processamento computacional. Assim, a simulação consiste no processo de construção de um modelo que replica o funcionamento de um sistema real ou ideal e na condução de experimentos computacionais com o objetivo de melhor entender o problema em estudo.

O método de processamento da informação utilizado neste trabalho emprega a técnica de simulação orientada a eventos *(discrete event* ou *event-driven*). Na simulação *event-driven*, os modelos agendam eventos para serem executados em um determinado instante de tempo simulado. Os modelos são ativados apenas quando recebem eventos para processar. Os eventos a serem executados são ordenados em termos de prioridades, sendo que os eventos que têm maior prioridade devem ser executados primeiro, porque possuem o menor tempo agendado para sua execução. Não existe um controle central sobre quando um modelo pode agendar eventos para serem executados, por ele próprio ou por outro modelo, durante a simulação. A simulação *event-driven* é uma maneira eficiente de simularmos sistemas nos quais não existe um relógio sincronizando os modelos ou quando determinados estados dos modelos podem ser alterados a qualquer instante [1].

1.2.1 Simulação Seqüencial

A simulação seqüencial é uma técnica onde um incremento de tempo, um bloco de dados ou um evento é executado de cada vez por um processador na simulação. A simulação seqüêncial de um sistema consiste na modelagem das entidades físicas (objetos ou componentes de um sistema que necessitam ser representados explicitamente no modelo de simulação) do sistema e das interações entre elas, através de eventos que modificam os estados do sistema. Três estruturas de dados básicas são

utilizadas: variáveis de estado que descrevem o estado do sistema; uma lista de eventos futuros que contém todos os eventos pendentes que foram escalonados para a execução; e o relógio global que controla o progresso da simulação.

Cada evento nesta técnica de simulação possui um *timestamp* e determina alguma mudança do estado do sistema que está sendo simulado. O *timestamp* determina quando essa mudança deve ocorrer. Os passos do programa de simulação, fundamentalmente, são os seguintes [16]:

1. Selecionar o evento de menor *timestamp* da lista de eventos futuros para ser processado;
2. Atualizar o valor do relógio global de acordo com o *timestamp* do evento a ser processado;
3. Processar o evento e fazer as modificações necessárias nas variáveis de estado do sistema;
4. Se o processamento do evento gerar novos eventos, esses devem ser postos na lista de eventos futuros;
5. Se a lista de eventos futuros não estiver vazia, voltar ao passo 1. Se não, terminar simulação.

A lista de eventos futuros é ordenada de modo não-decrescente pelo tempo de ocorrência dos eventos (*timestamp*). Essa ordenação limita a execução da simulação seqüencial restringindo a um único evento processado por ciclo da simulação. Porém, ela deve ser preservada, pois se um evento for executado fora da ordem, as variáveis de estado da simulação podem ser alteradas, afetando a execução dos eventos de tempo menor que ainda estão na lista. Essa restrição imposta pelo comportamento do sistema real é chamada de restrição de causa e efeito.

A simulação seqüencial pode ser empregada eficientemente em muitos casos, porém não fornece um bom desempenho na simulação de grandes sistemas com numerosas entidades e eventos. O tempo de processamento pode ser muito grande, tornando a simulação inviável. Além da restrição da lista de eventos única, a simulação seqüencial também fica limitada pela capacidade de memória de um único processador na qual a simulação é executada. Solucionando os problemas citados e tornando a aplicação da simulação mais eficiente, a simulação paralela e a simulação distribuída tem se tornado alternativas para a aplicação da simulação.

Existem várias formas propostas para a decomposição de uma simulação seqüencial para ser executada em múltiplos processadores [19]:

1. *Parallelizing compilers*: nesta abordagem, são utilizados compiladores de paralelização que exploram o paralelismo possível em um dado programa seqüencial. Como o compilador ignora a estrutura do problema, o paralelismo explorado é muito limitado.

2. *Replicated Trials (Parallel Independent Runs, Distributed Experiments)*: *"n"* simulações seqüenciais são executadas independentemente em *"n"* processadores, e seus resultados são calculados pela média de todas saídas das *"n"* execuções no final. Nenhuma coordenação é necessária entre as replicações, por isso alta eficiência é alcançada. Porém, os parâmetros de todas as replicações da simulação devem ser decididos antes da inicialização de qualquer uma delas. O espaço de memória é proporcional ao número das replicações.

3. *Distributed Functions*: as sub-tarefas essenciais da simulação (geração de números aleatórios, manipulação de arquivos, coleta de estatísticas) são associadas a um número de processadores. Isso requer algumas mudanças no código da simulação seqüencial e como o número de tais sub-tarefas é limitado, pouco paralelismo pode ser alcançado.

4. Distributed Events (com lista de eventos centralizada): um processador que esteja disponível processa o evento de menor *timestamp* da lista de eventos global. Para evitar computação errada, cada processador tem que garantir que o evento com o menor *timestamp* da lista não será cancelado por outros eventos que estejam sendo processados por outros processadores naquele instante. Isso requer um conhecimento do modelo de simulação, o que não é uma tarefa fácil. Além disso, a lista de eventos global pode se tornar o "*bottleneck*" se muitos processadores estiverem envolvidos na simulação.

5. Domain Decomposition: essa classe de decomposição está baseada na visão de que a execução da simulação é equivalente a preencher uma região bidimensional, sendo uma dimensão o tempo simulado e a outra as variáveis de estados, como na *Figura 1-1 (a)*. De acordo com essa visão, o domínio de espaço-tempo pode ser decomposto em relação ao tempo, *Figura 1-1 (b)*, ou ao espaço, *Figura 1-1 (c)*.

6. Space-Parallel Decomposition (ou *Distributed Simulation*): o modelo da simulação é decomposto em um número de submodelos ou componentes. Cada componente é associado a um processo, onde vários processos podem ser executados no mesmo processador.

7. Time-Parallel Decomposition (ou Time Parallelism): o domínio é dividido em um número de intervalos [ti-1, ti] para $0 \leq i \geq p$, sendo p o número de processadores. A cada processador está associado um intervalo e é responsável em computar os valores das variáveis de estado dentro daquele intervalo. Com esta abordagem, o mecanismo da simulação deve garantir que o estado do sistema no final do intervalo [ti-1, ti] combine com o estado inicial do intervalo [ti, ti+1]. A reexecução de alguns intervalos é necessária caso algum erro entre as variáveis de estado

seja detectado. A eficiência dessa decomposição depende de alguma forma de se prever com exatidão o estado inicial para cada intervalo.

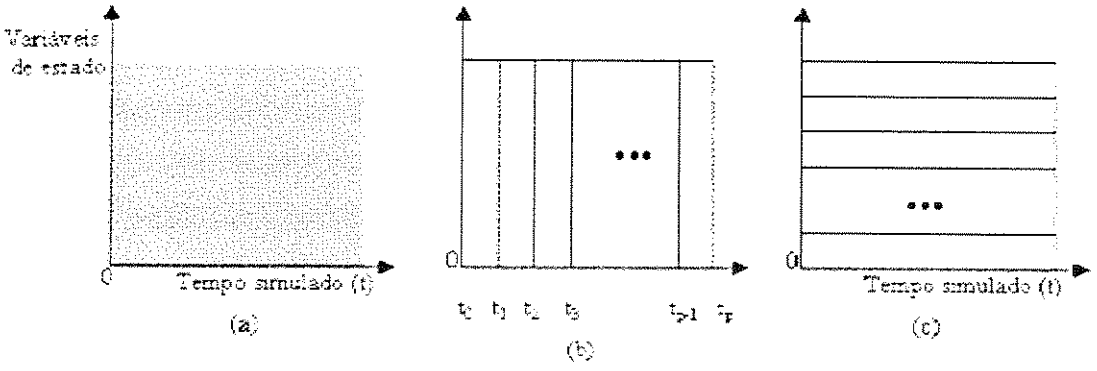

Figura 1-1: (a) Gráfico tempo-espaço. (b) Decomposição pelo tempo. (c) Decomposição pelo espaço.

1.2.2 Simulação Paralela

A simulação paralela é uma técnica que emprega memória compartilhada e vários processadores simultaneamente para realizar uma simulação. Na arquitetura de processadores com memória compartilhada tem-se múltiplos processadores operando independentemente, mas compartilhando o mesmo espaço de endereçamento, ou seja, um único bloco de memória, como pode ser observado na *Figura 1-2*. Somente um processador pode acessar a memória por vez. Por isso deve haver, por parte do programador, um controle de acesso para manter os dados consistentes. Não deve ser permitido que enquanto um processador esteja lendo um lado, um outro processador escreva na mesma localização. Isto é feito utilizando mecanismos como semáforos e monitores.

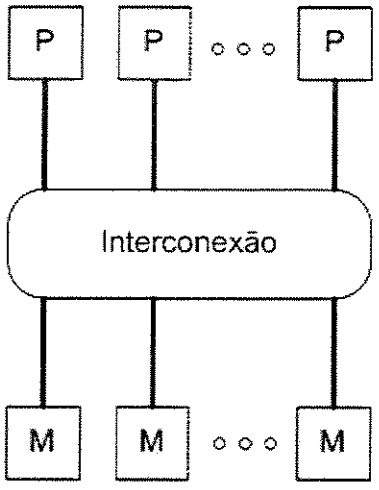

Figura 1-2: Processadores com memória compartilhada

A programação é feita em linguagens de programação paralela, onde construções e instruções paralelas permitem declarações de variáveis compartilhadas e seções paralelas de código. Seqüências de código escritas em alto nível para processadores individuais que podem acessar localidades compartilhadas ou *Threads* são utilizadas para apoiar a programação. A sincronização entre tarefas é feita por escrita/leitura na memória compartilhada e o usuário é responsável por sua especificação. A escalabilidade é limitada pelo número de caminhos entre memória e processadores.

1.2.3 Simulação Distribuída

A simulação distribuída é uma técnica que emprega vários computadores, onde cada um possui toda a área de memória reservada somente para o processamento do computador em questão. Após o resultado do processamento em cada computador, geralmente ocorre uma centralização destes resultados em algum computador responsável pela coordenação distribuída das tarefas.

A simulação distribuída constitui uma solução para os problemas encontrados na simulação seqüencial. Explorando-se o paralelismo inerente no modelo de simulação de um sistema, é possível dividir a simulação seqüencial em vários processos que podem ser executados paralelamente. Esses processos, chamados de processos lógicos, podem então ser distribuídos entre vários processadores de uma máquina paralela ou de um sistema distribuído, ganhando-se em *speedup* e desempenho.

Os processos lógicos representam processos físicos do sistema real (entidades da simulação). Um grupo de LPs (*Logical Process*) pode executar eventos paralelos de modo assíncrono ou síncrono. A Interface de Comunicação fornece ao *LP* mecanismos para a troca de dados com os outros *LPs*. Cada *LP* atua em uma região, uma parte do modelo da simulação, processando os seus eventos locais e gerando eventos remotos [17]. A *Figura 1-3* apresenta os principais componentes de um sistema de simulação: as entidades da simulação, suas variáveis de estado, suas listas de eventos futuros e os canais entre as entidades para a troca de mensagens .

A execução dos eventos da simulação deve ser análoga ao comportamento do sistema real e os relacionamentos de causa e efeito dos eventos nunca devem ser violados [18]. Se num sistema real, um evento A ocorre antes de um evento B, o mesmo deve acontecer na simulação. Caso o evento B seja processado na simulação antes do A, tem-se uma violação da relação de causa e efeito. O mecanismo de implementação da lista de eventos futuros deve sempre selecionar o evento de menor *timestamp* para ser executado, pois, assim, a relação de causa e efeito dos eventos é preservada. A lista de eventos futuros pode ser implementada de duas formas:

1. Lista de eventos centralizada: todos os eventos da simulação são mantidos em uma única lista de eventos central em um determinado processo. Um escalonador de eventos decide qual processo deve receber um evento para ser executado.

Oferece bom desempenho apenas para um grupo limitado de aplicações, onde a natureza da lista de eventos é seqüencial.

2. Lista de eventos distribuída: cada processo mantém um relógio local virtual (*LVT – Local Virtual Time*) e uma lista de eventos. Cada processo então executa os eventos de menor *timestamp* de suas listas de eventos. Um erro de causa e efeito é detectado quando o evento retirado da lista de eventos futuros possuir um *timestamp* menor do que o *LVT* do processo lógico.

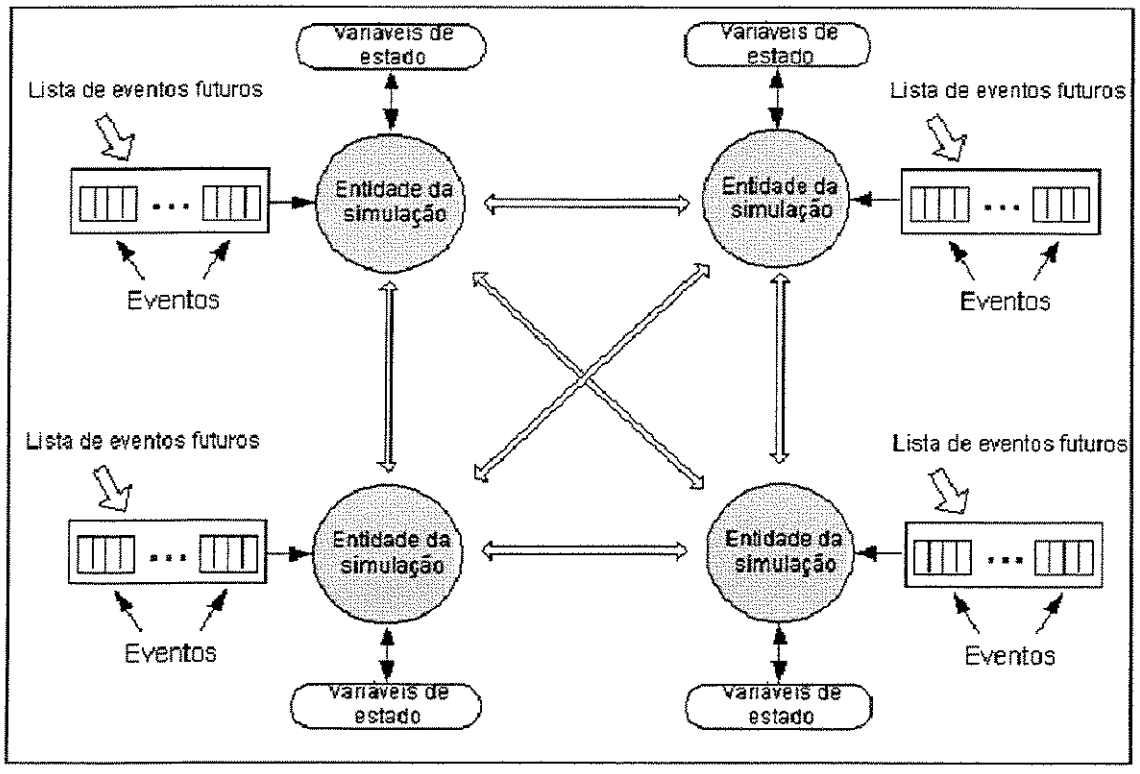

Figura 1-3: Estrutura de um Sistema de Simulação

A implementação distribuída da lista de eventos é a metodologia mais popular entre as duas citadas. Se cada *LP* pudesse prosseguir sua execução de forma independente das demais, melhores *speedups* poderiam ser alcançados. Porém, os *LPs* interagem entre si através de troca de mensagens, e não são independentes.

1.3 Introdução a Tecnologias Distribuídas

Para elaborar uma aplicação distribuída, existem diversas tecnologias que possibilitam a comunicação de mensagens entre aplicativos. Os aplicativos podem estar localizados fisicamente no mesmo computador ou em computadores distintos da rede. As tecnologias distribuídas estudadas neste trabalho foram *RPC*, *RMI*, *DCOM* e *CORBA*. Uma breve descrição sobre cada uma destas tecnologias é apresentada a seguir.

1.3.1 RPC

A Chamada de Procedimento Remoto (*Remote Procedure Call - RPC)* habilita o software a fazer chamadas a funções através de redes de computadores. Fundamentalmente, o mecanismo *RPC* cria uma chamada remota de procedimento para que uma aplicação comunique-se uma com a outra. Conceitualmente, uma chamada *RPC* é idêntica em sintaxe a uma chamada a uma função local. Existem algumas variações de implementação *RPC* : 1) *Sun RPC* (sistemas operacionais baseados no *UNIX*); 2) *Microsoft RPC* (sistema operacional *Windows*™); 3) *DCE RPC* (padrão *X/Open* suportado pela *Open Software Foundation - OSF*). A tecnologia *RPC* opera com o mesmo princípio que *Java RMI* (*Remote Method Invocation*). Contudo, ele difere de *RMI* em diversos aspectos chaves:

1. Em *RPC* não existe contexto de objetos, funções individuais são chamadas através de redes como uma *API* remota. A tecnologia *RPC* trabalha sobre uma *API* legada baseada em *C*, e não existe um caminho implícito para reter o estado entre

chamadas; a tecnologia *RMI* acopla isto naturalmente através de contexto de objetos.

2. Não existe a possibilidade de transferir comportamento com dados via *RPC*. A tecnologia *RMI* e a linguagem *Java* permitem ao programador construir uma classe que chama funções sem prévio conhecimento.

1.3.2 RMI

A Invocação de Métodos Remotos (*Remote Method Invocation - RMI)* é o padrão *Java* para objetos distribuídos. A tecnologia *RMI* permite a classes *Java* comunicarem-se com outras classes *Java* que podem estar localizadas em diferentes máquinas. Com um conjunto de *APIs* e um modelo para objetos distribuídos, a tecnologia *Java RMI* permite aos desenvolvedores construir sistemas distribuídos facilmente.

1.3.3 DCOM

A solução da *Microsoft* para o problema de objetos distribuídos é extender o popular *Component Object Model (COM)* com *Distributed COM (DCOM)*. Tal como *RMI*, a tecnologia *DCOM* isola o objeto desenvolvido dos detalhes de distribuição de um objeto. O maior esforço em construir objetos distribuídos *COM* está no desenvolvimento e registro do componente.

1.3.4 CORBA

O CORBA (Arquitetura de Agente de Solicitação de Objetos Comuns) é um padrão para objetos distribuídos [8]. A arquitetura deste padrão possibilita mecanismos pelos quais um objeto pode enviar requerimentos ou receber respostas para/de outro objeto no sistema distribuído, de uma forma transparente, como definido pelo *ORB* (*Object Request Broker*) do *OMG* (*Object Management Group*). O *ORB*, sob tal arquitetura, é

uma aplicação que possibilita interoperabilidade entre objetos, construídos em (possivelmente) linguagens diferentes, executados em (possivelmente) diferentes máquinas, em ambientes heterogêneos distribuídos. Em *CORBA*, o termo "*client*" ou cliente é um componente de *software* que pode executar sobre qualquer "nó computacional" de uma rede de computadores distribuídos; ou também um computador cliente usuário-final ou um computador servidor. Um cliente *CORBA* é uma definição lógica de *software* [8] que atua de modo que a chamada do cliente sobre um "servidor de serviços" fornece o acesso remoto aos dados e funções. Também em *CORBA*, o termo "server" ou servidor é uma definição lógica, porque ele também pode executar sobre qualquer nó computacional de uma rede de computadores distribuídos.

1.4 Contribuições

Este trabalho apresenta como contribuição técnica uma extensão do ambiente *Hydragyrum* que permite o desenvolvimento de modelos para elementos distribuídos de rede. Os modelos compõem um projeto *Hydragyrum* e através deles ocorre a troca de mensagens no ambiente. Conseqüentemente, com os elementos distribuídos, há condições de simular de forma mais realista o comportamento do tráfego de dados em redes complexas. Esta solução possibilitará também o desenvolvimento de estratégias para reduzir o tempo total da simulação.

O procedimento padronizado para realizar a interoperabilidade entre diversos projetos de simulação criados no ambiente *Hydragyrum,* conforme descrito no item *Resumo* deste trabalho, será alcançado usando o padrão *CORBA* no desenvolvimento da arquitetura *cliente/servidor* do *Hydragyrum*.

1.5 Organização do Trabalho

A discussão deste trabalho está dividida em 5 capítulos e 3 apêndices, sendo o Capítulo 1 este capítulo introdutório. O Capítulo 2 apresenta os fundamentos do trabalho com a descrição do ambiente de simulação *Hydragyrum* e da tecnologia *CORBA*. O Capítulo 3 apresenta a arquitetura *cliente/servidor* desenvolvida como extensão para o *Hydragyrum*. O Capítulo 4 descreve os resultados obtidos no trabalho. No Capítulo 5 são apresentadas as conclusões e as considerações finais do trabalho. O glossário dos termos utilizados durante o desenvolvimento do trabalho é descrito após o Capítulo 5. No Apêndice A, são ilustrados alguns cenários de diagramas de classes do ambiente de simulação *Hydragyrum*. Os diagramas de classes ilustrados neste apêndice são um guia para que o desenvolvedor possa compreender o relacionamento entre os principais objetos no *Hydragyrum.* No Apêndice B, é apresentada uma tabela com a métrica de linhas-de-código aplicada ao ambiente de simulação *Hydragyrum*. A métrica de linhas-de-código ajuda a avaliar o tamanho do *Hydragyrum* e fornece ao desenvolvedor uma estimativa de complexidade do sistema. Finalmente, no Apêndice C, é apresentada a listagem dos códigos-fonte exemplo para um projeto *Hydragyrum*. Um exemplo simplificado de projeto no *Hydragyrum* é um guia prático para o desenvolvedor iniciar o desenvolvimento de modelos de rede.

(Página Intencionalmente em Branco)

Capítulo 2

Fundamentos do Trabalho: O Ambiente de Simulação Hydragyrum e a Tecnologia CORBA

2.1 Introdução

Este capítulo apresenta uma visão geral dos principais fundamentos utilizados no trabalho: o ambiente de simulação *Hydragyrum* e a tecnologia *CORBA*. A estrutura conceitual do ambiente, as principais classes utilizadas na construção de modelos e a interface de linha de comando para o ambiente de simulação *Hydragyrum*, estão descritos no item 2.2. A tecnologia *CORBA* foi o padrão escolhido no desenvolvimento da arquitetura para a interoperabilidade de projetos no *Hydragyrum*. A justificativa para a escolha de *CORBA* é o fato desta tecnologia ser um padrão consolidado para aplicações distribuídas, prover uma infraestrutura de objetos reutilizáveis para a comunicação e possibilitar a implementação da comunicação distribuída em Java e C++. Para a compreensão desta tecnologia, noções introdutórias sobre o consórcio *OMG* da indústria de *software*, a arquitetura de gerenciamento de objetos, o núcleo *ORB* e a linguagem de definição de Interfaces *IDL* são conceitos brevemente abordados no item 2.3 deste capítulo.

2.2 O Ambiente Hydragyrum

O ambiente *Hydragyrum* é uma ferramenta utilizada para apoiar a simulação de redes em ambientes de simulação orientado a eventos discretos (*event-driven*). Uma visão geral sobre este ambiente está descrita nos itens a seguir. Informações detalhadas sobre este ambiente podem ser obtidas através de consulta ao trabalho entitulado "Ambiente de simulação de redes a eventos discretos" [4].

2.2.1 A Estrutura Conceitual do Ambiente Hydragyrum

Simular a operação de uma rede de comunicação consiste basicamente em projetar e implementar um conjunto de modelos que retrate as funcionalidades dos elementos de rede e desenvolver um ambiente em que estes modelos possam interoperar simulando o funcionamento da rede. A estrutura básica do ambiente de simulação *Hydragyrum* (Figura 2-1) é composta de três partes principais: o "Kernel"; os "Modelos dos Elementos de Rede"; e a "Interface de Linha de Comando". Além destas partes, também existe uma interface gráfica para o ambiente de simulação *Hydragyrum*.

O ***Kernel*** do *Hydragyrum* é responsável pelo gerenciamento e pela execução da simulação. O *Kernel* é uma biblioteca de ligação dinâmica (*"kernel.dll"*) com um conjunto de estruturas de dados (classes *Parameter*, *Table*, *Event*, *SMessage* e outras) e classes (*Kernel*, *Auxiliar*, *Command*, *Library*, *Scheduler* e *Dispatcher*), empregadas nas tarefas de gerenciamento, controle e interface. O *Kernel* atua como uma ponte entre os modelos e a *Interface de Linha de Comando*, carregando e descarregando os modelos necessários para uma determinada simulação. Também são responsabilidades do *Kernel* o agendamento e encaminhamento de eventos pelos modelos do sistema.

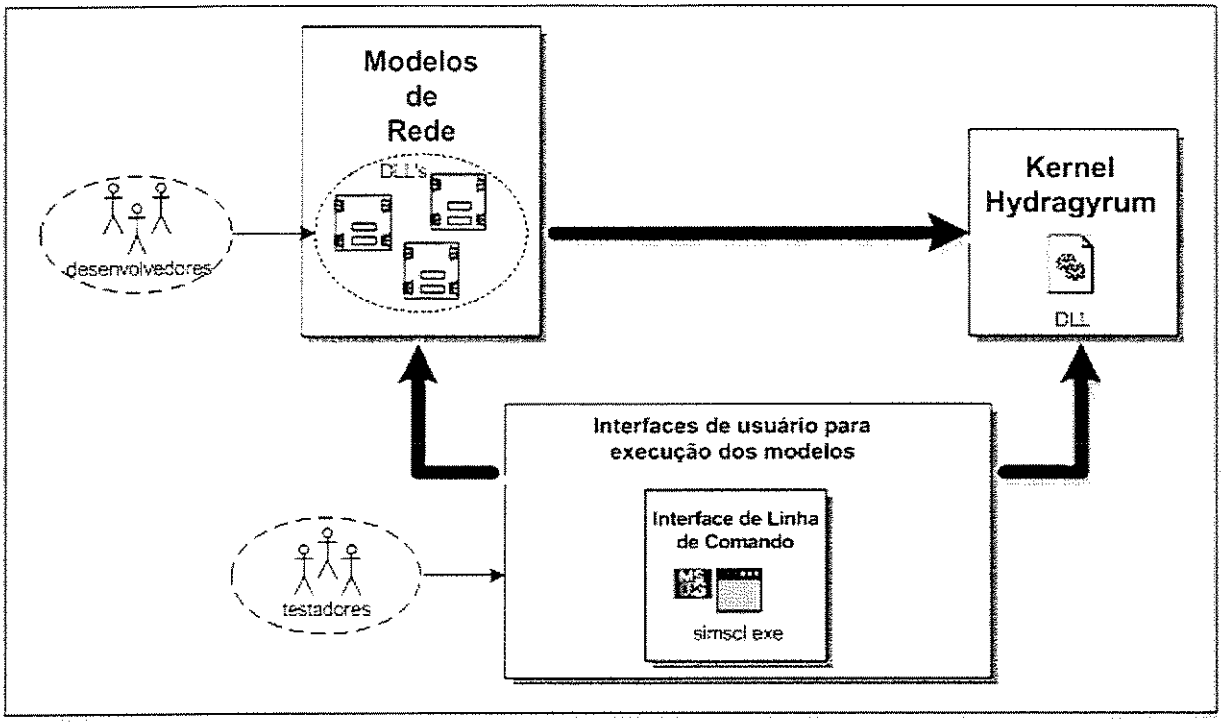

Figura 2-1: Configuração básica do ambiente de simulação Hydragyrum

Os **Modelos dos Elementos de Rede** formam os blocos do ambiente de simulação responsáveis pelo comportamento dos modelos. Todos os modelos do *Hydragyrum* são construídos como *DLLs* (bibliotecas de ligação dinâmica do sistema operacional *Microsoft Windows*^{TM} 32 bits). Os *Modelos dos Elementos de Rede* são divididos em três elementos básicos: os elementos de modelamento (classes *Block*, *Layer* e *Squeue*), que permitem a construção de novos modelos; os elementos de interconexão (classes *Connection*, *NetConnection* e *DataConnection*), que permitem a definição da conectividade entre os modelos; e os elementos para a representação de tipos e estruturas de dados (classes *Parameter*, *Table*, *Event*, *SMessage* e outras), empregados na criação de novos modelos para o ambiente de simulação.

A **Interface de Linha de Comando** (programa executável *"simscl.exe"*), através do interpretador de comandos do *Kernel* (classe *Command*), permite a realização de roteiros (*scripts*) de simulação dentro do ambiente *Hydragyrum*. O roteiro de simulação é descrito através de uma linguagem de simulação denominada *SCL* (*Simulation Command Language*) [4]. O interpretador de comandos do *Kernel* é responsável pela verificação da sintaxe dos comandos, tratamento de erros e pela chamada das rotinas associadas aos comandos, enviados pela *Interface de Linha de Comando* do simulador. Os comandos são traduzidos em ações dentro do *Kernel* tais como ler um arquivo com a configuração de uma simulação, inicializar a simulação e executar a simulação.

A simulação de um projeto no ambiente *Hydragyrum* é composta de quatro fases principais, conforme ilustrado na Figura 2-2. Na primeira fase realiza-se a carga do projeto (*Load*) para a memória do computador a partir da biblioteca de modelos. Nesta fase a função *Setup* é chamada em todos os modelos (blocos) e em todos os elementos de cada modelo (camadas). Na segunda fase realiza-se a inicialização (*Initiate*) para cada modelo, onde parâmetros de configuração podem ser passados na descrição do modelo. Nesta fase a função *Initiate* é chamada em todos os modelos (blocos) e em todos os elementos de cada modelo (camadas). Na terceira fase realiza-se a execução (*Run*) da simulação propriamente dita. A função *Run* é chamada em todos os modelos (blocos) e em todas as camadas de cada modelo onde existem eventos agendados à executar. Na quarta fase realiza-se o término da simulação (*Quit*), onde ocorre a desconexão do projeto com o servidor *Hydragyrum* e a remoção do modelo da memória. Nesta fase a função *Terminate* é chamada em todos os modelos (blocos) e em todas as camadas de cada modelo.

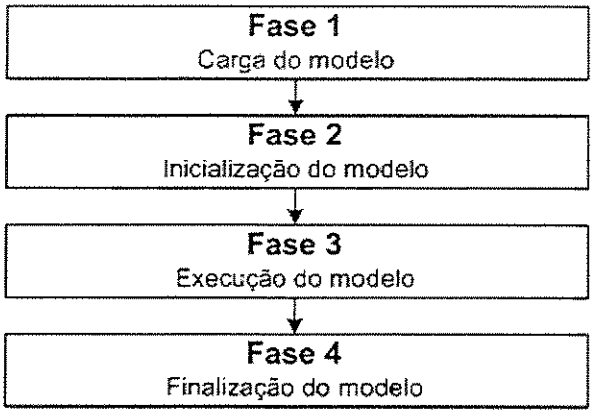

Figura 2-2: Fases da simulação de um modelo

2.2.2 Os Modelos para o Ambiente Hydragyrum

Os elementos de modelamento que permitem a adição de novos modelos ao ambiente de simulação são as classes *Block*, *Layer* e *Squeue*, ilustradas na Figura 2-3. Estas classes fornecem uma série de funcionalidades para integração dos modelos com o ambiente de simulação, tais como manipulação de estruturas de dados; criação, envio e processamento de eventos; envio de mensagens entre modelos; padronização para arquivos de entrada e saída na interface gráfica; e manipulação de parâmetros do modelo. Um modelo do ambiente de simulação é definido como um bloco (classe *Block*) onde são agregados camadas (classes derivadas da classe *Layer*) e sistemas de filas (classes derivadas da classe *Squeue*), que se comportam conforme as características do elemento que se pretende modelar. A classe *Block* define a interface para a carga da *DLL* do modelo para dentro do *Kernel*. A classe *Layer* deve abrigar os diferentes algoritmos que retratam o comportamento do modelo na simulação. A classe *Squeue* deve representar os elementos de armazenamento e serviço das mensagens na simulação. Um modelo do ambiente de simulação constitui uma *DLL*.

A classe *Kernel* (ilustrada na Figura 2-3) é a entidade central denominada de gerente da simulação, encarregada de efetuar a coordenação das diferentes funções necessárias à realização da simulação. Ela coordena as atividades das outras classes e responde as solicitações enviadas através das interfaces do ambiente de simulação. O gerenciador mantém os vários elementos da simulação armazenados dentro de suas estruturas de dados. Ele armazena as diversas instâncias dos modelos de um sistema simulado e as relações de interconexão entre estes modelos que refletem a topologia do sistema. O gerenciador armazena também um conjunto de parâmetros globais que podem ser manipulados e acessados pelos modelos durante a simulação.

A classe *Library* é a entidade que realiza o carregamento das bibliotecas de ligação dinâmicas (*DLLs*), as quais representam modelos construídos para o cenário de simulação desejado no ambiente *Hydragyrum*. A partir de uma solicitação do gerenciador, a classe *Library* carrega para a memória as *DLLs*, cada uma delas associada a um bloco do sistema. O bloco associado a uma instância do modelo é armazenado no gerenciador e pode agora acessar os parâmetros globais da simulação, mantidos pelo gerenciador. Quando o bloco associado ao modelo é apagado do ambiente de simulação, a biblioteca de modelos descarrega este modelo da memória.

A classe *Scheduler* é a entidade denominada de agendador de eventos dentro do ambiente de simulação. O agendador implementa uma fila de prioridades para os eventos a serem executados durante a simulação. Os eventos na fila de prioridades estão ordenados em ordem crescente de tempo de simulação, sendo que o evento que é mantido no topo da pilha é aquele que possui o menor tempo agendado para a simulação. Desta forma, quando for necessário processar um evento, é só retirá-lo do topo da pilha, uma vez que a estrutura implementada na fila de prioridade garante a correta ordenação dos eventos a serem simulados.

A classe *Dispatcher* realiza o encaminhamento dos eventos. Durante a execução da simulação, o gerenciador (classe *Kernel*) retira do topo da fila de prioridade (classe *Scheduler*) o evento que possui o menor tempo de execução agendado. Esse evento, é então repassado ao encaminhador, que realiza a função de entregar o evento ao modelo ao qual este evento se destina. Quando o evento chega ao modelo de destino, ele é executado e o modelo passa a deter o controle da simulação. Quando termina a execução do evento, o controle da simulação é devolvido ao gerenciador, que inicia novamente o ciclo de execução.

Após a construção dos modelos utilizando as classes base de modelamento, é necessário representar dentro do ambiente de simulação a maneira pela qual estes modelos estarão interconectados. Para representar a interconexão entre os modelos do ambiente de simulação, as classes *Connection*, *NetConnection* e *DataConnection* são utilizadas. Os elementos de interconexão representam as relações entre os modelos nos vários níveis (camadas) da hierarquia da rede simulada. As classes de interconexão definem a topologia da rede, o caminho e a direção do fluxo de eventos da simulação.

A classe *Connection* pode representar conexões físicas ou lógicas (pontos de acesso ao serviço) entre os blocos dos modelos da simulação. Um objeto da classe *Connection* é bidirecional [4]. A classe *Connection* é a classe sobre a qual todas as outras classes de conexão são construídas. Isto implica que, no momento em que se destrói um objeto da classe *Connection*, por exemplo, todas as conexões de rede (*NetConnection*) e conexões de dados (*DataConnection*) que usam o objeto da classe *Connection*, que está sendo apagado, são destruídas. A classe *NetConnection* representa um caminho unidirecional a ser percorrido na rede, composto pela associação de uma seqüência de objetos da classe *Connection*. No estabelecimento de um caminho pela rede de conexões na topologia a ser simulada (criação de um objeto da classe *NetConnection*), os objetos da classe *Connection* são ordenados e verificados automaticamente pelo

Kernel. A partir da especificação da fonte e do destino do caminho de rede, e baseando-se na informação de quais blocos e camadas estão conectados pelos objetos da classe *Connection*, o *Kernel* pode estabelecer de forma automática um caminho através da rede, interconectando dois blocos de modelo. A classe *DataConnection* pode representar as conexões de dados entre os aplicativos que geram a informação a ser transmitida na rede. O objeto da classe *DataConnection* é unidirecional, sendo composto pela associação de um ou mais objetos de conexões de rede *NetConnection* que estão disponíveis entre a fonte e o destino especificados.

As estruturas de dados do ambiente de simulação são utilizadas para armazenar as informações usadas pelo *Kernel* e pelos modelos durante a simulação. A classe *Sstring* é uma abstração de dados para trabalhar com vetores de caracteres alfanuméricos, sendo usada na maioria das trocas de nomes e de informação alfanumérica nos modelos. As classes de parâmetros (*Param*, *Parameter* e *Joker*) representam os tipos de dados suportados pelo simulador. A classe *Param* é responsável pelo armazenamento de um conjunto de objetos da classe *Parameter* indexados em um dicionário pelo nome do parâmetro. A classe *Parameter* apenas adiciona ao dado da classe *Joker* um conjunto de *Sstrings*, as quais representam uma descrição do parâmetro, informação de formato e uma opção. A classe *Joker* descreve o local de armazenamento dos tipos de dados suportados pelos parâmetros. Os tipos de dados primitivos do ambiente de simulação suportados são *long*, *int*, *double*, *Sstring* e *unsigned int*, e estão definidos no arquivo *Simdefs.h*, que pode ser estendido para outros tipos de dados.

Os eventos e as mensagens trocadas entre os elementos da simulação são representados pelas classes *Event* e *SMessage*. A classe *Event* representa os eventos trocados na simulação. Estes eventos contêm as informações do tempo em que devem ser processados, para qual elemento da simulação se destinam e uma *string* que identifica o tipo do evento gerado. Os eventos são gerados por funções específicas

dentro dos blocos (classe *Block*), camadas (classe *Layer*) e sistemas de filas (classe *Squeue*) dos modelos. A classe *SMessage* é a classe base para a derivação de novas mensagens a serem transportadas pelos eventos do simulador.

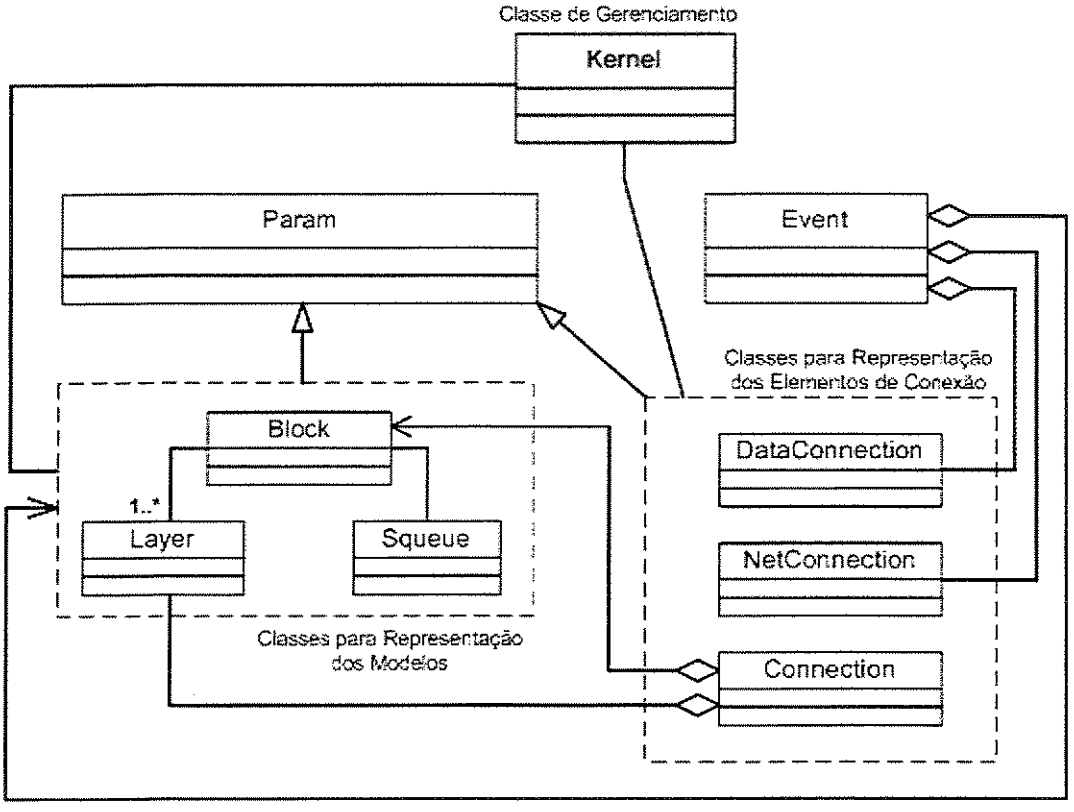

Figura 2-3: Relacionamento entre as principais Classes do Ambiente Hydragyrum

2.2.3 A Interface de Linha de Comando do Ambiente Hydragyrum

A interface de linha de comando do ambiente *Hydragyrum*, definida pelo programa denominado "*simscl.exe*" e ilustrada na Figura 2-4, permite que o usuário utilize os comandos definidos na linguagem de comandos do ambiente de simulação (*SCL*) para configurar e executar as simulações. A utilização desta interface possibilita a execução

de arquivos de *script*, que são roteiros de simulação, com uma série de comandos a serem executados pelo ambiente de simulação.

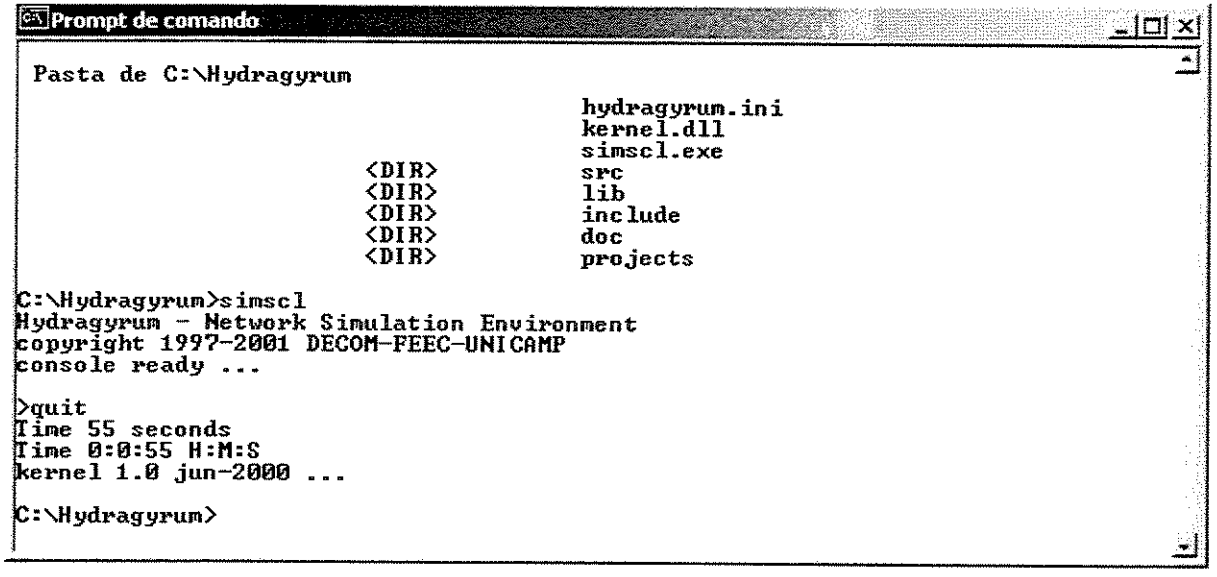

Figura 2-4: Interface de linha de comando do ambiente Hydragyrum

A execução de um roteiro de simulação no interpretador de comandos ("*simscl.exe*"), consiste na seqüência de entrada dos seguintes comandos:

load *sim1.scl*

initiate

run *100*

reset

quit

No exemplo acima, o comando *load* irá carregar o arquivo descritor denominado "*sim1.scl*" para a memória do ambiente de simulação. Este arquivo descreve o projeto a ser simulado, que é basicamente composto de modelos, conexões e parâmetros. O

comando *initiate* irá executar uma chamada à função de inicialização dos modelos descritos no arquivo de descrição da rede que será simulada. O comando *run* realiza a execução da simulação até que o relógio de tempo (100 segundos de tempo simulado neste caso) seja atingido ou que não existam mais eventos a serem processados. O comando *reset* realiza a limpeza de quaisquer eventos que tenham ficado pendentes no agendador de eventos. O comando *quit* realiza o encerramento do interpretador de comandos.

2.3 A Tecnologia CORBA

2.3.1 OMG

O *Object Management Group* (*OMG*) é um consórcio internacional da indústria de *software* fundado em maio de 1989 com o propósito de criar padrões para possibilitar a interoperabilidade e portabilidade das aplicações distribuídas utilizando a tecnologia de objetos. Diferentemente de outras organizações como a *Open Software foundation* (*OSF*), o *OMG* não visa a produção de *software*, somente especificações. Estas especificações são criadas a partir de idéias e tecnologias propostas e discutidas pelas suas organizações membro. A grande força destas especificações reside no fato de que a maior parte das grandes empresas comerciais envolvidas com a computação distribuída está entre as centenas (mais de 700) de organizações internacionais filiadas ao *OMG*.

Os trabalhos de padronização do *OMG* são executados sobre um modelo conceitual denominado *core object model* e uma arquitetura geral de referência denominada *Object Management Architecture* (*OMA*) que busca definir de forma abstrata as várias funcionalidades necessárias à computação distribuída. No modelo *OMA*, cada componente de *software* é representado como um objeto (Figura 2-5). A *OMA* classifica

os objetos conforme três categorias: Objetos de Aplicação (*Application Objects*), Serviços de Objetos (*Object Services*) e Facilidades Comuns (*Common Facilities*). Objetos de Aplicação estão relacionados às aplicações específicas do usuário que serão integradas. As Facilidades Comuns compreendem as funcionalidades de propósito geral que estarão disponíveis no ambiente distribuído. Os Serviços de Objetos fornecem as funcionalidades básicas do ambiente distribuído como ciclo de vida de objetos, consulta e transações. Objetos comunicam-se com outros objetos via *Object Request Broker* (*ORB*), que é o elemento chave de comunicação. O *ORB* fornece mecanismos pelos quais objetos são ativados, enviam requisições e recebem respostas de forma transparente.

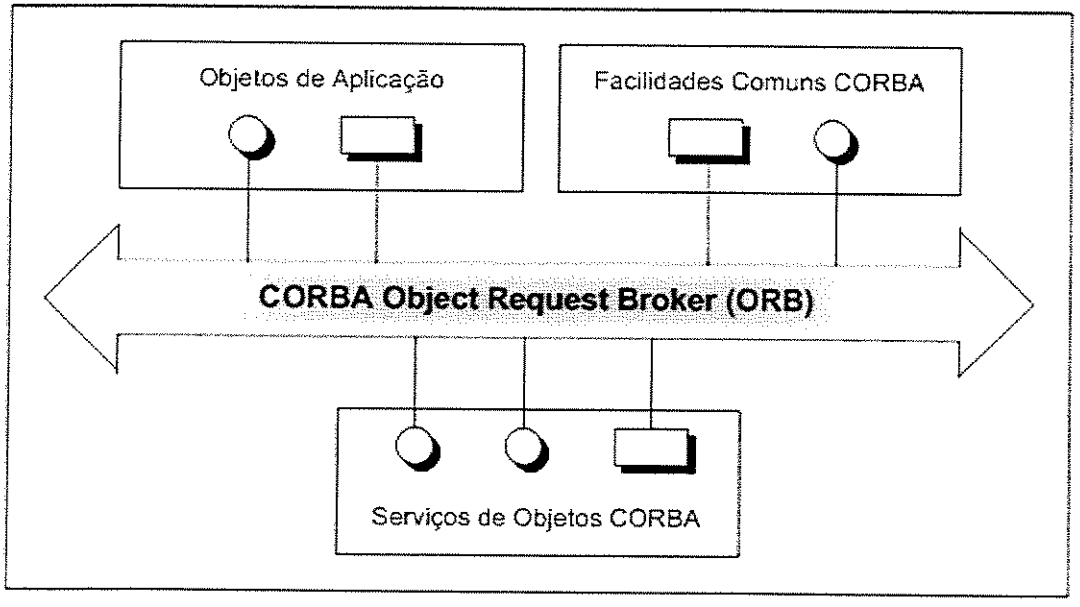

Figura 2-5: Arquitetura de Gerenciamento de Objetos CORBA – Modelo Referência de Comercialização da OMG

Em outubro de 1999 a *OMG* publicou a revisão 2.3.1 da especificação *Common Object Request Broker Architecture* (*CORBA*) com a descrição das interfaces e serviços que um *ORB* deveria possuir. Esta é a especificação referência atualmente adotada para os diversos *ORBs* que implementam a especificação *OMG CORBA*.

2.3.2 CORBA

O *CORBA* é uma especificação de uma arquitetura e interface que permite que aplicações façam solicitações aos objetos, de uma forma transparente e independente, indiferente à linguagem, sistema operacional ou considerações de localização. Sendo assim, o *CORBA* pode ser visto como um paradigma de comunicação, baseado na tecnologia de orientação a objetos, para ambientes de computação distribuídos. Atualmente existem várias implementações de *CORBA* disponíveis no mercado.

CORBA utiliza um modelo orientado a objetos e define um módulo intermediário entre clientes e servidores, o *Object Request Broker* (*ORB*). Neste modelo, clientes enviam requisições para o *ORB* solicitando algum serviço para ser executado por qualquer servidor que possa responder as suas requisições. Somente o *ORB* necessita conhecer a localização dos clientes e servidores na rede, com isso a localização do servidor é transparente para o cliente e por sua vez, a do cliente para o servidor. Uma biblioteca de funções, chamada de *Basic Object Adapter* (*BOA*), é utilizada pelo programa do servidor para localizar, inicializar as implementações e invocar o método apropriado para responder a requisição do cliente.

No *CORBA* existe uma separação formal entre o cliente e o servidor. Isto é, o cliente, utilizando a interface orientada a objetos do *CORBA*, não necessita saber como o servidor realiza suas tarefas, ele só necessita saber como chamar o servidor. *CORBA* alcança esta separação através da restrição da comunicação entre o cliente e o servidor para um tipo de mensagem chamada requisição. Cada requisição que é

enviada do cliente para o servidor contém uma operação a ser executada e um objeto específico sobre o qual a operação deve ser executada. Os objetos e operações que um cliente pode requisitar e que um servidor pode responder são definidos por uma interface comum tanto para o cliente como para o servidor. Objetos podem alternar-se entre papéis clientes e papéis servidores. Um objeto está em papél cliente quando ele é originador de uma invocação de objeto. Um objeto está em um papél servidor quando ele é o receptor de uma invocação de objeto. Objetos servidores são chamados de implementação dos objetos. Na terminologia orientada a objetos uma interface é muito similar à definição de uma classe que contém as características e o comportamento de um tipo de objeto.

O modelo *CORBA* oferece uma série de benefícios que facilitam a integração de aplicações em um ambiente distribuído. Este modelo permite que os projetistas de sistemas distribuídos tirem proveito de técnicas orientadas a objeto como encapsulamento e herança. Estas técnicas permitem que os objetos sejam definidos como "caixas pretas" capazes de executar determinadas tarefas, sem que o sistema necessite conhecer o funcionamento interno destes objetos. Outra vantagem é a definição mais clara das interfaces entre as partes do sistema, as quais podem ser alteradas sem afetar o sistema como um todo. Assim, consegue-se uma melhor modularidade, já que os módulos passam a operar com mais independência. Os sistemas tornam-se mais extensíveis já que novas características podem ser adicionadas através da definição de um novo conjunto de operações. Outro ponto importante é que a especificação do *CORBA* permite a implementação em linguagens de programação bastante difundidas atualmente como "*C*", "*C++*" e "*Java*" [6].

No modelo *cliente/servidor* tradicional, o cliente e o servidor são modelados cada qual como sendo um único processo. No *CORBA* os clientes são, tipicamente, um único processo, contudo, os servidores podem ou não constituírem um processo único. *CORBA* não assume um relacionamento um-para-um entre clientes e servidores.

Múltiplos servidores podem trabalhar com um único cliente ou um único servidor pode trabalhar com múltiplos clientes. E, como mencionado anteriormente, clientes e servidores comunicam-se via *ORB*, não diretamente.

2.3.3 Núcleo ORB

Um *ORB* é um componente distribuído que pode comunicar com outros *ORB's* para prover o ciclo de vida dos objetos distribuídos através de serviços multiplataformas. O *ORB* simplifica a programação distribuída porque ele desacopla o cliente dos detalhes das invocações dos métodos. O *ORB* é uma biblioteca de classes que habilita a comunicação a baixo nível entre aplicações *Java IDL* e outras aplicações *CORBA* complementares. O mecanismo *ORB* permite que objetos sejam criados e invocados remotamente sobre outra rede de computadores provendo transparência de sistema operacional e rede. Quando um cliente invoca uma operação, o *ORB* é responsável por encontrar a implementação do objeto, ativando transparentemente se necessário, disponibilizando a requisição do objeto e retornando qualquer resposta para a chamada.

O núcleo do *ORB* fornece a representação básica dos objetos e das requisições de comunicação. Ele é responsável pelo transporte de uma requisição de um cliente para o adaptador apropriado do objeto alvo. Para que um cliente possa enviar uma requisição a um objeto, ele deve possuir uma referência para aquele objeto. Um *ORB* utiliza as referências para identificar e localizar objetos para que ele possa enviar as requisições diretamente a eles. Enquanto um determinado objeto referenciado existir, o *ORB* permitirá ao cliente que possua a referência enviar requisições de serviços para ele. Para que as referências possam ser armazenadas pelos clientes, o *ORB* possui a funcionalidade de conversão de referências em valores do tipo texto (*string*) e vice-versa. Esta funcionalidade permite que relacionamentos entre objetos possam ser guardados por aplicações que os utilizem.

Como o *CORBA* visa suportar uma grande diversidade de características de objetos, como granularidade, estilo de implementação, mecanismos de ativação, ciclo de vida, entre outras, torna-se muito difícil para o núcleo do *ORB* fornecer uma interface única que seja apropriada e eficiente para todos os objetos. Devido a este fato o *ORB* é estruturado em camadas com componentes acrescentados acima do seu núcleo, os quais fornecem interfaces que escondem as diferenças existentes. Por exemplo, um determinado adaptador de objetos deve ser utilizado com implementações de objetos que pertençam a um determinado domínio de aplicação, com interfaces específicas para o fim desejado, permitindo desta forma que o núcleo do *ORB* permaneça simples.

A estrutura de um *ORB* individual é ilustrado na Figura 2-6. O programa cliente realiza a ligação diretamente para o *stub* da interface *OMG IDL*. Da perspectiva do cliente, o *stub* atua como uma chamada a uma função local. O esqueleto (*skeleton*) é a implementação correspondente do lado servidor da interface *OMG IDL*. Quando o *ORB* recebe as requisições, o esqueleto provê uma chamada de volta (*callback*) para a implementação da função fornecida pelo servidor. Quando o servidor completa o processamento das requisições, o esqueleto e o *stub* retornam o resultado para o programa cliente, juntamente com qualquer informação de exceção. Exceções podem ser geradas pelo servidor ou pelo *ORB* em caso de erros. O protocolo padrão é o *IIOP (Internet Inter-ORB Protocol)*, que usa o Protocolo Geral Inter-ORB (*GIOP*) para definir as mensagens que são trocadas através de uma rede *TCP/IP*.

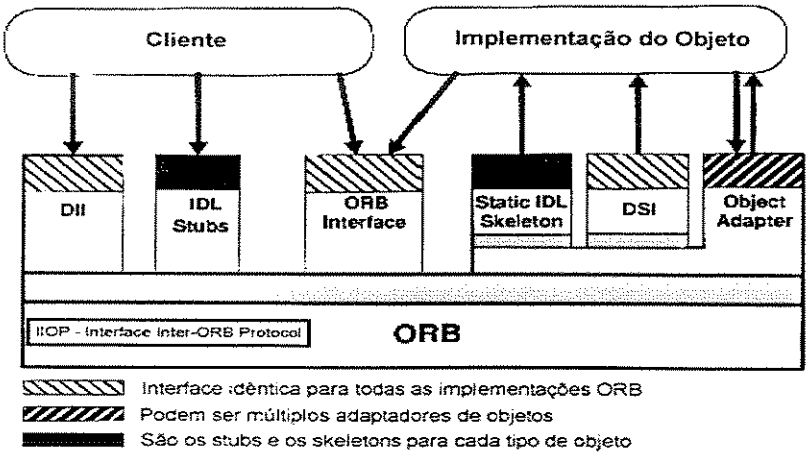

Figura 2-6: Estrutura de um ORB individual [6]

2.3.4 Linguagem de Definição de Interface IDL

A linguagem de definição de interface - *Interface Definition Language* (*IDL*) - define tipos de objetos através da especificação de suas interfaces. Todos os serviços são especificados utilizando esta linguagem declarativa que enfatiza a separação entre interface e implementação. A *IDL* é independente da linguagem de programação e dos protocolos de comunicação, sendo utilizada como um meio para descrever tipos de dados. A *IDL* pode ser utilizada como uma linguagem universal para definição de componentes de uma arquitetura de software. A sintaxe da *OMG IDL* é derivada da linguagem *C++*, onde as funcionalidades ligadas à implementação são subtraídas. Por outro lado, são acrescentadas novas funcionalidades necessárias para especificação de ambientes distribuídos. *IDL* possui tipos de dados básicos (como *short*, *long*, *float*, *double* e *boolean*), tipos estruturados (como *struct* e *union*) e tipos template (como *sequence* e *string*). Estes tipos são utilizados para definir os tipos dos argumentos e valores de retorno das operações. Estas operações são utilizadas na declaração das interfaces para definir os serviços fornecidos pelos objetos. *IDL* ainda fornece um

construtor de módulos que pode conter interfaces, definição de tipos e outros módulos para controlar escopo de nomes (Figura 2-7).

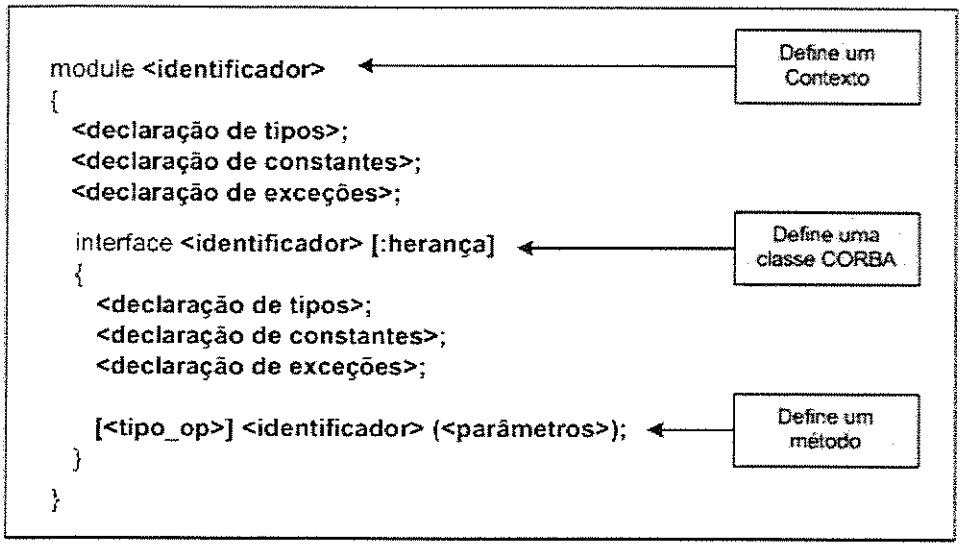

Figura 2-7: Estrutura de um Arquivo CORBA IDL

A *IDL* fornece o mecanismo de herança através do qual as interfaces derivadas herdam as operações e tipos definidos na interface base, contudo não existe a noção de herança de implementação. Na herança de interfaces, um objeto de uma interface derivada se compromete a suportar aquela interface, contudo, não existe compromisso na forma de como será feita a implementação das operações.

Quando um código *IDL* é compilado, ele gera *stubs* para o cliente e esqueletos (*skeletons*) para o servidor. Os *stubs* representam o mapeamento entre a linguagem de implementação do cliente e do *ORB*. Do ponto de vista do cliente os *stubs* atuam como chamadas a funções locais. Transparentemente, os *stubs* fornecem uma interface para o *ORB* que executa as codificações e decodificações necessárias para que os parâmetros das operações possam ser enviados pelos canais de comunicação. O esqueleto do servidor torna possível para o *ORB* e o adaptador de objetos traduzir a

requisição do cliente para um método específico no servidor. O esqueleto do servidor e o *stub* do cliente são responsáveis pelo retorno do resultado (se houver) para o cliente, juntamente com possíveis informações de ocorrência de exceções. A Figura 2-8 ilustra o diagrama de desenvolvimento para o compilador *IDL ORBACUS* na linguagem de programação *Java*.

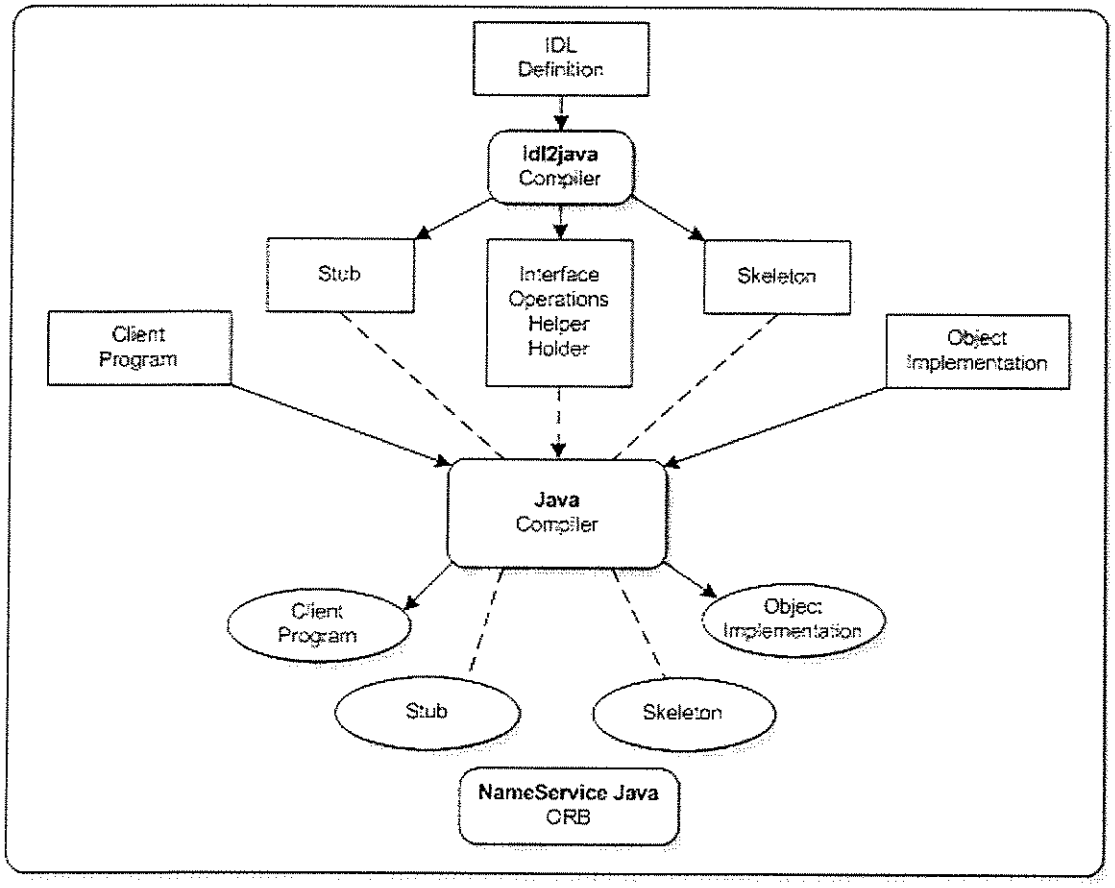

Figura 2-8: Diagrama de desenvolvimento para o compilador IDL2Java ORBACUS

2.3.5 Repositório de Interfaces

O repositório de interfaces fornece um meio de armazenamento persistente para declarações de interfaces *IDL*. O repositório de interfaces possui serviços para navegação pela hierarquia de interface de um determinado objeto e fornecimento das descrições de todas as operações que o objeto suporta.

Os repositórios de interfaces podem ser utilizados para múltiplos propósitos. Programas de visualização podem navegar pelas informações contidas no repositório de interface para auxiliar desenvolvedores na tarefa de localizar componentes de *software* potencialmente reutilizáveis. *ORBs* podem utilizá-los para verificação de tipos de parâmetros de operações em tempo de execução. A função principal destes repositórios é fornecer as informações necessárias para enviar requisições utilizando a Interface de Invocação Dinâmica - *Dynamic Invocation Interface* (*DII*).

2.3.6 Interface de Invocação Dinâmica - DII

Como dito anteriormente, a compilação das declarações *IDL* geram *stubs* que permitem aos clientes requisitarem operações a objetos conhecidos. Contudo, algumas aplicações podem necessitar requisitar operações de objetos sem que elas possuam conhecimento da interface do objeto em tempo de compilação. Como exemplo, podemos ter uma aplicação que manipule objetos gráficos na tela do computador. A partir de uma lista de referências de objetos os usuários poderiam navegar pelo repositório de interfaces, aprender a respeito das operações suportadas por cada objeto, e então invocar (requisitar) uma operação para verificar como o objeto será apresentado na tela. Esta aplicação somente necessitará saber como navegar pelas informações no repositório de interfaces e perguntar ao usuário pelos dados necessários para preenchimento dos parâmetros da operação desejada. Assim, qualquer operação que o usuário desejar, poderá ser requisitada pela *DII*. Cabe

ressaltar que esta flexibilidade gera perda de eficiência na execução da requisição devido a sobrecarga na comunicação *cliente-ORB* necessária para concluir uma requisição.

2.3.7 Adaptadores de Objetos

CORBA permite que as implementações de objetos variem amplamente. Em alguns casos, múltiplas interfaces *IDL* podem ser implementadas por um único programa servidor, enquanto que em outros casos uma única interface *IDL* pode ser implementada por uma série de programas, um para cada operação. Estas implementações podem ser programas pré-existentes, os sistemas legados (*legacy systems*), ou podem ser sistemas *OO* (Orientados-a-Objetos) especificamente desenvolvidos para operar com o *ORB*. O *ORB* oferece esta flexibilidade para permitir a integração de aplicações pré-existentes sem restringir os novos objetos componentes em um conjunto limitado de critérios de implementação.

O Adaptador de Objetos - *Object Adapter* (*OA*) - é o meio pelo qual vários e diferentes tipos de implementações de objetos utilizam o *ORB*. O adaptador de objetos fornece serviços como: registro de implementações; geração e interpretação de referências para objetos; mapeamento de referências de objetos para as implementações correspondentes; ativação e desativação de implementações de objetos; invocação de métodos de objetos via esqueleto ou *DII*; e coordenação da segurança.

Dependendo do *ORB*, um *OA* pode escolher entre fornecer estes serviços através da delegação para o *ORB* ou executá-los sozinho. Em ambos os casos a implementação do objeto não se preocupa com esta diferença porque ela só possui interface para o *ORB*. Cada *ORB* deve possuir um *OA* genérico, denominado Adaptador Básico de Objetos - *Basic Object Adapter* (*BOA*) - para que as implementações de objetos possam ser suportadas como programas isolados. Um *BOA* deve possuir serviços flexíveis para

acomodar diferentes tipos de implementações de objetos. O *BOA* pode manipular um programa por método, por objeto, ou um programa compartilhado por múltiplas instâncias de um objeto. O *BOA* faz a diferenciação entre um servidor que corresponde a uma unidade de execução e um objeto que implementa um método ou interface.

2.3.8 Serviços CORBA

O *ORB*, por si só, não executa todas as tarefas necessárias para os objetos interoperarem. Ele só fornece os mecanismos básicos. Outros serviços necessários são oferecidos por objetos com interface *IDL*, que a *OMG* vem padronizando para os objetos de aplicação poderem utilizar. O serviço de nomes, por exemplo, é basicamente um serviço de localização de objetos que permite um objeto descobrir outros objetos através de identificadores ou nomes. O serviço de controle de concorrência fornece um gerente de "locks" que coordena múltiplos acessos a recursos compartilhados, permitindo resolução de conflitos de acesso.

O serviço de tempo especifica uma interface de serviço de tempo que permite, basicamente, aos objetos das aplicações obterem o tempo atual e executarem comparações com este tempo. O serviço de tempo também estende o serviço de eventos, especificando uma interface de serviço de eventos temporizados. Esse novo serviço gerencia objetos manipuladores de eventos temporizados. Um cenário típico de utilização desses serviços é um usuário criar um canal de eventos e registrá-lo como destino dos eventos gerados por um manipulador de eventos temporizados. Dessa forma, o usuário pode usar o objeto manipulador de eventos temporizados para criar eventos temporizados quando desejar. Basicamente, o serviço de eventos temporizados permite que um objeto possa "ligar" o tempo para disparo do evento (indicando inclusive que esse disparo pode ser periódico). Permite também que um

objeto que recebeu a notificação de um evento possa determinar o tempo em que o evento ocorreu.

2.4 Conclusão

Neste capítulo são descritos os fundamentos utilizados neste trabalho. Estes fundamentos são aplicados no desenvolvimento da arquitetura *cliente/servidor* para a interoperabilidade de projetos no ambiente *Hydragyrum*, apresentada em detalhes no capítulo a seguir.

(Página Intencionalmente em Branco)

Capítulo 3

A Arquitetura Cliente/Servidor para o Ambiente Hydragyrum

3.1 Introdução

Este capítulo descreve um conjunto de requisitos que visam esclarecer sob vários pontos de vista a arquitetura *cliente/servidor* desenvolvida para o ambiente *Hydragyrum*. Baseado nestes requisitos, o projeto da arquitetura implementada e alguns cenários de simulação distribuídos que podem ser contemplados na execução de projetos são apresentados.

A arquitetura de *software* diz respeito a estrutura, comportamento, uso, funcionalidade, desempenho, flexibilidade, reutilização, abrangência e restrições tecnológicas. A arquitetura de *software* abrange: as decisões significativas sobre a organização de um sistema de *software*; a seleção dos elementos estruturais e das interfaces através das quais o sistema é composto em conjunto com o comportamento, como especificado na colaboração entre esses elementos; e o estilo de arquitetura que guia essa organização. Em uma arquitetura cliente/servidor, o servidor controla o fluxo dos dados e informações de entrada e saída nos computadores clientes. Nesta arquitetura, alguns computadores são dedicados a servir outros.

A arquitetura cliente/servidor é o modelo de interação em processamento de dados distribuídos, no qual um programa em um local envia uma solicitação a um programa em outro local e espera uma resposta. O programa que solicita é chamado de cliente, enquanto o programa que responde chama-se servidor. O termo cliente/servidor

significa qualquer sistema que distribua o processamento de dados entre dois ou mais computadores distintos. Por esta definição, os sistemas cliente/servidor abrangem qualquer aplicativo que tenha uma parte de interface de usuário ("*front-end*") rodando de modo local no cliente e uma parte de processamento rodando no servidor ("*back-end*"). As principais vantagens de um sistema *cliente/servidor* são provenientes da divisão do processamento entre o sistema cliente e o sistema servidor. Outra grande vantagem de um sistema *cliente/servidor* é a preservação da integridade dos dados. O sistema de gerenciamento de banco de dados pode fornecer qualquer número de serviços que protejam os dados, como o armazenamento de arquivos criptografados, *backup's* de tempo real em fitas, enquanto o banco de dados está sendo acessado. Dentre as características de um ambiente em arquitetura *cliente/servidor*, podemos destacar:

- Desempenho: O modelo cliente/servidor consiste em um processo cliente que realiza solicitações de serviços a um processo servidor, reduzindo o processamento na máquina cliente.
- Interoperabilidade: A parte cliente e a parte servidor podem operar em diferentes plataformas.
- Modularidade: Tanto a plataforma do cliente como a do servidor podem ser atualizadas independentemente.
- Transparência: O servidor pode atender a vários clientes simultâneamente e os clientes podem acessar vários servidores.
- Conectividade: Os sistemas cliente/servidor incluem algum tipo de capacidade de operar em rede.
- Encapsulamento de Serviços: O servidor é um "especialista" na requisição de serviços. Servidores podem ser atualizados sem afetar o cliente.

3.2 Os Requisitos do Sistema

O ambiente de simulação *Hydragyrum* deve possibilitar a troca de mensagens entre projetos de simulação numa rede de elementos distribuídos. Para tal, é importante a escolha de uma tecnologia padronizada para realizar a interoperabilidade na comunicação distribuída. Um componente de *software* concebido como *servidor* ("Hydragyrum Server") irá possibilitar o encaminhamento de mensagens distribuidas entre projetos de simulação. Este *servidor* será útil para dar transparência quanto à localização física na rede de cada projeto. Apenas a localização do servidor será conhecida e necessária pelos diversos projetos distribuídos. O ocultamento da localização física de cada projeto distribuído possibilita um maior controle da comunicação quanto a segurança no aspecto de identificação dos projetos. A identificação de cada projeto junto ao servidor possibilita a associação de *perfis* relacionados a permissões na comunicação dos projetos. Com a separação funcional do componente *cliente* e do componente *servidor*, múltiplas instâncias de clientes ou servidores poderão ser concebidas, ficando a cargo do *analista/desenvolvedor* a escolha mais adequada do cenário de simulação desejado. Esta separação funcional também busca isolar a eventual possibilidade que um problema com um projeto *cliente* perturbe ou retarde a comunicação delegada ao *servidor*.

3.2.1 Descrição Detalhada dos Requisitos

Três papéis para usuários válidos podem ser contemplados inicialmente para um projeto na arquitetura do ambiente *Hydragyrum cliente/servidor*: o usuário testador, o usuário moderador e o usuário *Web*. A Figura 3-1 mostra o relacionamento dos atores no sistema com as interfaces distribuídas de comunicação. O ator denominado *usuário/Web* usa a interface *Model*Web que apresenta acesso limitado a operações via *Internet*. O ator denominado *usuário/Moderador/HydragyrumServer* usa a interface que disponibiliza acesso ao gerenciamento das simulações. O ator denominado

usuário/Modelo/Hydragyrum usa a interface *Model* e somente tem acesso aos dados do escopo do modelo de simulação.

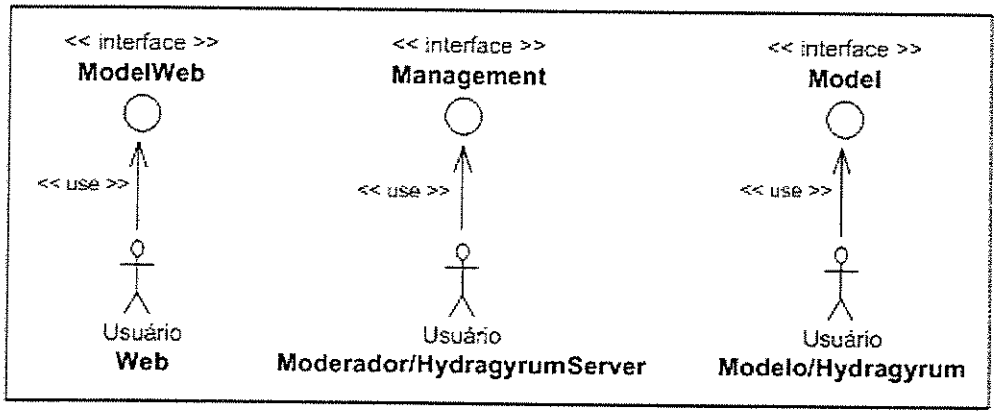

Figura 3-1: Interfaces de Comunicação x Usuários

O perfil do usuário *testador* é responsável por gerenciar um projeto local. Desta forma, é concedido a este usuário, uma vez que a conexão e a autenticação no servidor *Hydragyrum* é estabelecida, executar os modelos e obter os resultados da simulação. O perfil do usuário *moderador* é responsável por gerenciar múltiplos projetos que interoperam numa rede de computadores. O usuário *moderador* realiza previamente o cadastro do projeto a ser simulado no banco de dados. Para cada projeto, estará descrito quais são os modelos, as camadas, os usuários testadores e, opcionalmente, os usuários *Web*. O perfil do usuário *Web* possibilita a comunicação com os modelos de rede através do envio de mensagens via *Web*.

O servidor *Hydragyrum* (*"Hydragyrum Server"*) é responsável por gerenciar a seqüência de ocorrência de envio das mensagens, autenticar os usuários, conectar os modelos ao *ORB*, registrar os modelos juntamente ao servidor de nomes *CORBA* e situar cada modelo em um determinado escopo de simulação.

Os *casos-de-uso* no ponto de vista do servidor, conforme ilustrado na Figura 3-2, compreendem os requisitos necessários para que projetos de simulação distribuídos possam realizar a troca de mensagens. O servidor é responsável por gerenciar o recebimento de mensagens dos modelos, armazenando as mensagens recebidas em uma lista de objetos do tipo mensagem. Como é o ambiente de simulação de um determinado projeto que tem controle sobre a seqüência de execução dos eventos, o servidor abstrai requisitos de seqüência de execução dos projetos e de encaminhamento das mensagens, sendo delegado aos projetos de simulação o total controle sobre esta atividade. A seqüência de execução dos eventos pelos projetos pode estar armazenada em um banco de dados e compartilhada entre todos os projetos que necessitam destas informações.

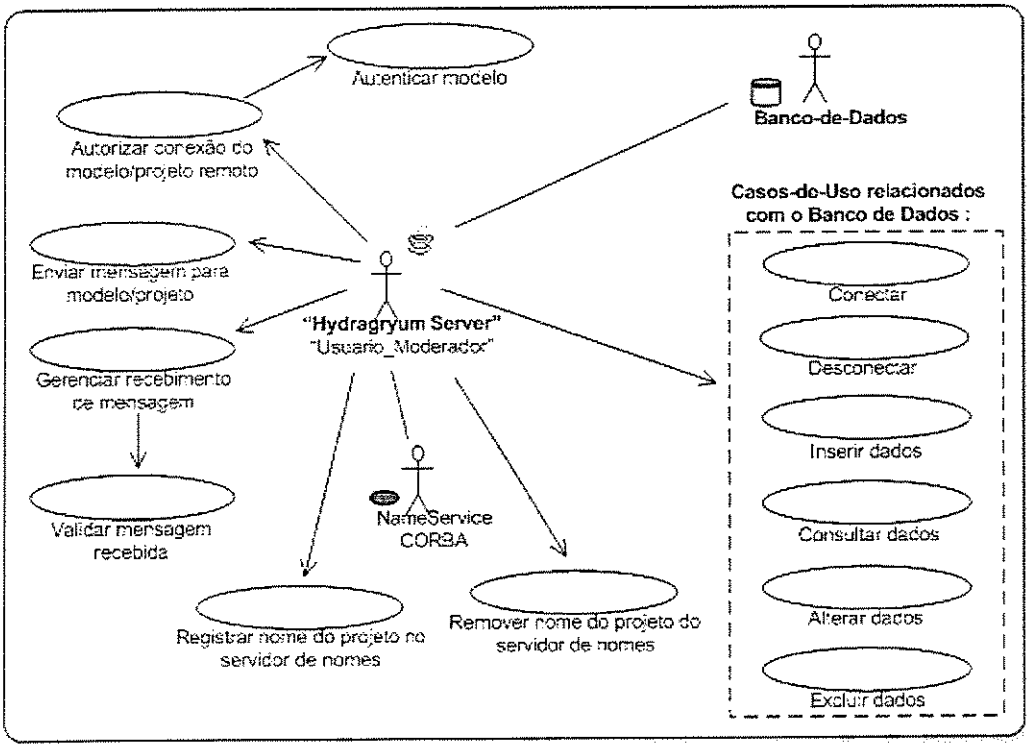

Figura 3-2: Diagrama de Casos-de-Uso para o Servidor

Notação Gráfica UML

A validação de uma mensagem recebida consiste em verificar se esta é um objeto válido de mensagem de um determinado modelo. Um tratamento interno ao servidor bloqueia o tráfego de mensagens nulas, para evitar exceções *CORBA*, uma vez que estas podem provocar falhas no sistema. O servidor pode conectar-se a um banco de dados (*MySQL*) para eventualmente inserir, alterar, consultar ou excluir dados em alguma tabela, como por exemplo o controle do histórico das simulações. Ao encerrar a execução do servidor, a remoção do nome do projeto junto ao servidor de nomes é efetuada.

Após a carga e inicialização do modelo pelo ambiente de simulação *Hydragyrum*, inicia-se a fase de execução do projeto de simulação. A Figura 3-3 ilustra os casos-de-uso contemplados por um modelo de simulação distribuído. Durante a fase de execução do modelo, quando ocorre a necessidade de interoperabilidade com um modelo remoto, o modelo irá tentar estabelecer conexão com o servidor. Caso positivo, o modelo está pronto para enviar uma mensagem para o servidor. O envio de uma mensagem por um modelo só poderá ocorrer se previamente o modelo localizar o servidor, realizar a autenticação de acesso (baseado em informações previamente cadastradas no banco de dados) e estabelecer conexão. No caso de um modelo estar aguardando uma mensagem remota, este é responsável por verificar se existe mensagem disponível no servidor e, caso exista, obter a mensagem.

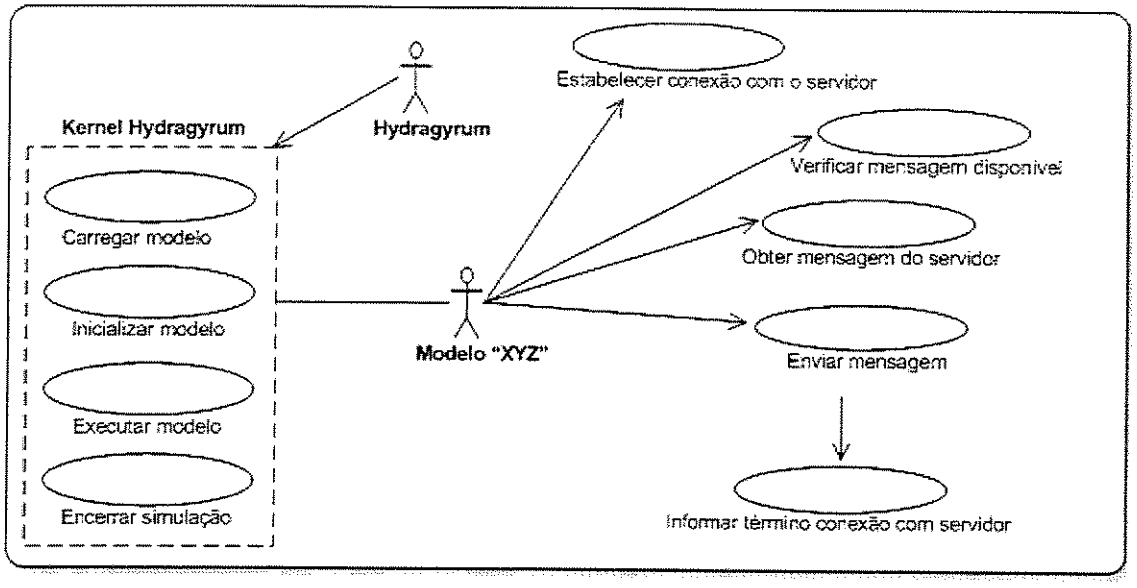

Figura 3-3: Diagrama de Casos-de-Uso para um modelo distribuído
Notação Gráfica UML

Um usuário *Web* só possui a possibilidade de enviar uma mensagem para o servidor. Este, por sua vez, conforme parâmetros recebidos na mensagem, poderá armazená-la na lista de objetos mensagem, podendo ser posteriormente obtida por algum modelo. Para interconectar o envio de mensagem via *Web*, existe uma aplicação (denominada "*Hydragyrum Servlet*") junto ao *Web Container* que é responsável por receber a mensagem do usuário *Web* e encaminhá-la para o servidor. A Figura 3-4 ilustra os casos-de-uso contemplados para o perfil de um usuário *Web* válido para um projeto de simulação distribuído.

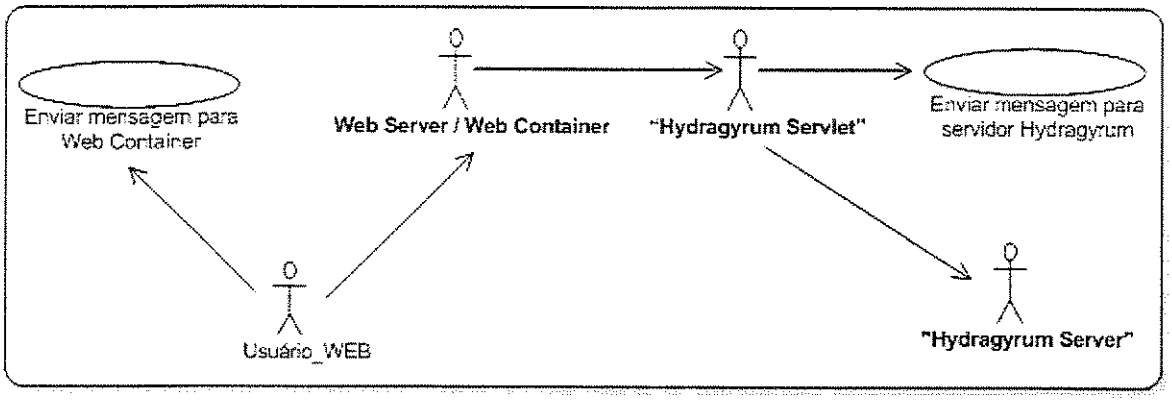

Figura 3-4: Diagrama de Casos-de-Uso para usuário Web
Notação Gráfica UML

3.2.2 O Projeto da Arquitetura

No modelo da arquitetura *cliente/servidor* conhecida como *Three-Tier*, três *Tier's* são concebidas: a *"Client Tier"*, nível de abstração do projeto responsável por apresentar os dados ao usuário, interagindo com o usuário e comunicando com outras camadas da aplicação; a *"Middle Tier"* intermedia a comunicação entre a aplicação rodando no cliente com a aplicação rodando no servidor, cuja função é tornar transparente ao construtor, detalhes físicos que cuidam de serviços específicos, como o gerenciamento de mensagens entre classes, comunicação *cliente/servidor*, controle transacional e acesso a bases de dados; e a *"Data Tier"*, nível de abstração do projeto responsável pelo *RDBMS* e aplicações organizacionais legadas. A *"Data Tier"* reforça segurança, oferece escalabilidade e permite integrar bancos de dados existentes e aplicações com novas funcionalidades. Como principais vantagens desta arquitetura, destacam-se:

- aumento do desempenho através da distribuição: é possível aumentar o desempenho de acordo com a necessidade, através da redistribuição dos

serviços pelos servidores existentes, alocando instâncias de objetos a vários servidores;

- arquitetura multiplataforma e independência do sistema operacional: uma aplicação distribuída pode ser composta de vários "objetos" distribuídos, cada um sendo executado em um diferente ambiente computacional de hardware (plataforma) e software (sistema operacional);
- escalabilidade: é possível aumentar o poder de processamento de acordo com o aumento da demanda, através da inclusão de novos servidores de processamento;
- otimização de custos: devido à possibilidade de distribuição dos processos, não é necessário adquirir computadores centralizados de grande poder de processamento, minimizando os custos com equipamentos.

A arquitetura *Three-Tier cliente/servidor* para o ambiente *Hydragyrum,* conforme ilustrado na Figura 3-5, contém:

- A *"Client Tier".* Nesta *"Tier"*, duas formas de interface de usuário são contempladas:
  - ✓ Através de *Browser Web* para modelagem ou visualização de resultados de uma simulação. A comunicação com a *"Middle Tier"* faz-se utilizando o protocolo *HTTP* sobre *TCP/IP*.
  - ✓ Através de linha de comando pelo interpretador de comandos *"SIMSCL"* do ambiente de simulação *Hydragyrum*. A comunicação com a *"Middle Tier"* faz-se com a utilização do componente de *software "distrib"* responsável pela distribuição dos dados através do protocolo *IIOP* sobre *TCP/IP*. O descritores (*".SCL"*) utilizados para referenciar os modelos a serem carregados para a memória do ambiente de simulação não sofrem

nenhuma modificação, herdando as características originais do *Hydragyrum*. Os parâmetros, tais como endereço *IP*, porta de comunicação, usuário e senha válidos, utilizados para conectar-se ao servidor *Hydragyrum*, podem ser contemplados nas próprias sentenças no arquivo descritor dos modelos, ao invés de armazenar estas informações estaticamente em código dentro do modelo.

- A *"Middle Tier"*. Através da entidade denominada *"Hydragyrum Server"*, esta *"Tier"* é responsável por recepcionar as mensagens e distribuí-las adequadamente para os modelos. As mensagens podem ser obtidas pelos destinatários quando solicitadas. O servidor *Web Apache* é a referência oficial para servir páginas hipertexto (*HTML*). As *Servlets* são objetos *Java* executados no servidor em resposta a uma solicitação do navegador. As *Servlets* podem chamar uma *JSP (Java Server Pages)* para produzir a saída ou gerar *HTML* ou *XML* diretamente para ser mostrado no *Browser*. O servidor de *Container Tomcat* é um *"servlet container"* com um ambiente *JSP*. *Tomcat* é a referência oficial para a implementação das tecnologias *Java Servlet* e *Java Server Pages*. Um *"servlet container"* é um *"runtime shell"* que gerencia e invoca *servlets* de interesse dos usuários. O *AJP12* e *AJP13* são os protocolos utilizados pelo módulo de redirecionamento no *Apache* para enviar mensagens para o *Web Container Tomcat*.

- A *"Data Tier"*. Esta *Tier* é responsável pelo armazenamento dos dados. A comunicação com esta *"Tier"* faz-se através de linguagem *SQL* sob protocolo *JDBC* sobre *TCP/IP*. O *JDBC* é a especificação em *Java* que define a *API* (um conjunto de classes totalmente escritas em *Java*) que permite aos programas acessarem bancos de dados compatíveis com esse padrão, utilizando a adição de comandos *ANSI SQL/92 (Structured Query Language)* em uma aplicação *Java*. O objetivo do *driver JDBC* é ser uma interface independente de qualquer

sistema de gerenciamento de banco de dados, permitir um acesso genérico a banco de dados através de *SQL* e ter uma interface uniforme para diferentes fontes de dados. O *MySQL* é o gerenciador de banco de dados relacional, compacto, portável para múltiplas plataformas de sistema operacional.

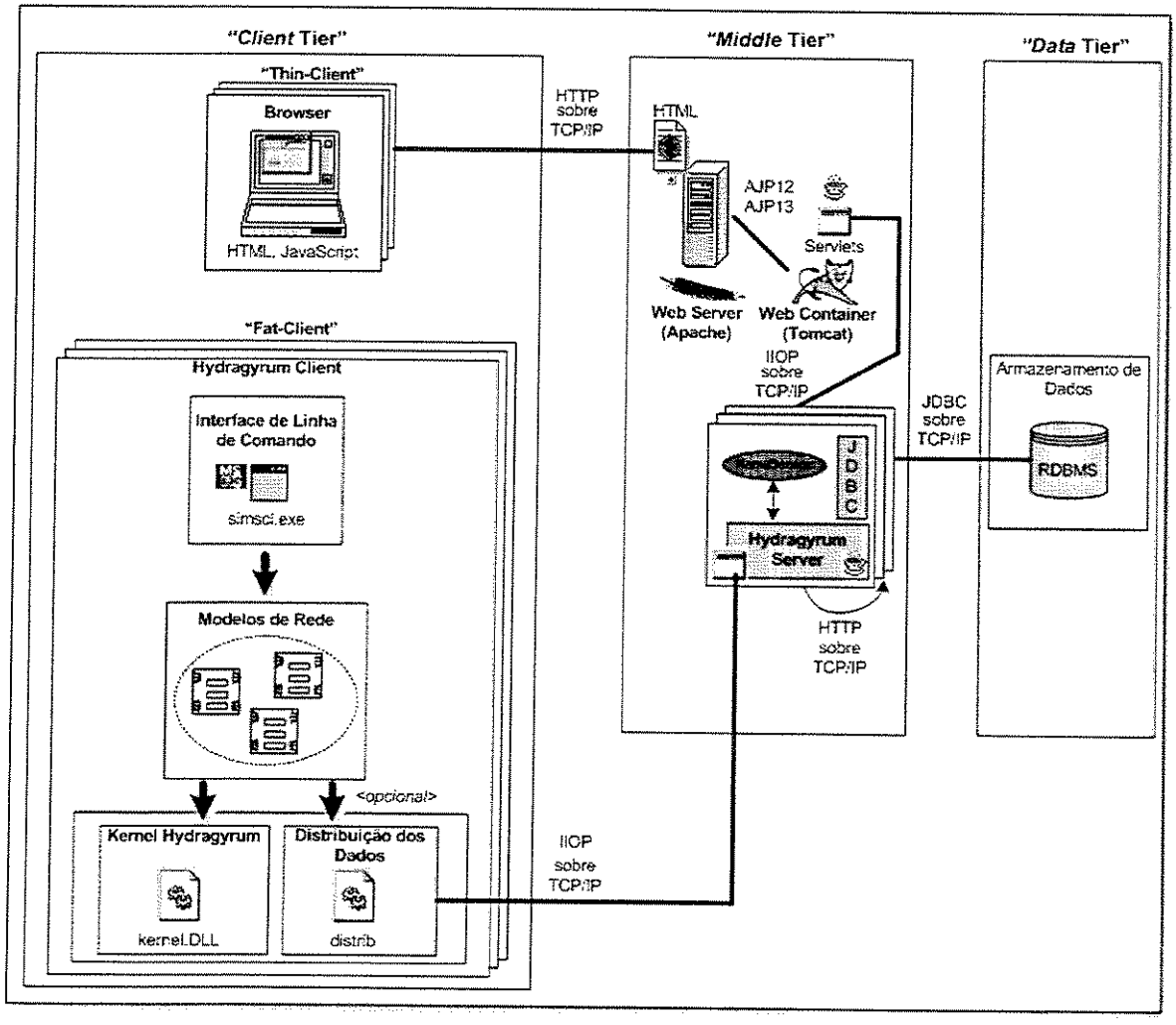

Figura 3-5: Arquitetura three-tier para o ambiente Hydragyrum Cliente/Servidor

Como observa-se na Figura 3-6, "n" modelos poderão conectar-se ao *ORB*, após a devida autenticação pela aplicação "Hydragyrum Server" (visualizada na "Middle Tier" da Figura 3-5) que realiza o gerenciamento dos modelos no contexto distribuído. A Figura 3-7 ilustra a possibilidade de diversos ambientes de simulação poderem realizar a interoperabilidade com o ambiente *Hydragyrum* através das interfaces *IDL* definidas no trabalho. Na Figura 3-8, o diagrama de camadas baseado no modelo *OSI* para o ambiente *Hydragyrum cliente/servidor* é apresentado. Como pode ser observado na Figura 3-8, o uso do padrão *CORBA* para o desenvolvimento da arquitetura cliente/servidor possibilita abstração das camadas 5 e 6, ficando a cargo do desenvolvedor somente a responsabilidade de elaboração das aplicações que encontram-se na camada 7.

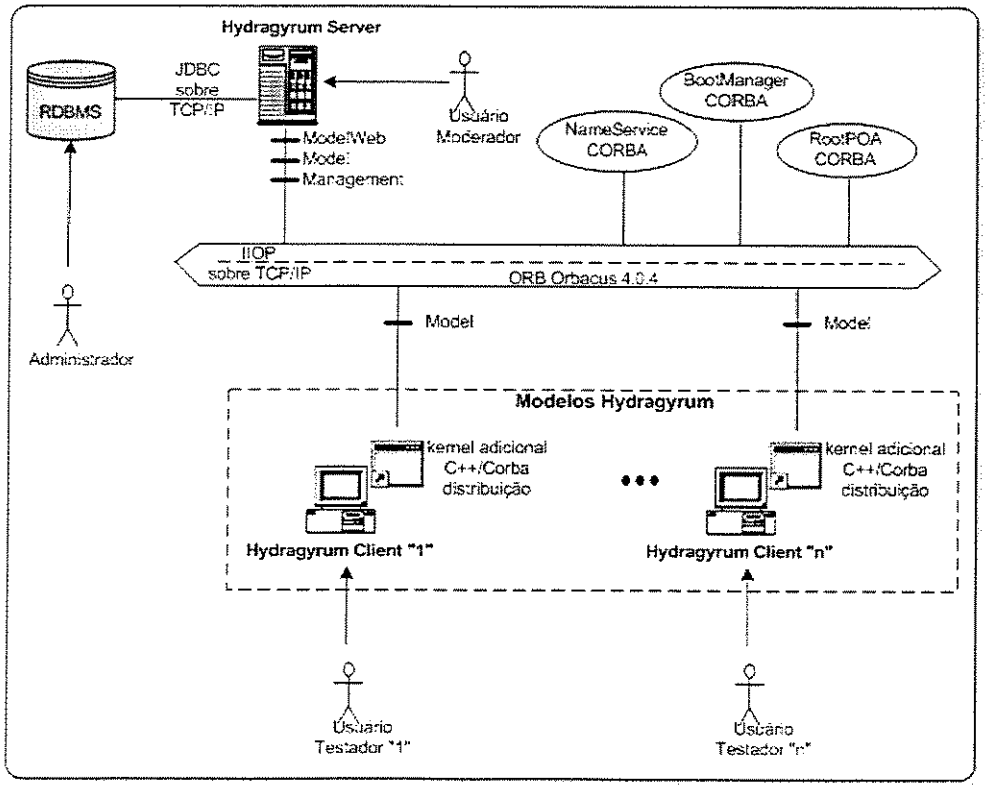

Figura 3-6: Diagrama de Distribuição baseado no Modelo CORBA

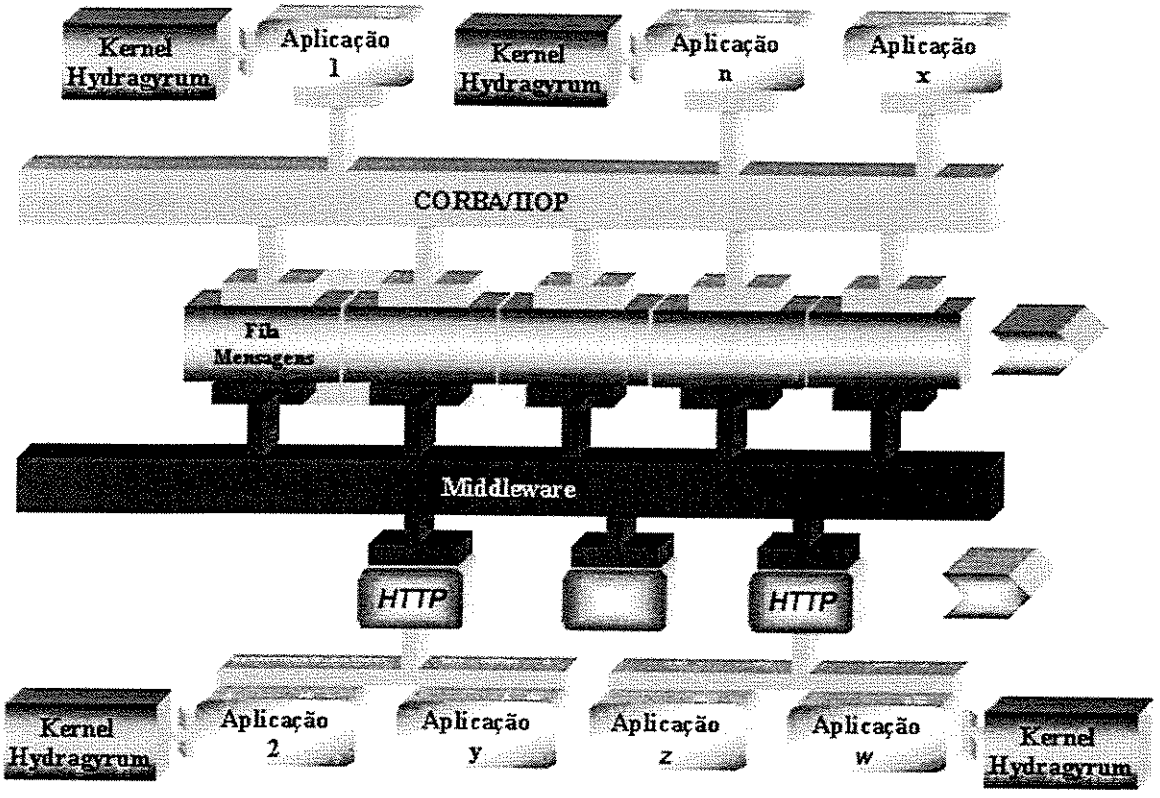

Figura 3-7: Diagrama da Solução sob o Ponto de Vista de *Framework*

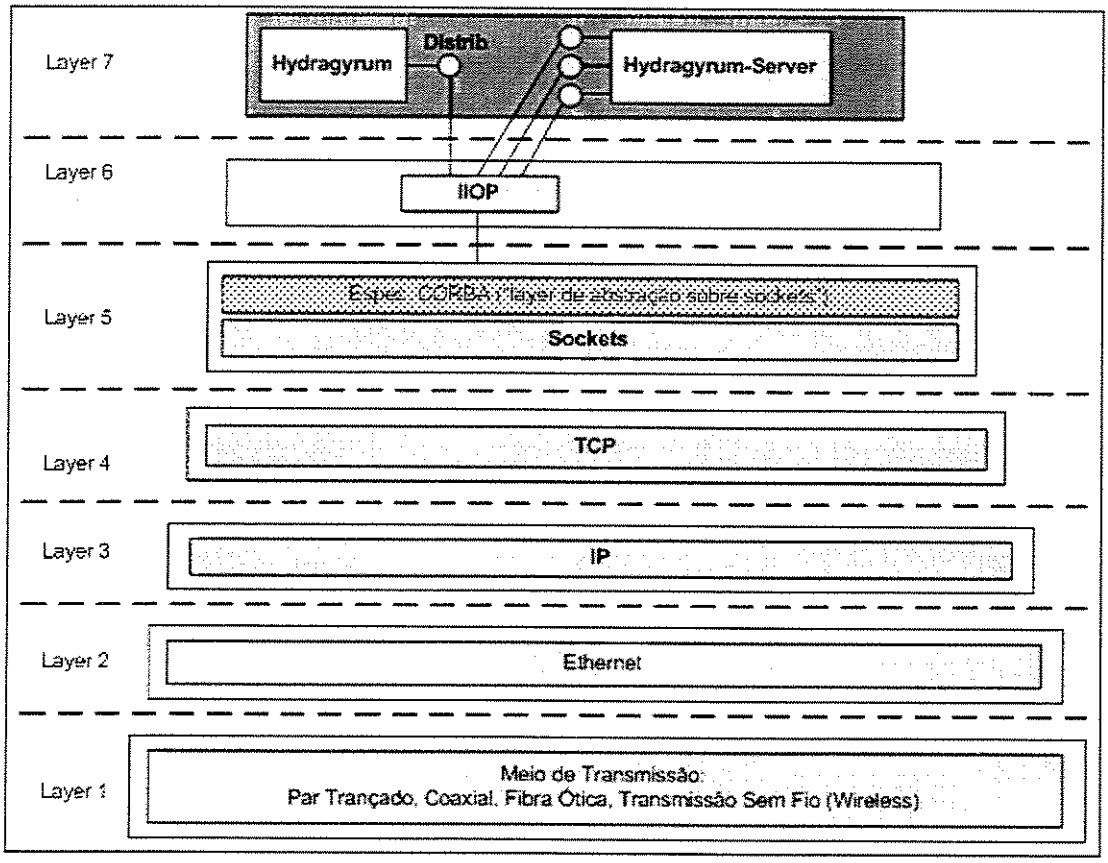

Figura 3-8: Diagrama de Camadas baseado no Modelo OSI

Em qualquer uma das fases da simulação de um modelo no Hydragyrum (Figura 2-2) uma mensagem para o servidor *Hydragyrum* (responsável pela gerência dos eventos distribuídos) pode ser enviada. As *Interfaces IDL* definem as operações que são disponibilizadas pela implementação do objeto (aplicação que será inicializada no servidor *"Hydragyrum Server"*). A seguir, as *Interfaces IDL* são apresentadas para a arquitetura *cliente/servidor* para o ambiente de simulação *Hydragyrum.*

Interface ***Management***

```
// Operação utilizada para realizar a conexão junto ao servidor Hydragyrum
long connect(in string login, in string passwd, in string projectName, in Object ior);

// Operação utilizada para realizar a desconexão junto ao servidor Hydragyrum
void disconnect(in long idModel);

// Operação utilizada para realizar o envio de mensagem para o servidor Hydragyrum
void sendMessage(in string originaryBlockName, in string originaryLayerName,
                 in string destinyBlockName, in string destinyLayerName,
                 in string type, in any message);

// Operação utilizada para realizar o encerramento da execução do servidor Hydragyrum
void shutdownServer(in string login, in string passwd);
```

Interface ***ModelWeb***

```
// Operação utilizada para realizar o envio de mensagem para o servidor Hydragyrum
void sendMessage(in string projectName, in string originaryBlockName,
                 in string originaryLayerName, in string destinyBlockName,
                 in string destinyLayerName, in string type, in any message);
```

Interface ***Model***

```
// Operação utilizada para realizar o envio de mensagem para o servidor Hydragyrum
long CreateLayerControlEvent_D(in double time, in string type,
                               in string destinyBlock,
                               in string destinyLayerName,
                               in any _SMessage);

// Operação utilizada para verificar a existência de uma determinada mensagem no
// servidor Hydragyrum
boolean verifyMessage(in string type);

// Operação utilizada para obter uma mensagem do servidor Hydragyrum
any getMessage(in string type);
```

3.3 Cenários de Simulação

A seguir, é apresentada uma descrição de alguns cenários de simulação distribuídos que podem ser contemplados na execução de projetos. Para efeito ilustrativo, visando simplificar o entendimento na modelagem, os cenários foram elaborados com apenas dois projetos, onde cada projeto apresenta apenas um modelo. A escolha do cenário de simulação mais apropriado poderá ser feita em função de critérios tais como facilidade de desenvolvimento, recursos disponíveis (infra-estrutura), necessidade de otimização de tráfego/desempenho e complexidade das simulações.

3.3.1 Somente um Servidor

Neste caso usa-se uma arquitetura de simulação com apenas um servidor para todos os projetos distribuídos, conforme ilustrado na Figura 3-9. A grande vantagem é sua facilidade de desenvolvimento e execução. Contudo, esta arquitetura de simulação não é apropriada para simulações complexas que necessitam de grande tráfego de informações entre modelos distribuídos.

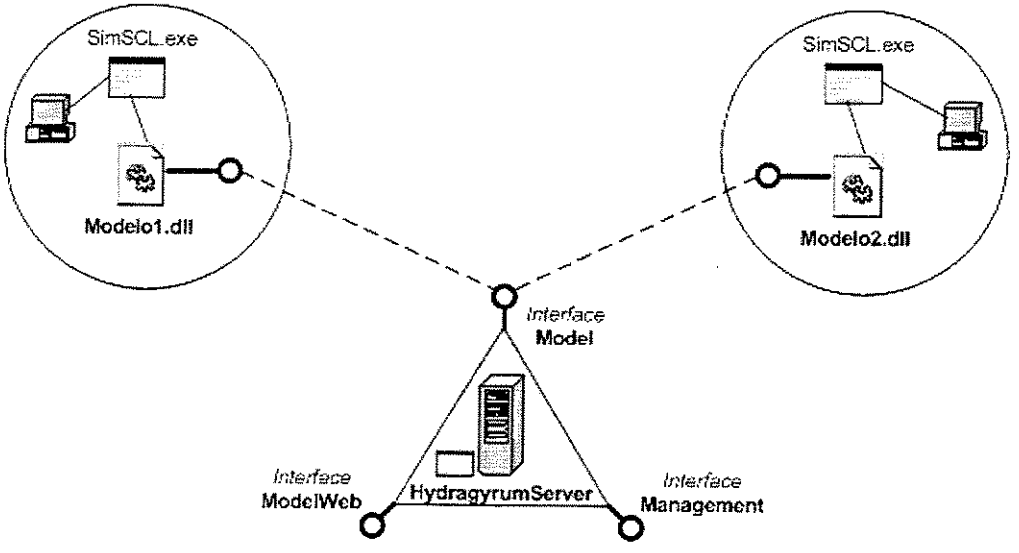

Figura 3-9: Cenário de execução dos projetos com somente um servidor

A experimentação desta arquitetura de simulação distribuída pode ser concebida pela simulação em computador local (onde todos os elementos da simulação encontram-se no mesmo computador) conforme pode ser visto na Figura 3-10.

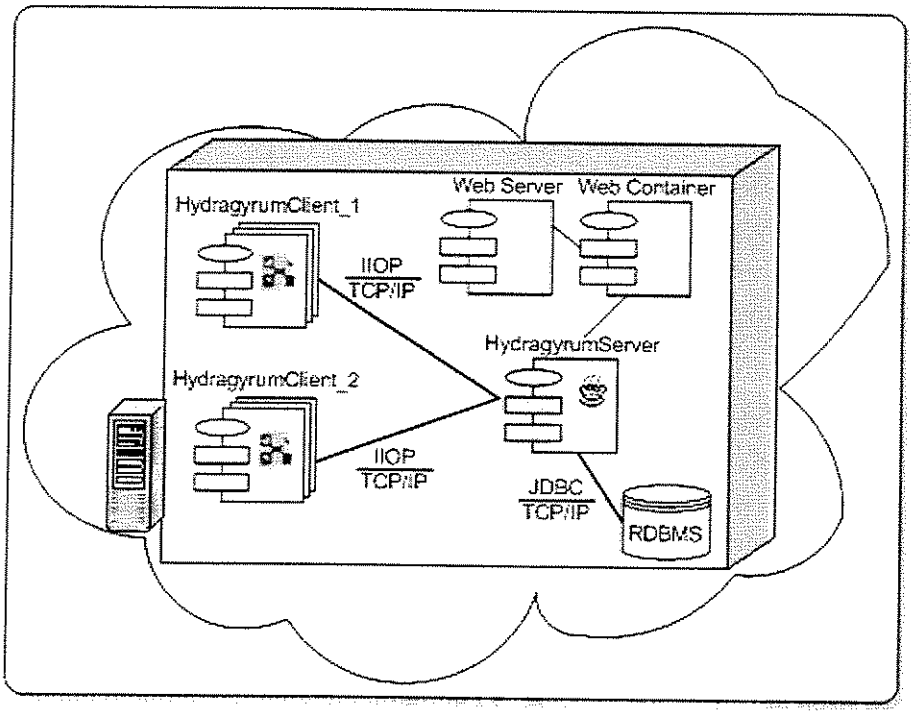

Figura 3-10: Cenário de simulação em computador local
Diagrama de Distribuição - Notação Gráfica UML

Também é possível contemplar a experimentação desta simulação em computadores distintos (onde os elementos da simulação encontram-se em diferentes computadores) distribuídos em uma rede local, via protocolo *IIOP*, conforme ilustrado na Figura 3-11.

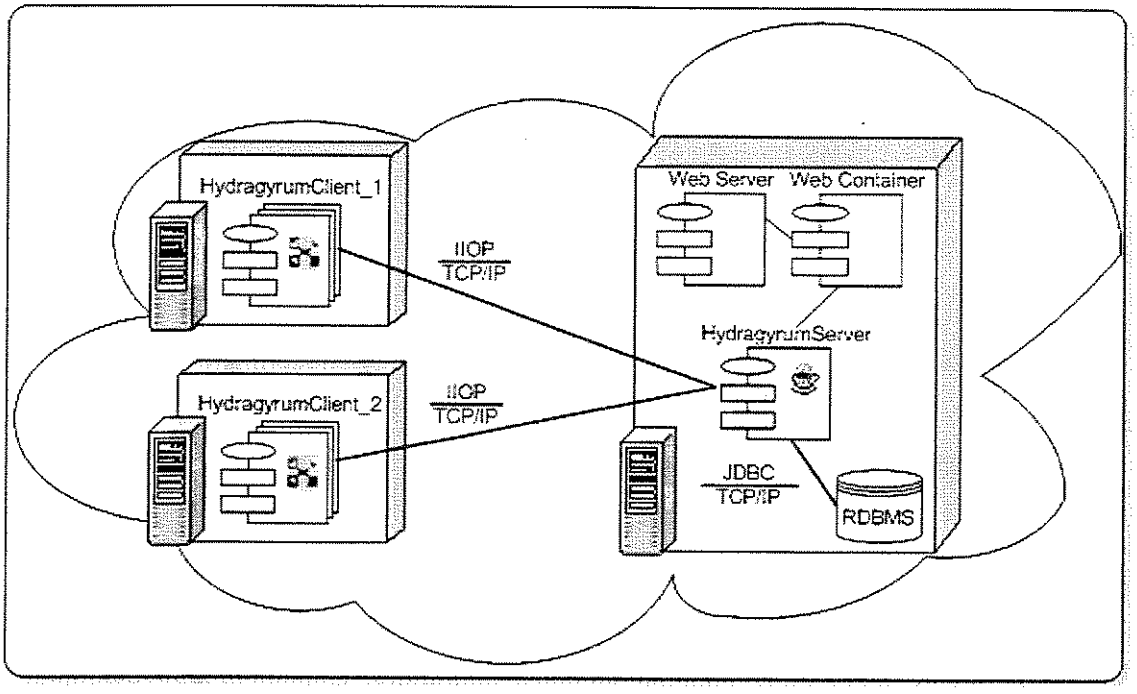

Figura 3-11: Cenário de simulação em computadores distintos
Diagrama de Distribuição - Notação Gráfica UML

3.3.2 Servidor Local para Cada Projeto

Conforme ilustrado na Figura 3-12, um servidor local para cada projeto possibilita otimizar a quantidade de tráfego de dados na rede de comunicação em virtude de ser delegado ao servidor *Hydragyrum* notificar a outros servidores remotos quando resultados de eventos estiverem disponíveis para outros projetos de simulação.

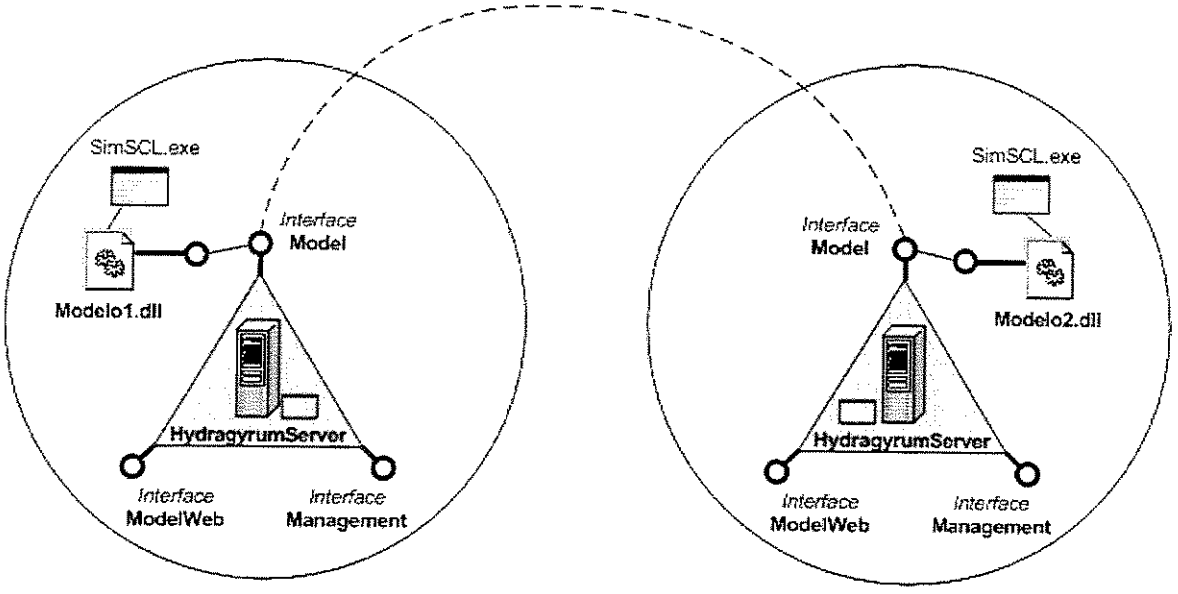

Figura 3-12: Cenário de simulação com servidor local para cada projeto

3.3.3 Múltiplos Servidores Descentralizados

É uma arquitetura de simulação com a possibilidade de múltiplos servidores descentralizados a fim de maximizar o desempenho distribuído dos modelos. Através da inclusão de um servidor *Hydragyrum* adicional, atuando como entidade de controle, é possível reduzir o tráfego de dados de solicitação de mensagens distribuídas, conforme apresentado na Figura 3-13. Basicamente, a idéia consiste em encaminhar as mensagens armazenadas em cada servidor para a entidade de controle, que por sua vez se responsabiliza por entregar a mensagem para outro servidor.

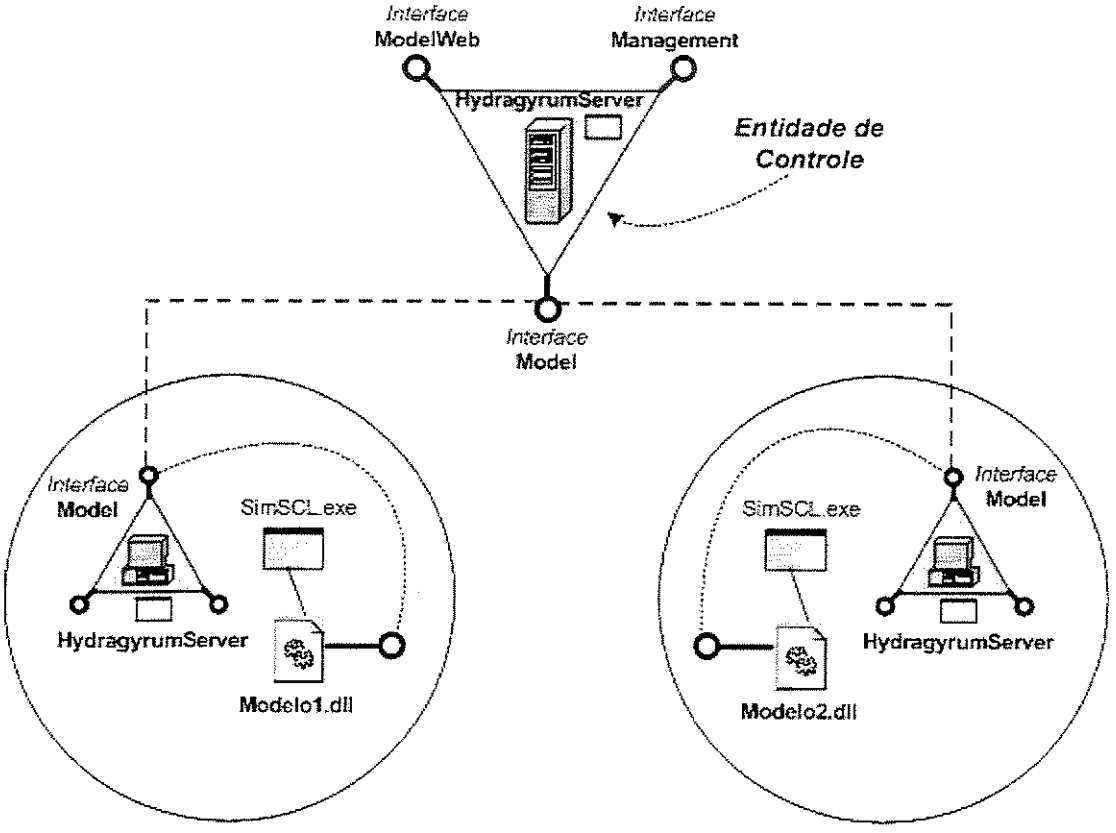

Figura 3-13: Cenário de simulação com múltiplos servidores descentralizados

3.4 Elementos de Software Envolvidos na Simulação Distribuída

A Figura 3-14 mostra os principais elementos de *software* envolvidos no desenvolvimento de um projeto de simulação distribuído para o ambiente *Hydragyrum*. O aplicativo "SimSCL.EXE" usa a biblioteca de vínculo dinâmica (*Kernel.dll*) do *Hydragyrum* e realiza a carga dos modelos a partir do descritor *SCL* (Linguagem de Comandos do Ambiente de Simulação *Hydragyrum*), para posterior execução.

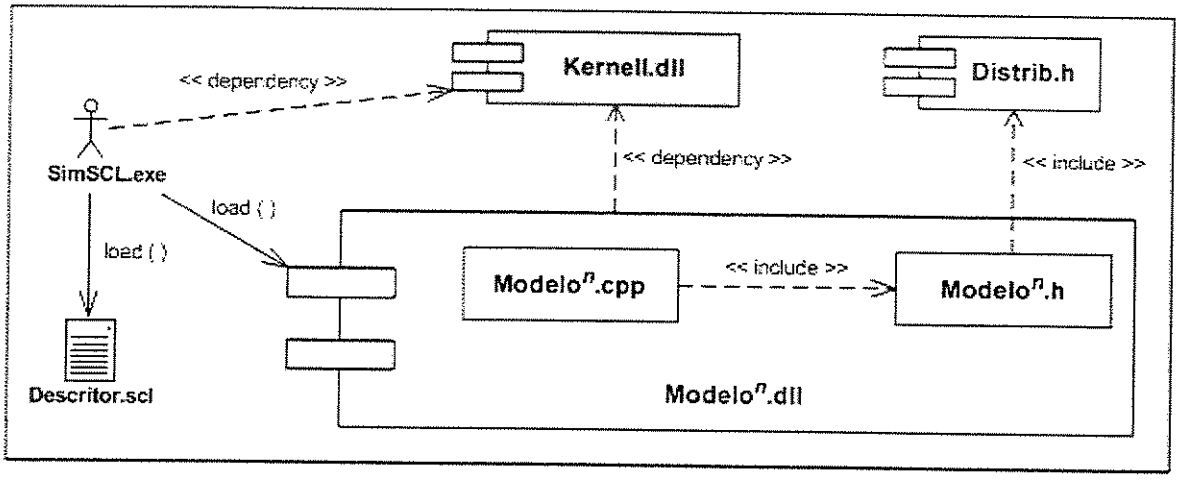

Figura 3-14: Elementos de Software em um Projeto Distribuído

3.5 Conclusão

Neste capítulo são apresentados os requisitos que guiaram o desenvolvimento da arquitetura *cliente/servidor* proposta para o ambiente *Hydragyrum*. Baseado nestes requisitos foi elaborado o projeto da arquitetura *cliente/servidor* a ser implementada. O capítulo a seguir apresenta os resultados provenientes do desenvolvimento do trabalho.

(Página Intencionalmente em Branco)

Capítulo 4

Contribuições e Resultados

4.1 Contribuições

Para o desenvolvimento da aplicação distribuída utilizando a tecnologia *CORBA*, inicialmente é necessário se ter a disposição o *software* que implementa esta arquitetura. Este trabalho utiliza o *CORBA ORB ORBacus* versão 4.0.4 [7], *software* que implementa a especificação *CORBA* versão 2.3.1 integralmente. O código-fonte do *ORB ORBacus* sofreu adaptações de configuração para o ambiente operacional *Windows* com as definições *multithread* compatíveis com o ambiente *Hydragyrum* sendo inteiramente recompiladas para as necessidades do trabalho. As adaptações de configuração impactaram pequenas alterações nos *scripts* de compilação do *ORB ORBacus* e o maior esforço concentrou-se em determinar quais modificações eram necesssárias para o uso do *ORB* com o ambiente *Hydragyrum*. Para determinar as mudanças necesssárias no *ORB ORBacus* utilizou-se como base a sua documentação e foi necessário um aprofundamento do uso das *threads* no ambiente operacional *Windows*.

A partir dos requisitos da aplicação descritos em *Casos-de-Uso UML*, o projeto da arquitetura de *Tiers* foi realizado. Neste projeto foram definidas as funcionalidades que cada *Tier* teria que desempenhar. Uma vez que os *Casos-de-Uso* do sistema a ser construído foram mapeados, realizou-se a identificação dos objetos da aplicação e a definição das interfaces de comunicação em *CORBA IDL*. Com o projeto elaborado, iniciou-se a implementação das interfaces em linguagem *C++* e *Java*. O componente

"distrib" (Figura 4-1), construido em C++, implementa as operações descritas na Interface *IDL ("Model")* para um modelo *Hydragyrum.* Ele tem a finalidade de fornecer um mecanismo para a troca de dados entre projetos de simulação distribuídos no ambiente *Hydragyrum*. Para a compilação da Interface *IDL ("Model")*, o compilador *CORBA IDL* para *C++* do *ORB ORBACUS* foi utilizado, obtendo-se os *stubs* e os *skeletons* dos objetos que descrevem a Interface.

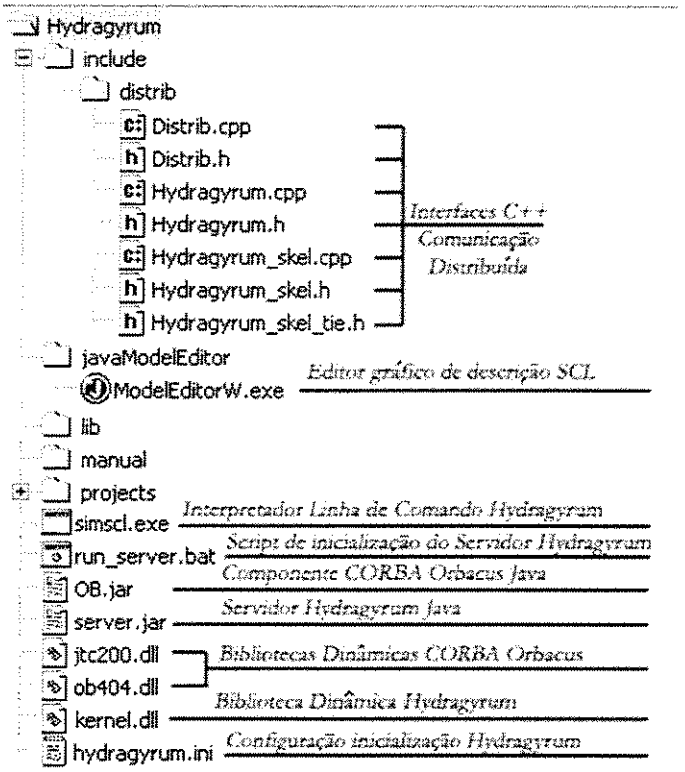

Figura 4-1: Artefatos do Ambiente Hydragyrum Cliente/Servidor

A Figura 4-2 ilustra o relacionamento dos componentes com as interfaces *distribuídas* de comunicação. O componente denominado "*HydragyrumServer*" é o próprio servidor *Hydragyrum*. O *servidor* foi desenvolvido em *Java* e possibilita que clientes *Hydragyrum* previamente cadastrados conectem-se junto a este *servidor* e realizem a troca de dados

entre projetos distribuídos. O *servidor* implementa as três Interfaces *OMG IDL* (*Model*, *Management*, *ModelWeb*) geradas a partir do compilador *CORBA IDL* para *Java* do *ORB ORBACUS*. Para auxiliar a experimentação dos cenários, um gerenciador de banco de dados pode ser utilizado para compartilhar informações e armazenar o histórico dos resultados realizados nos projetos para acesso posterior. Uma proposta de modelagem para a base de dados é apresentada na Figura 4-3.

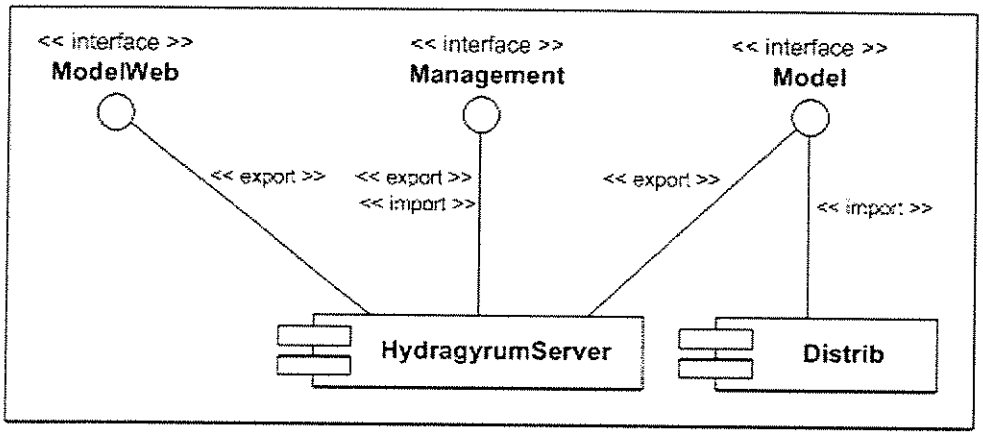

Figura 4-2: Interfaces de Comunicação x Componentes

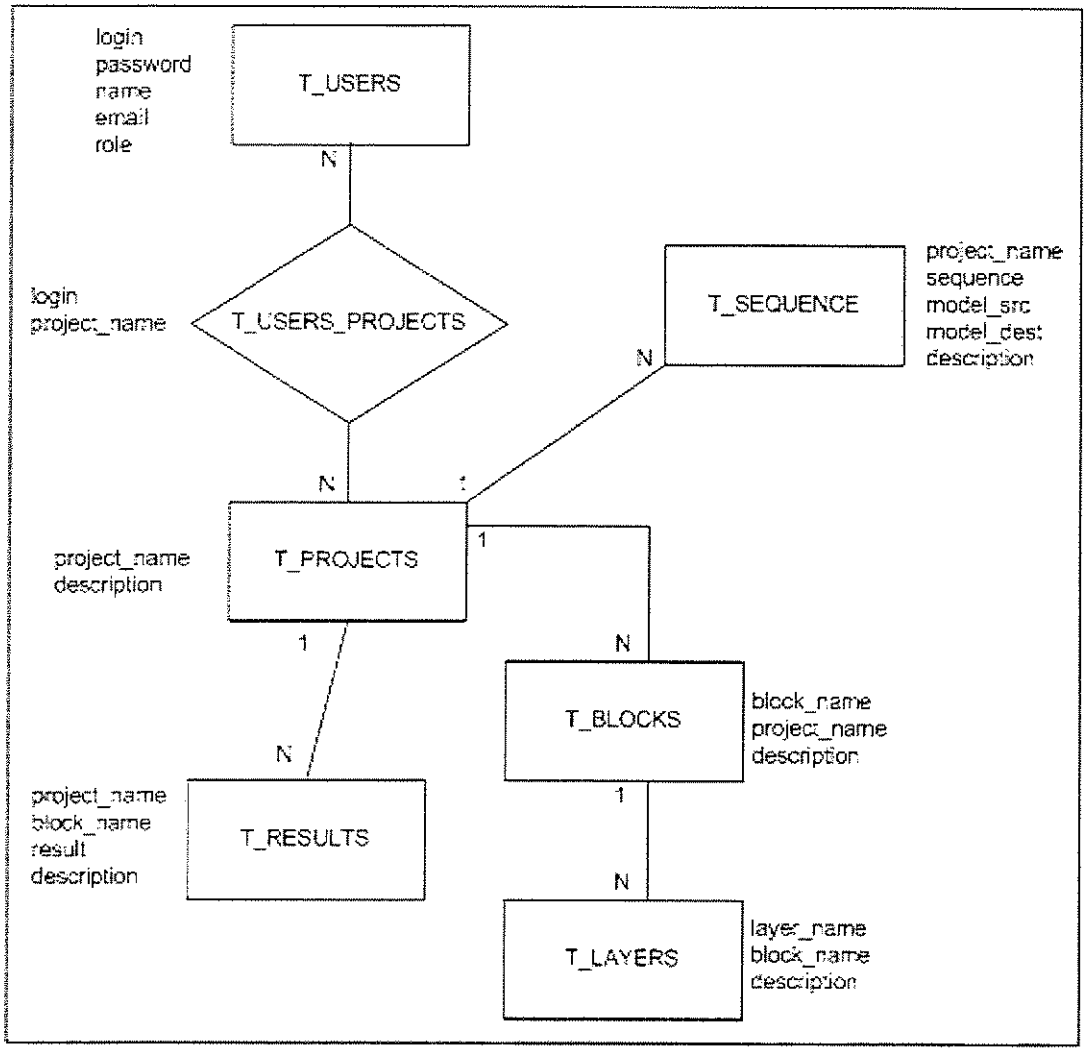

Figura 4-3: Diagrama MER – Modelo Entidade Relacionamento

A seguir é apresentado uma descrição das tabelas do banco de dados:

Nome da Tabela:	T_USERS	
Campos:	login	VARCHAR(8) NOT NULL
	password	VARCHAR(20) NOT NULL
	name	VARCHAR(40) NOT NULL
	email	VARCHAR(40)
	role	TINYINT NOT NULL
OBSERVAÇÕES:	Esta tabela armazena as informações dos usuários. O campo "login" é a identificação primária do usuário utilizada para autenticação na aplicação "Hydragyrum Server". Portanto é chave primária na base de dados. O campo "role" indica o papel que um usuário assume nos projetos, podendo ter o privilégio de acesso representado pelo valor numérico "0" para usuário sem privilégios (conta bloqueada), "1" para usuário moderador, "2" para usuário desenvolvedor, "3" para usuário testador via aplicação, "4" para usuário Testador *Web*, "5" para usuário Cliente final dos resultados e "9" para usuário Visitante.	

Nome da Tabela:	T_USERS_PROJECTS	
Campos:	login	VARCHAR(8) NOT NULL
	project_name	VARCHAR(25) NOT NULL
Observações:	Esta tabela armazena o relacionamento da tabela Usuários com a tabela Projetos. O campo "login" é a identificação do usuário. O campo "project_name" é a identificação do projeto que o usuário está cadastrado.	

Nome da Tabela:	T_PROJECTS	
Campos:	project_name	VARCHAR(25) NOT NULL
	description	VARCHAR(100)
OBSERVAÇÕES:	Esta tabela descreve os projetos cadastrados. O campo "project_name" é a identificação primária do projeto, sendo chave primária na base de dados. O campo "description" é utilizado para uma breve descrição sobre o funcionamento do projeto.	

Nome da Tabela:	T_SEQUENCE	
Campos:	project_name	VARCHAR(25) NOT NULL
	sequence	TINYINT NOT NULL
	model_src	VARCHAR(25) NOT NULL
	model_dest	VARCHAR(25) NOT NULL
	description	VARCHAR(100)
Observações:	Esta tabela descreve a seqüência de execução dos modelos para os projetos cadastrados. O campo "project_name" é a identificação primária do projeto, sendo chave primária na base de dados. O campo "sequence" define a seqüência de execução dos eventos distribuídos. O campo "model_src" é o nome do modelo onde o evento é originado. O campo "model_dest" é o nome do modelo onde o evento é recebido. O campo "description" apresenta uma breve explicação sobre a seqüência de execução do modelo.	

Nome da Tabela:	T_RESULTS	
CAMPOS:	project_name	VARCHAR(25) NOT NULL
	block_name	VARCHAR(25) NOT NULL
	result	TEXT
	description	VARCHAR(100)
Observações:	Os resultados das simulações em função do projeto e modelo associado podem ser armazenados nesta tabela. O campo "result" contém o resultado da simulação de um determinado modelo.	

Nome da Tabela:	T_BLOCKS	
CAMPOS:	block_name	VARCHAR(25) NOT NULL
	project_name	VARCHAR(25) NOT NULL
	description	VARCHAR(100)
Observações:	Esta tabela descreve os modelos para os projetos cadastrados. O campo "block_name" é a identificação primária do modelo, sendo chave primária na base de dados.	

Nome da Tabela:	T_LAYERS	
Campos:	layer_name	VARCHAR(25) NOT NULL
	block_name	VARCHAR(25) NOT NULL
	description	VARCHAR(100)
Observações:	Esta tabela descreve a seqüência de execução dos modelos para os projetos cadastrados. O campo "layer_name" é a identificação primária da camada, sendo chave primária na base de dados.	

Para auxiliar a construção dos descritores da *SCL* (Linguagem de Comandos do Ambiente de Simulação) dos modelos *Hydragyrum*, foi desenvolvido um editor básico (*Figura 4-4*) em linguagem de programação *Java*, para facilitar a visualização da edição das sentenças dos modelos, com as funcionalidades: "Novo" modelo, "Abrir" modelo, "Fechar" modelo, "Salvar" modelo e "Salvar Como".

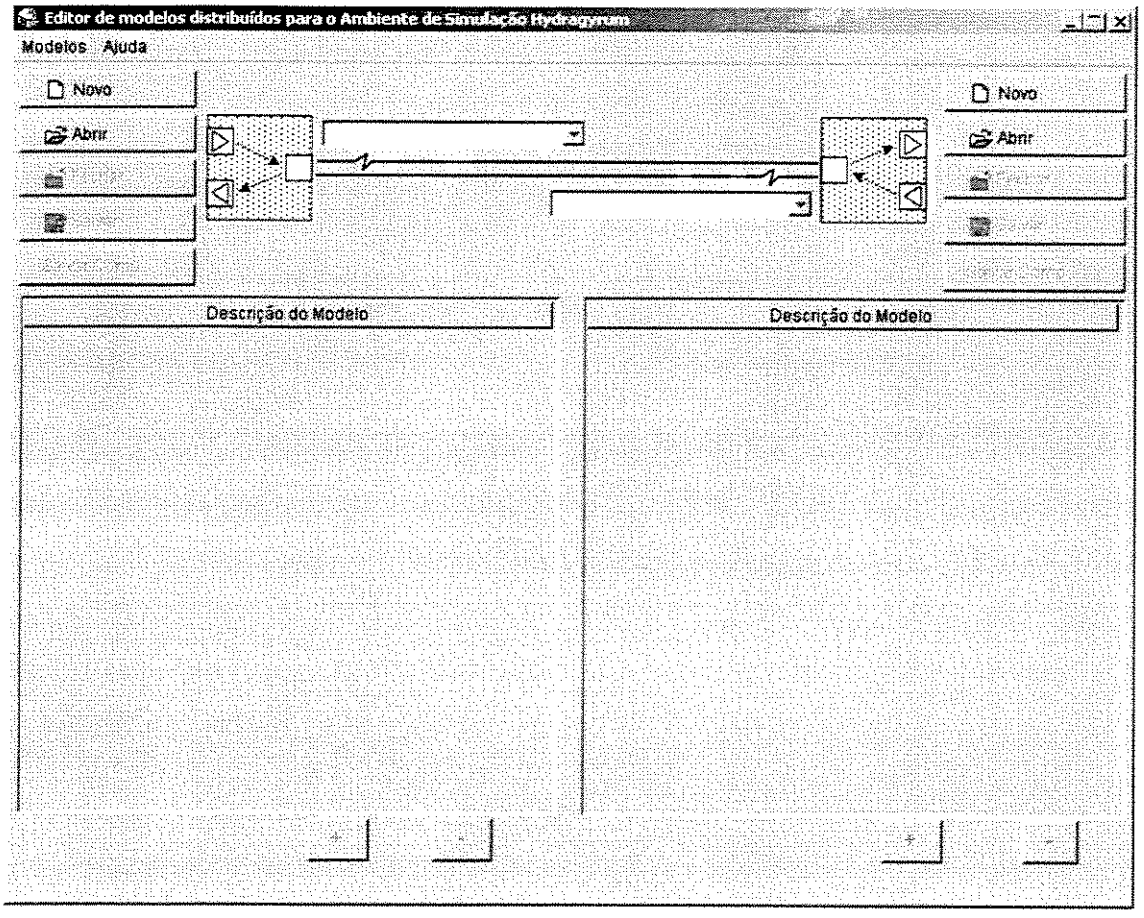

Figura 4-4: Interface gráfica da ferramenta para a edição dos descritores da SCL

4.2 Simulação de Projeto – Exemplo 1

4.2.1 Descrição do Requisitos do Projeto a ser Simulado

Para demonstrar o desenvolvimento de um projeto no ambiente *Hydragyrum Cliente/Servidor*, um exemplo simplificado para efeito didático com intuito de exercitar um caso de teste no ambiente de simulação *Hydragyrum cliente/servidor* é apresentado a seguir. O projeto consiste em realizar o cálculo de uma equação matemática (*Figura 4-5*) e apresentar o resultado.

$$R = \frac{\sum_{K=1}^{10000} \left(\sum_{n=1}^{500000} a_k * 0.006415 * \frac{(k)}{n} \right)}{10000}$$

Figura 4-5: Equação matemática para o projeto exemplo a ser simulado

4.2.2 Descrição do Desenvolvimento do Projeto

O projeto será implementado usando a estratégia de processamento seqüêncial e a estratégia de processamento distribuído. No processamento seqüêncial apenas um computador será utilizado. No processamento distribuído serão utilizados dois computadores a fim de diminuir o tempo no processamento para alcançar o resultado. Ao final, será apresentado o resultado comparativo dos tempos de execução para cada uma das duas estratégias.

No processamento distribuído, a fórmula matemática será particionada em duas partes (R1 e R2), conforme ilustrado na Figura 4-6, sendo que cada parte da fórmula está sendo executada por um projeto. O projeto denominado "Projeto 1" realiza o cálculo da

equação matemática *R1* e envia o resultado para o servidor *Hydragyrum*. O projeto denominado "Projeto 2" realiza o cálculo da equação matemática *R2* e ao final solicita o resultado R1 para o servidor *Hydragyrum*. Assim que o o servidor *Hydragyrum* possuir o resultado R1, ele entrega ao "Projeto 2" que consolida os resultados e apresenta o resultado final (R_{12}). A Figura 4-7 ilustra o *Diagrama de Casos-de-Uso* da *UML* com as funcionalidades a serem implementadas e a Figura 4-8 ilustra a seqüência de execução da troca de mensagens entre projetos. A Figura 4-9 ilustra a interconexão entre dois projetos, onde a denominação "Projeto 3" é uma abstração concebida para representar a interoperabilidade entre o "Projeto 1" e o "Projeto 2".

$$R_1 = \sum_{K=1}^{5000} \left(\sum_{n=1}^{500000} a_k * 0.006415 * \frac{(k)}{n} \right)$$

$$R_2 = \sum_{K=5001}^{10000} \left(\sum_{n=1}^{500000} a_k * 0.006415 * \frac{(k)}{n} \right)$$

$$R_{12} = \frac{R_1 + R_2}{10000}$$

Figura 4-6: Particionamento da equação matemática para o processamento distribuído em dois computadores

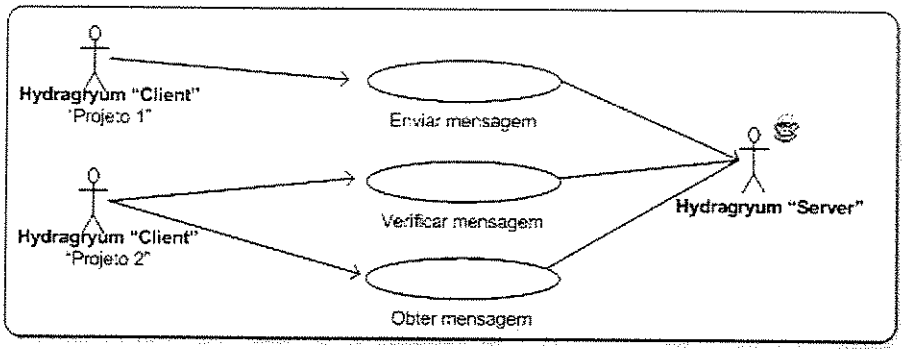

Figura 4-7: Exemplo de Cenário Trivial de Simulação Distribuída

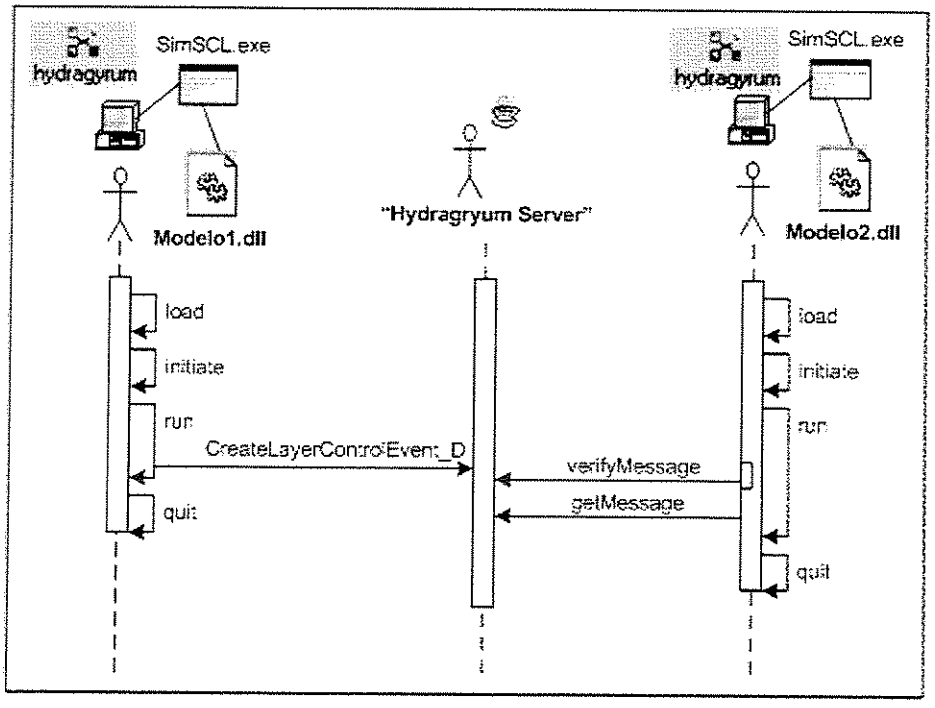

Figura 4-8: Diagrama de Seqüências UML para o Exemplo de Cenário Trivial de Simulação Distribuída

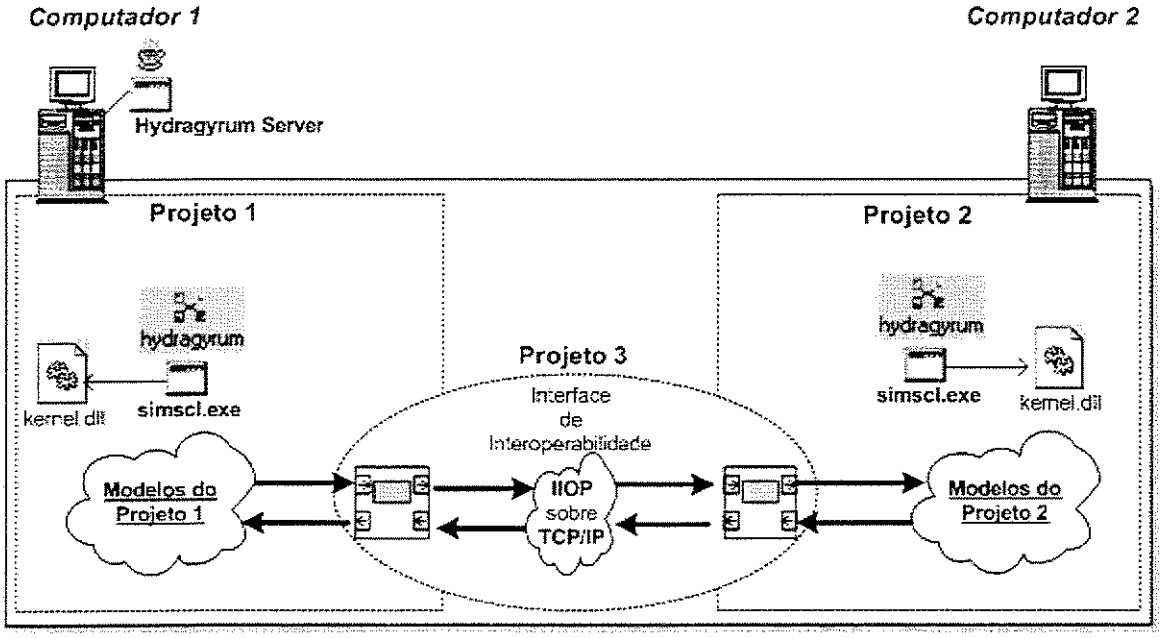

Figura 4-9: Interconexão entre dois projetos em um sistema distribuído

4.2.3 Resultados da Simulação

Para a realização da simulação foi usada a seguinte configuração dos computadores: CPU *PIII-733 MHz, 512 Mb RAM, Sistema Operacional Microsoft Windows™ 2000 Professional*. Com o propósito de constatar que modelos de complexidade média podem obter ganhos no tempo de processamento, realizou-se a simulação através de processamento seqüêncial e de processamento distribuído. Os resultados numéricos obtidos referentes ao tempo de execução das simulações estão descritos a seguir.

- Simulação através de processamento seqüencial (Figura 4-10):

Valor da variável "k"	Tempo de execução (ms)
1	130
10	380
100	2.874
1000	33.147
10000	329.303

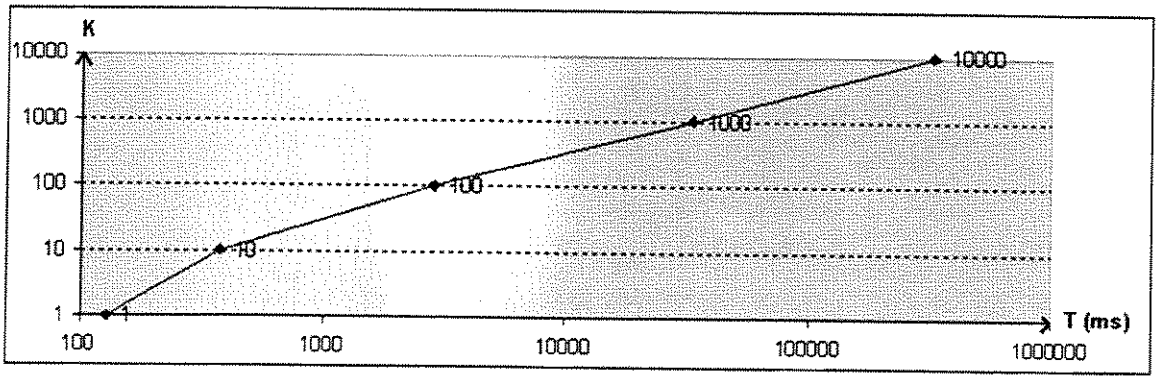

Figura 4-10: Gráfico Tempo x Iterações K para a Simulação Seqüêncial

- Simulação através de processamento distribuído:

  Cálculo de R1 - Computador 1 – K = 5000 – 164.935 ms

  Cálculo de R2 - Computador 2 – K = 5000 – 164.898 ms

  Cálculo de R3 ≈ 30 ms

  Tempo gasto na rede para envio da informação e processamento ≈ 1.870 ms

  Tempo total gasto para chegar ao resultado ≈ 166.835 ms

Baseado nos requisitos propostos para este projeto, evidenciou-se uma significativa redução no tempo total gasto para alcançar o resultado utilizando a simulação através

de processamento distribuído com dois computadores quando comparado à simulação através do processamento seqüencial onde apenas um computador foi utilizado, conforme ilustrado na Figura 4-11.

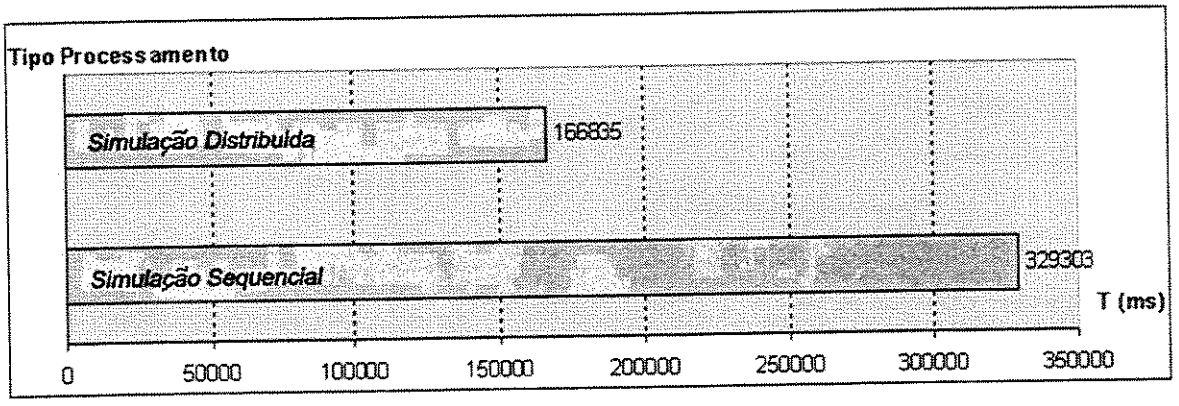

Figura 4-11: Gráfico comparativo da Simulação Sequencial x Simulação Distribuída

4.3 Simulação de Projeto – Exemplo 2

4.3.1 Descrição do Requisitos do Projeto a ser Simulado

O projeto a ser simulado no ambiente *Hydragyrum Cliente/Servidor*, consiste na interoperabilidade de duas redes de computadores (Figura 4-12) com alguns modelos de geração de tráfego em cada rede. A primeira rede simulada ("Projeto 1") e a segunda rede simulada ("Projeto 2") apresentam a estrutura física semelhante. O diferencial entre as duas redes são os pâmetros de entrada nos modelos de geração de tráfego. Estes parâmetros estão localizados no descritor *SCL* de cada projeto.

4.3.2 Descrição do Desenvolvimento do Projeto

A simulação foi implementada usando a estratégia de processamento seqüencial e a estratégia de processamento distribuído. No processamento seqüencial apenas um

computador foi utilizado. Na estratégia de processamento seqüencial a segunda rede é simulada após o término da simulação da primeira rede. No processamento distribuído foram utilizados dois computadores a fim de diminuir o tempo no processamento para alcançar o resultado. Na estratégia de processamento distribuído, cada uma das redes encontra-se executando em um computador. Durante a simulação da primeira rede, as mensagens recebidas de pós-processamento pelos blocos RECEIVER_0, RECEIVER_1, RECEIVER_2, RECEIVER_3 do "Projeto 1" foram encaminhadas respectivamente para os blocos RECEIVER_0, RECEIVER_1, RECEIVER_2, RECEIVER_3 do "Projeto 2".

A Figura 4-12 ilustra a interconexão entre os projetos, onde a denominação "Projeto 3" é uma abstração concebida para representar a interoperabilidade entre o "Projeto 1" e o "Projeto 2". A Figura 4-13 descreve as interfaces de comunicação em cada modelo para o "Projeto 1" e o "Projeto 2".

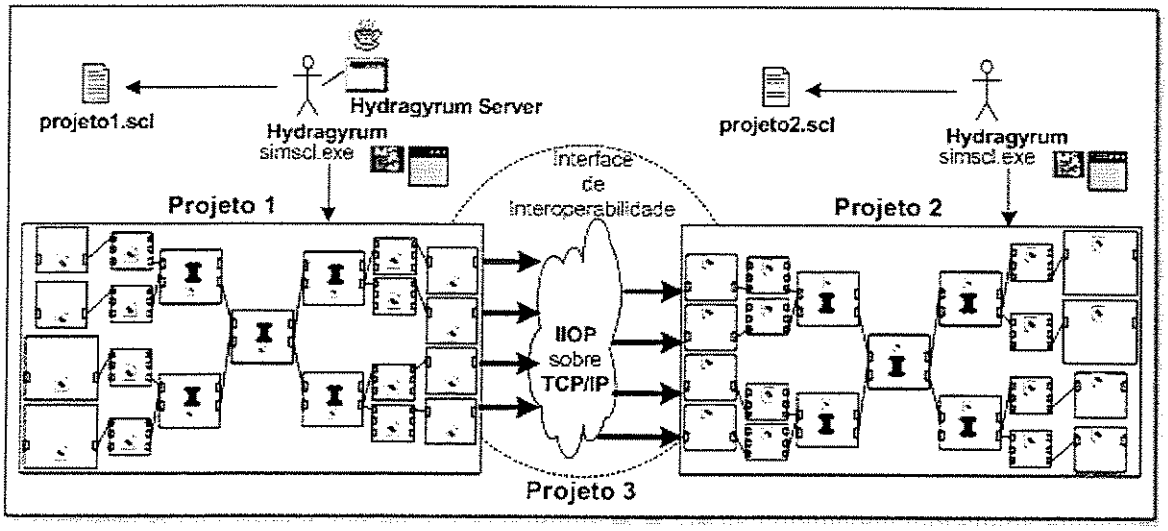

Figura 4-12: Interconexão entre dois projetos em um sistema distribuído

Ref.	Nome do Bloco	Nome da Camada	Nome da Camada (deste ou outro bloco) permitida para se conectar	Nome do Evento
1	SOURCE_0	SOURCE	NODE_INTERFACE	SEND
2	SOURCE_FILE_TIME_0	SOURCE	NODE_INTERFACE	SEND
3	NODE_0	NODE_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
4	NODE_0	NODE_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
5	NODE_0	NODE_INTERFACE	SOURCE	SEND
6	NODE_0	NODE_INTERFACE	RECEIVER_END	RECEIVE
7	NODE_SWITCH_0	NODE_SWITCH_INPUT	NODE_OUTPUT	PHYSICAL
8	NODE_SWITCH_0	NODE_SWITCH_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
9	NODE_SWITCH_0	NODE_SWITCH_OUTPUT	NODE_INPUT	PHYSICAL
10	NODE_SWITCH_0	NODE_SWITCH_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
11	RECEIVER_0	RECEIVER_END	NODE_INTERFACE	RECEIVE
12	SOURCE_1	SOURCE	NODE_INTERFACE	SEND
13	SOURCE_FILE_TIME_1	SOURCE	NODE_INTERFACE	SEND
14	NODE_1	NODE_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
15	NODE_1	NODE_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
16	NODE_1	NODE_INTERFACE	SOURCE	SEND
17	NODE_1	NODE_INTERFACE	RECEIVER_END	RECEIVE
18	NODE_SWITCH_1	NODE_SWITCH_INPUT	NODE_OUTPUT	PHYSICAL
19	NODE_SWITCH_1	NODE_SWITCH_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
20	NODE_SWITCH_1	NODE_SWITCH_OUTPUT	NODE_INPUT	PHYSICAL
21	NODE_SWITCH_1	NODE_SWITCH_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
22	RECEIVER_1	RECEIVER_END	NODE_INTERFACE	RECEIVE
23	NODE_2	NODE_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
24	NODE_2	NODE_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
25	NODE_2	NODE_INTERFACE	SOURCE	SEND

26	NODE_2	NODE_INTERFACE	RECEIVER_END	RECEIVE
27	NODE_SWITCH_2	NODE_SWITCH_INPUT	NODE_OUTPUT	PHYSICAL
28	NODE_SWITCH_2	NODE_SWITCH_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
29	NODE_SWITCH_2	NODE_SWITCH_OUTPUT	NODE_INPUT	PHYSICAL
30	NODE_SWITCH_2	NODE_SWITCH_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
31	RECEIVER_2	RECEIVER_END	NODE_INTERFACE	RECEIVE
32	NODE_3	NODE_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
33	NODE_3	NODE_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
34	NODE_3	NODE_INTERFACE	SOURCE	SEND
35	NODE_3	NODE_INTERFACE	RECEIVER_END	RECEIVE
36	NODE_SWITCH_3	NODE_SWITCH_INPUT	NODE_OUTPUT	PHYSICAL
37	NODE_SWITCH_3	NODE_SWITCH_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
38	NODE_SWITCH_3	NODE_SWITCH_OUTPUT	NODE_INPUT	PHYSICAL
39	NODE_SWITCH_3	NODE_SWITCH_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
40	RECEIVER_3	RECEIVER_END	NODE_INTERFACE	RECEIVE
41	NODE_4	NODE_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
42	NODE_4	NODE_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
43	NODE_4	NODE_INTERFACE	SOURCE	SEND
44	NODE_4	NODE_INTERFACE	RECEIVER_END	RECEIVE
45	NODE_SWITCH_4	NODE_SWITCH_INPUT	NODE_OUTPUT	PHYSICAL
46	NODE_SWITCH_4	NODE_SWITCH_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
47	NODE_SWITCH_4	NODE_SWITCH_OUTPUT	NODE_INPUT	PHYSICAL
48	NODE_SWITCH_4	NODE_SWITCH_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
49	NODE_5	NODE_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
50	NODE_5	NODE_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
51	NODE_5	NODE_INTERFACE	SOURCE	SEND
52	NODE_5	NODE_INTERFACE	RECEIVER_END	RECEIVE

53	NODE_6	NODE_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
54	NODE_6	NODE_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
55	NODE_6	NODE_INTERFACE	SOURCE	SEND
56	NODE_6	NODE_INTERFACE	RECEIVER_END	RECEIVE
57	NODE_7	NODE_INPUT	NODE_SWITCH_OUTPUT	PHYSICAL
58	NODE_7	NODE_OUTPUT	NODE_SWITCH_INPUT	PHYSICAL
59	NODE_7	NODE_INTERFACE	SOURCE	SEND
60	NODE_7	NODE_INTERFACE	RECEIVER_END	RECEIVE

Figura 4-13: Interfaces de comunicação em cada modelo do Projeto-1 e Projeto-2

O arquivo descritor *SCL* para o "Projeto-1" é listado a seguir.

```
interf block NODE_0 node.dll 26 138
interf block NODE_1 node.dll 125 138
interf block NODE_2 node.dll 226 140
interf block NODE_3 node.dll 329 139
interf block NODE_SWITCH_0 node_switch.dll 81 239
interf block NODE_SWITCH_1 node_switch.dll 268 234
interf block NODE_SWITCH_2 node_switch.dll 72 481
interf block NODE_SWITCH_3 node_switch.dll 284 486
interf block NODE_SWITCH_4 node_switch.dll 184 362
interf block SOURCE_FILE_TIME_0 source_file_time.dll 331 48
interf block NODE_4 node.dll 26 616
interf block NODE_5 node.dll 119 617
interf block NODE_6 node.dll 234 617
interf block NODE_7 node.dll 344 619
interf block SOURCE_0 source.dll 18 51
interf block RECEIVER_0 receiver.dll 25 704
interf block RECEIVER_1 receiver.dll 233 710
interf block RECEIVER_2 receiver.dll 349 714
interf block RECEIVER_3 receiver.dll 120 709
interf block SOURCE_1 source.dll 234 50
interf block SOURCE_FILE_TIME_1 source_file_time.dll 127 50
interf line Connec_0  NODE_0 NODE_SWITCH_0 OUTPUT INPUT 192 74 RIGHT 239 108 LEFT
interf line Connec_1  NODE_1 NODE_SWITCH_0 OUTPUT INPUT 192 178 RIGHT 239 110 LEFT
interf line Connec_2  NODE_2 NODE_SWITCH_1 OUTPUT INPUT 194 272 RIGHT 234 293 LEFT
interf line Connec_3  NODE_3 NODE_SWITCH_1 OUTPUT INPUT 193 378 RIGHT 234 294 LEFT
interf line Connec_4  NODE_SWITCH_1 NODE_SWITCH_4 OUTPUT INPUT 284 309 RIGHT 362 213 LEFT
interf line Connec_5  NODE_SWITCH_0 NODE_SWITCH_4 OUTPUT INPUT 289 123 RIGHT 362 211 LEFT
interf line Connec_6  NODE_SWITCH_4 NODE_SWITCH_2 OUTPUT INPUT 412 228 RIGHT 481 99 LEFT
interf line Connec_7  NODE_SWITCH_4 NODE_SWITCH_3 OUTPUT INPUT 412 225 RIGHT 486 310 LEFT
interf line Connec_15  SOURCE_FILE_TIME_0 NODE_3 SOURCE_FILE_TIME_SRC INTERFACE 100 357 RIGHT 139 346 LEFT
interf line Connec_9  NODE_SWITCH_2 NODE_4 OUTPUT INPUT 531 112 RIGHT 616 57 LEFT
interf line Connec_10  NODE_SWITCH_2 NODE_5 OUTPUT INPUT 531 115 RIGHT 617 152 LEFT
```

```
interf line Connec_11  NODE_SWITCH_3 NODE_6 OUTPUT INPUT 536 324 RIGHT 617 266 LEFT
interf line Connec_12  NODE_SWITCH_3 NODE_7 OUTPUT INPUT 536 326 RIGHT 619 377 LEFT
interf line Connec_13  NODE_4 RECEIVER_0 INTERFACE END 670 41 RIGHT 704 53 LEFT
interf line Connec_16  NODE_7 RECEIVER_2 INTERFACE END 673 360 RIGHT 714 378 LEFT
interf line Connec_17  SOURCE_0 NODE_0 SOURCE_CONSTANT INTERFACE 103 43 RIGHT 138 42 LEFT
interf line Connec_18  SOURCE_FILE_TIME_1 NODE_1 SOURCE_FILE_TIME_SRC INTERFACE 102 151 RIGHT 138 143 LEFT
interf line Connec_19  SOURCE_1 NODE_2 SOURCE_CONSTANT INTERFACE 102 257 RIGHT 140 244 LEFT
interf line Connec_21  NODE_5 RECEIVER_3 INTERFACE END 671 134 RIGHT 709 148 LEFT
interf line Connec_22  NODE_6 RECEIVER_1 INTERFACE END 671 251 RIGHT 710 262 LEFT
create netconnection NC_0 NC
addconnec NC_0 Connec_0
addconnec NC_0 Connec_5
addconnec NC_0 Connec_6
addconnec NC_0 Connec_9
complete nc NC_0 NODE_0 NODE_4
create netconnection NC_1 NC
addconnec NC_1 Connec_1
addconnec NC_1 Connec_5
addconnec NC_1 Connec_7
addconnec NC_1 Connec_11
complete nc NC_1 NODE_1 NODE_6
create netconnection NC_2 NC
addconnec NC_2 Connec_2
addconnec NC_2 Connec_4
addconnec NC_2 Connec_7
addconnec NC_2 Connec_12
complete nc NC_2 NODE_2 NODE_7
create netconnection NC_3 NC
addconnec NC_3 Connec_3
addconnec NC_3 Connec_4
addconnec NC_3 Connec_6
addconnec NC_3 Connec_10
complete nc NC_3 NODE_3 NODE_5
create dataconnection DC_0 DC
addnetconnec DC_0 NC_0
complete dc DC_0 SOURCE_0 RECEIVER_0 SOURCE_CONSTANT END
create dataconnection DC_1 DC
addnetconnec DC_1 NC_1
complete dc DC_1 SOURCE_FILE_TIME_1 RECEIVER_1 SOURCE_FILE_TIME_SRC END
create dataconnection DC_2 DC
addnetconnec DC_2 NC_2
complete dc DC_2 SOURCE_1 RECEIVER_2 SOURCE_CONSTANT END
create dataconnection DC_3 DC
addnetconnec DC_3 NC_3
complete dc DC_3 SOURCE_FILE_TIME_0 RECEIVER_3 SOURCE_FILE_TIME_SRC END
param block "NODE_0" "OutPut Files" "string" 6 4 "output_prj1/NODE_0_INTERFACE_delay.dat" "Log file for the
total cell delay in the network." "DATA" "OFF" "output_prj1/NODE_0_INTERFACE_input.dat" "Log file for the node
input traffic." "DATA" "OFF" "output_prj1/NODE_0_QUEUE_event.dat" "Log file for the event sampling of the
buffer." "DATA" "OFF" "output_prj1/NODE_0_QUEUE_evservice.dat" "Log file for the event sampling of the buffer."
"DATA" "OFF" "output_prj1/NODE_0_QUEUE_stat.dat" "BTE queueing system statistics." "TEXT" "OFF"
"output_prj1/NODE_0_QUEUE_delay.dat" "Log file for the cell delay in the queueing system." "DATA" "OFF" "List
of Block Output Files." "SHOW"
param layer "NODE_0" "INTERFACE" "Generate Input Traffic File" "int" 1 1 0 "Generate the file with the input
traffic for this node" "SHOW"
param layer "NODE_0" "INTERFACE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays."
"SHOW"
param layer "NODE_0" "INTERFACE" "Cell Packetization Delay" "double" 1 1 1e-08 "Delay to process a cell in the
interface (seconds)." "SHOW"
```

```
param queue "NODE_0" "QUEUE" "Service Rate" "double" 1 1 2.048e+06 "Service rate (cells / second)." "SHOW"
param queue "NODE_0" "QUEUE" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_0" "QUEUE" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_0" "QUEUE" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue."
"SHOW"
param queue "NODE_0" "QUEUE" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_0" "QUEUE" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing
system." "SHOW"
param queue "NODE_0" "QUEUE" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the
physical medium (seconds)." "SHOW"
param queue "NODE_0" "QUEUE" "Distance" "double" 1 1 10 "Length of the connection physycal link (meters)."
"SHOW"
param queue "NODE_0" "QUEUE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays."
"SHOW"
param queue "NODE_0" "QUEUE" "Generate Statistics" "int" 1 1 0 "Switch ON(1) or OFF(0) the generation of the
statistics output file" "SHOW"
param queue "NODE_0" "QUEUE" "Trigger by Service Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of
buffer service events." "SHOW"
param queue "NODE_0" "QUEUE" "Trigger by Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer
events." "SHOW"
param block "NODE_1" "OutPut Files" "string" 6 4 "output_prj1/NODE_1_INTERFACE_delay.dat" "Log file for the
total cell delay in the network." "DATA" "OFF" "output_prj1/NODE_1_INTERFACE_input.dat" "Log file for the node
input traffic." "DATA" "OFF" "output_prj1/NODE_1_QUEUE_event.dat" "Log file for the event sampling of the
buffer." "DATA" "OFF" "output_prj1/NODE_1_QUEUE_evservice.dat" "Log file for the event sampling of the buffer."
"DATA" "OFF" "output_prj1/NODE_1_QUEUE_stat.dat" "BTE queueing system statistics." "TEXT" "OFF"
"output_prj1/NODE_1_QUEUE_delay.dat" "Log file for the cell delay in the queueing system." "DATA" "OFF" "List
of Block Output Files." "SHOW"
param layer "NODE_1" "INTERFACE" "Generate Input Traffic File" "int" 1 1 0 "Generate the file with the input
traffic for this node" "SHOW"
param layer "NODE_1" "INTERFACE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays."
"SHOW"
param layer "NODE_1" "INTERFACE" "Cell Packetization Delay" "double" 1 1 1e-08 "Delay to process a cell in the
interface (seconds)." "SHOW"
param queue "NODE_1" "QUEUE" "Service Rate" "double" 1 1 1.55e+011 "Service rate (cells / second)." "SHOW"
param queue "NODE_1" "QUEUE" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_1" "QUEUE" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_1" "QUEUE" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue."
"SHOW"
param queue "NODE_1" "QUEUE" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_1" "QUEUE" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing
system." "SHOW"
param queue "NODE_1" "QUEUE" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the
physical medium (seconds)." "SHOW"
param queue "NODE_1" "QUEUE" "Distance" "double" 1 1 10 "Length of the connection physycal link (meters)."
"SHOW"
param queue "NODE_1" "QUEUE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays."
"SHOW"
param queue "NODE_1" "QUEUE" "Generate Statistics" "int" 1 1 0 "Switch ON(1) or OFF(0) the generation of the
statistics output file" "SHOW"
param queue "NODE_1" "QUEUE" "Trigger by Service Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of
buffer service events." "SHOW"
param queue "NODE_1" "QUEUE" "Trigger by Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer
events." "SHOW"
param block "NODE_2" "OutPut Files" "string" 6 4 "output_prj1/NODE_2_INTERFACE_delay.dat" "Log file for the
total cell delay in the network." "DATA" "OFF" "output_prj1/NODE_2_INTERFACE_input.dat" "Log file for the node
input traffic." "DATA" "OFF" "output_prj1/NODE_2_QUEUE_event.dat" "Log file for the event sampling of the
buffer." "DATA" "OFF" "output_prj1/NODE_2_QUEUE_evservice.dat" "Log file for the event sampling of the buffer."
"DATA" "OFF" "output_prj1/NODE_2_QUEUE_stat.dat" "BTE queueing system statistics." "TEXT" "OFF"
"output_prj1/NODE_2_QUEUE_delay.dat" "Log file for the cell delay in the queueing system." "DATA" "OFF" "List
of Block Output Files." "SHOW"
param layer "NODE_2" "INTERFACE" "Generate Input Traffic File" "int" 1 1 0 "Generate the file with the input
traffic for this node" "SHOW"
param layer "NODE_2" "INTERFACE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays."
```

```
"SHOW"
param layer "NODE_2" "INTERFACE" "Cell Packetization Delay" "double" 1 1 1e-08 "Delay to process a cell in the interface (seconds)." "SHOW"
param queue "NODE_2" "QUEUE" "Service Rate" "double" 1 1 3.4e+07 "Service rate (cells / second)." "SHOW"
param queue "NODE_2" "QUEUE" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_2" "QUEUE" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_2" "QUEUE" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_2" "QUEUE" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_2" "QUEUE" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
param queue "NODE_2" "QUEUE" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the physical medium (seconds)." "SHOW"
param queue "NODE_2" "QUEUE" "Distance" "double" 1 1 10 "Length of the connection physycal link (meters)." "SHOW"
param queue "NODE_2" "QUEUE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param queue "NODE_2" "QUEUE" "Generate Statistics" "int" 1 1 0 "Switch ON(1) or OFF(0) the generation of the statistics output file" "SHOW"
param queue "NODE_2" "QUEUE" "Trigger by Service Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer service events." "SHOW"
param queue "NODE_2" "QUEUE" "Trigger by Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer events." "SHOW"
param block "NODE_3" "OutPut Files" "string" 6 4 "output_prj1/NODE_3_INTERFACE_delay.dat" "Log file for the total cell delay in the network." "DATA" "OFF" "output_prj1/NODE_3_INTERFACE_input.dat" "Log file for the node input traffic." "DATA" "OFF" "output_prj1/NODE_3_QUEUE_event.dat" "Log file for the event sampling of the buffer." "DATA" "OFF" "output_prj1/NODE_3_QUEUE_evservice.dat" "Log file for the event sampling of the buffer." "DATA" "OFF" "output_prj1/NODE_3_QUEUE_stat.dat" "BTE queueing system statistics." "TEXT" "OFF" "output_prj1/NODE_3_QUEUE_delay.dat" "Log file for the cell delay in the queueing system." "DATA" "OFF" "List of Block Output Files." "SHOW"
param layer "NODE_3" "INTERFACE" "Generate Input Traffic File" "int" 1 1 0 "Generate the file with the input traffic for this node" "SHOW"
param layer "NODE_3" "INTERFACE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param layer "NODE_3" "INTERFACE" "Cell Packetization Delay" "double" 1 1 1e-08 "Delay to process a cell in the interface (seconds)." "SHOW"
param queue "NODE_3" "QUEUE" "Service Rate" "double" 1 1 1.55e+011 "Service rate (cells / second)." "SHOW"
param queue "NODE_3" "QUEUE" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_3" "QUEUE" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_3" "QUEUE" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_3" "QUEUE" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_3" "QUEUE" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
param queue "NODE_3" "QUEUE" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the physical medium (seconds)." "SHOW"
param queue "NODE_3" "QUEUE" "Distance" "double" 1 1 10 "Length of the connection physycal link (meters)." "SHOW"
param queue "NODE_3" "QUEUE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param queue "NODE_3" "QUEUE" "Generate Statistics" "int" 1 1 0 "Switch ON(1) or OFF(0) the generation of the statistics output file" "SHOW"
param queue "NODE_3" "QUEUE" "Trigger by Service Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer service events." "SHOW"
param queue "NODE_3" "QUEUE" "Trigger by Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer events." "SHOW"
param block "NODE_SWITCH_0" "Processing Delay" "double" 1 1 1e-09 "Delay to switch a cell in seconds." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
```

```
param queue "NODE_SWITCH_0" "Connec_5" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the physical medium." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Distance" "double" 1 1 10 "Length of the port physical connection." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Trigger By Event Status" "int" 1 1 0 "Switch ON or OFF the by event sampling of buffer statistics." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Sampling Interval" "double" 1 1 0.1 "Sampling interval used to sample queue state variables" "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Generate Time Sample" "int" 1 1 0 "Generate the output filw with time samples of queue state variables" "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Generate Service File" "int" 1 1 0 "Generate the output file with cell service instants." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param queue "NODE_SWITCH_0" "Connec_5" "Generate Statistics File" "int" 1 1 1 "Generate the output to the statistics file." "SHOW"
param block "NODE_SWITCH_1" "Processing Delay" "double" 1 1 1e-09 "Delay to switch a cell in seconds." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the physical medium." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Distance" "double" 1 1 10 "Length of the port physical connection." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Trigger By Event Status" "int" 1 1 0 "Switch ON or OFF the by event sampling of buffer statistics." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Sampling Interval" "double" 1 1 0.1 "Sampling interval used to sample queue state variables" "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Generate Time Sample" "int" 1 1 0 "Generate the output filw with time samples of queue state variables" "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Generate Service File" "int" 1 1 0 "Generate the output file with cell service instants." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param queue "NODE_SWITCH_1" "Connec_4" "Generate Statistics File" "int" 1 1 1 "Generate the output to the statistics file." "SHOW"
param block "NODE_SWITCH_2" "Processing Delay" "double" 1 1 1e-09 "Delay to switch a cell in seconds." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
```

```
param queue "NODE_SWITCH_2" "Connec_9" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame
in the physical medium." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Distance" "double" 1 1 10 "Length of the port physical connection."
"SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Trigger By Event Status" "int" 1 1 0 "Switch ON or OFF the by event
sampling of buffer statistics." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Sampling Interval" "double" 1 1 0.1 "Sampling interval used to sample
queue state variables" "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Generate Time Sample" "int" 1 1 0 "Generate the output filw with time
samples of queue state variables" "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Generate Service File" "int" 1 1 0 "Generate the output file with cell
service instants." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell
delays." "SHOW"
param queue "NODE_SWITCH_2" "Connec_9" "Generate Statistics File" "int" 1 1 1 "Generate the output to the
statistics file." "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)."
"SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)."
"SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue."
"SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the
queue." "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the
queueing system." "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame
in the physical medium." "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Distance" "double" 1 1 10 "Length of the port physical connection."
"SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Trigger By Event Status" "int" 1 1 0 "Switch ON or OFF the by event
sampling of buffer statistics." "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Sampling Interval" "double" 1 1 0.1 "Sampling interval used to sample
queue state variables" "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Generate Time Sample" "int" 1 1 0 "Generate the output filw with time
samples of queue state variables" "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Generate Service File" "int" 1 1 0 "Generate the output file with cell
service instants." "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell
delays." "SHOW"
param queue "NODE_SWITCH_2" "Connec_10" "Generate Statistics File" "int" 1 1 1 "Generate the output to the
statistics file." "SHOW"
param block "NODE_SWITCH_3" "Processing Delay" "double" 1 1 1e-09 "Delay to switch a cell in seconds." "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)."
"SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)."
"SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue."
"SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the
queue." "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the
queueing system." "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame
in the physical medium." "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Distance" "double" 1 1 10 "Length of the port physical connection."
"SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Trigger By Event Status" "int" 1 1 0 "Switch ON or OFF the by event
sampling of buffer statistics." "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Sampling Interval" "double" 1 1 0.1 "Sampling interval used to sample
```

```
queue state variables" "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Generate Time Sample" "int" 1 1 0 "Generate the output filw with time samples of queue state variables" "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Generate Service File" "int" 1 1 0 "Generate the output file with cell service instants." "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param queue "NODE_SWITCH_3" "Connec_11" "Generate Statistics File" "int" 1 1 1 "Generate the output to the statistics file." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the physical medium." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Distance" "double" 1 1 10 "Length of the port physical connection." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Trigger By Event Status" "int" 1 1 0 "Switch ON or OFF the by event sampling of buffer statistics." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Sampling Interval" "double" 1 1 0.1 "Sampling interval used to sample queue state variables" "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Generate Time Sample" "int" 1 1 0 "Generate the output filw with time samples of queue state variables" "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Generate Service File" "int" 1 1 0 "Generate the output file with cell service instants." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param queue "NODE_SWITCH_3" "Connec_12" "Generate Statistics File" "int" 1 1 1 "Generate the output to the statistics file." "SHOW"
param block "NODE_SWITCH_4" "Processing Delay" "double" 1 1 1e-09 "Delay to switch a cell in seconds." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the physical medium." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Distance" "double" 1 1 10 "Length of the port physical connection." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Trigger By Event Status" "int" 1 1 0 "Switch ON or OFF the by event sampling of buffer statistics." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Sampling Interval" "double" 1 1 0.1 "Sampling interval used to sample queue state variables" "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Generate Time Sample" "int" 1 1 0 "Generate the output filw with time samples of queue state variables" "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Generate Service File" "int" 1 1 0 "Generate the output file with cell service instants." "SHOW"
param queue "NODE_SWITCH_4" "Connec_6" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
```

```
param queue "NODE_SWITCH_4" "Connec_6" "Generate Statistics File" "int" 1 1 1 "Generate the output to the statistics file." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the physical medium." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Distance" "double" 1 1 10 "Length of the port physical connection." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Trigger By Event Status" "int" 1 1 0 "Switch ON or OFF the by event sampling of buffer statistics." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Sampling Interval" "double" 1 1 0.1 "Sampling interval used to sample queue state variables" "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Generate Time Sample" "int" 1 1 0 "Generate the output filw with time samples of queue state variables" "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Generate Service File" "int" 1 1 0 "Generate the output file with cell service instants." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param queue "NODE_SWITCH_4" "Connec_7" "Generate Statistics File" "int" 1 1 0 "Generate the output to the statistics file." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Source Rate" "double" 1 1 15000 "Constant source rate for packet generation in (bits / seconds)." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Start Delay" "double" 1 1 0 "Delay to start packet transmition." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Packet Limited" "int" 1 1 1 "Finish the transmition when it reaches Max Number Of Packets For Trasmition." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Time Limited" "int" 1 1 0 "Finish the transmition at Final Transmition Time." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Max Number Of Packets For Trasmition" "double" 1 1 1e+09 "Total number of packets to be  trasmited  by the source." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Final Transmition Time" "double" 1 1 1e-06 "Transmition end time ( s ) to stop the source activity." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "External File Name" "string" 1 1 "output_prj1/h08_2.dat" "Name of the source input file." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Use Path" "int" 1 1 1 "Use the path entered by the user to search the file." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Path" "string" 1 1 "C:\Hydragyrum\projects\sample3\project_3\final\Release\" "Use the path to search the file (end path with \)." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Priority" "int" 1 1 0 "Priority of the packets sent by the source." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Length" "double" 1 1 40 "Source packet length." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Number Of Trasmited Packets" "double" 1 1 0 "Total number of packets trasmited by the source." "SHOW"
param layer "SOURCE_FILE_TIME_0" "SOURCE_FILE_TIME_SRC" "Stop Time" "double" 1 1 0 "End time of the source activity." "SHOW"
param block "NODE_4" "OutPut Files" "string" 6 4 "output_prj1/NODE_4_INTERFACE_delay.dat" "Log file for the total cell delay in the network." "DATA" "OFF" "output_prj1/NODE_4_INTERFACE_input.dat" "Log file for the node input traffic." "DATA" "OFF" "output_prj1/NODE_4_QUEUE_event.dat" "Log file for the event sampling of the buffer." "DATA" "OFF" "output_prj1/NODE_4_QUEUE_evservice.dat" "Log file for the event sampling of the buffer." "DATA" "OFF" "output_prj1/NODE_4_QUEUE_stat.dat" "BTE queueing system statistics." "TEXT" "OFF" "output_prj1/NODE_4_QUEUE_delay.dat" "Log file for the cell delay in the queueing system." "DATA" "OFF" "List of Block Output Files." "SHOW"
param layer "NODE_4" "INTERFACE" "Generate Input Traffic File" "int" 1 1 0 "Generate the file with the input
```

```
traffic for this node" "SHOW"
param layer "NODE_4" "INTERFACE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param layer "NODE_4" "INTERFACE" "Cell Packetization Delay" "double" 1 1 1e-08 "Delay to process a cell in the interface (seconds)." "SHOW"
param queue "NODE_4" "QUEUE" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_4" "QUEUE" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_4" "QUEUE" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_4" "QUEUE" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_4" "QUEUE" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_4" "QUEUE" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
param queue "NODE_4" "QUEUE" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the physical medium (seconds)." "SHOW"
param queue "NODE_4" "QUEUE" "Distance" "double" 1 1 10 "Length of the connection physycal link (meters)." "SHOW"
param queue "NODE_4" "QUEUE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param queue "NODE_4" "QUEUE" "Generate Statistics" "int" 1 1 0 "Switch ON(1) or OFF(0) the generation of the statistics output file" "SHOW"
param queue "NODE_4" "QUEUE" "Trigger by Service Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer service events." "SHOW"
param queue "NODE_4" "QUEUE" "Trigger by Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer events." "SHOW"
param block "NODE_5" "OutPut Files" "string" 6 4 "output_prj1/NODE_5_INTERFACE_delay.dat" "Log file for the total cell delay in the network." "DATA" "OFF" "output_prj1/NODE_5_INTERFACE_input.dat" "Log file for the node input traffic." "DATA" "OFF" "output_prj1/NODE_5_QUEUE_event.dat" "Log file for the event sampling of the buffer." "DATA" "OFF" "output_prj1/NODE_5_QUEUE_evservice.dat" "Log file for the event sampling of the buffer." "DATA" "OFF" "output_prj1/NODE_5_QUEUE_stat.dat" "BTE queueing system statistics." "TEXT" "OFF" "output_prj1/NODE_5_QUEUE_delay.dat" "Log file for the cell delay in the queueing system." "DATA" "OFF" "List of Block Output Files." "SHOW"
param layer "NODE_5" "INTERFACE" "Generate Input Traffic File" "int" 1 1 0 "Generate the file with the input traffic for this node" "SHOW"
param layer "NODE_5" "INTERFACE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param layer "NODE_5" "INTERFACE" "Cell Packetization Delay" "double" 1 1 1e-08 "Delay to process a cell in the interface (seconds)." "SHOW"
param queue "NODE_5" "QUEUE" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_5" "QUEUE" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_5" "QUEUE" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_5" "QUEUE" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue." "SHOW"
param queue "NODE_5" "QUEUE" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_5" "QUEUE" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing system." "SHOW"
param queue "NODE_5" "QUEUE" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the physical medium (seconds)." "SHOW"
param queue "NODE_5" "QUEUE" "Distance" "double" 1 1 10 "Length of the connection physycal link (meters)." "SHOW"
param queue "NODE_5" "QUEUE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays." "SHOW"
param queue "NODE_5" "QUEUE" "Generate Statistics" "int" 1 1 0 "Switch ON(1) or OFF(0) the generation of the statistics output file" "SHOW"
param queue "NODE_5" "QUEUE" "Trigger by Service Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer service events." "SHOW"
param queue "NODE_5" "QUEUE" "Trigger by Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer events." "SHOW"
param block "NODE_6" "OutPut Files" "string" 6 4 "output_prj1/NODE_6_INTERFACE_delay.dat" "Log file for the total cell delay in the network." "DATA" "OFF" "output_prj1/NODE_6_INTERFACE_input.dat" "Log file for the node input traffic." "DATA" "OFF" "output_prj1/NODE_6_QUEUE_event.dat" "Log file for the event sampling of the buffer." "DATA" "OFF" "output_prj1/NODE_6_QUEUE_evservice.dat" "Log file for the event sampling of the buffer." "DATA" "OFF" "output_prj1/NODE_6_QUEUE_stat.dat" "BTE queueing system statistics." "TEXT" "OFF"
```

```
"output_prj1/NODE_6_QUEUE_delay.dat" "Log file for the cell delay in the queueing system." "DATA" "OFF" "List
of Block Output Files." "SHOW"
param layer "NODE_6" "INTERFACE" "Generate Input Traffic File" "int" 1 1 0 "Generate the file with the input
traffic for this node" "SHOW"
param layer "NODE_6" "INTERFACE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays."
"SHOW"
param layer "NODE_6" "INTERFACE" "Cell Packetization Delay" "double" 1 1 1e-08 "Delay to process a cell in the
interface (seconds)." "SHOW"
param queue "NODE_6" "QUEUE" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_6" "QUEUE" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_6" "QUEUE" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_6" "QUEUE" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue."
"SHOW"
param queue "NODE_6" "QUEUE" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_6" "QUEUE" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing
system." "SHOW"
param queue "NODE_6" "QUEUE" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the
physical medium (seconds)." "SHOW"
param queue "NODE_6" "QUEUE" "Distance" "double" 1 1 10 "Length of the connection physycal link (meters)."
"SHOW"
param queue "NODE_6" "QUEUE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays."
"SHOW"
param queue "NODE_6" "QUEUE" "Generate Statistics" "int" 1 1 0 "Switch ON(1) or OFF(0) the generation of the
statistics output file" "SHOW"
param queue "NODE_6" "QUEUE" "Trigger by Service Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of
buffer service events." "SHOW"
param queue "NODE_6" "QUEUE" "Trigger by Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer
events." "SHOW"
param block "NODE_7" "OutPut Files" "string" 6 4 "output_prj1/NODE_7_INTERFACE_delay.dat" "Log file for the
total cell delay in the network." "DATA" "OFF" "output_prj1/NODE_7_INTERFACE_input.dat" "Log file for the node
input traffic." "DATA" "OFF" "output_prj1/NODE_7_QUEUE_event.dat" "Log file for the event sampling of the
buffer." "DATA" "OFF" "output_prj1/NODE_7_QUEUE_evservice.dat" "Log file for the event sampling of the buffer."
"DATA" "OFF" "output_prj1/NODE_7_QUEUE_stat.dat" "BTE queueing system statistics." "TEXT" "OFF"
"output_prj1/NODE_7_QUEUE_delay.dat" "Log file for the cell delay in the queueing system." "DATA" "OFF" "List
of Block Output Files." "SHOW"
param layer "NODE_7" "INTERFACE" "Generate Input Traffic File" "int" 1 1 0 "Generate the file with the input
traffic for this node" "SHOW"
param layer "NODE_7" "INTERFACE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays."
"SHOW"
param layer "NODE_7" "INTERFACE" "Cell Packetization Delay" "double" 1 1 1e-08 "Delay to process a cell in the
interface (seconds)." "SHOW"
param queue "NODE_7" "QUEUE" "Service Rate" "double" 1 1 1.55e+08 "Service rate (cells / second)." "SHOW"
param queue "NODE_7" "QUEUE" "Queue Capacity" "long" 1 1 100000000 "Queue storage capacity (cells)." "SHOW"
param queue "NODE_7" "QUEUE" "Queue Size" "long" 1 1 0 "Queue size (cells)." "SHOW"
param queue "NODE_7" "QUEUE" "Queue ReceivedCells" "double" 1 1 0 "Number of cells received by the queue."
"SHOW"
param queue "NODE_7" "QUEUE" "Queue SentCells" "double" 1 1 0 "Number of cells sent by the queue." "SHOW"
param queue "NODE_7" "QUEUE" "Discarded Cells" "double" 1 1 0 "Number of cells discarded by the queueing
system." "SHOW"
param queue "NODE_7" "QUEUE" "Propagation Delay" "double" 1 1 3.33333e-08 "Delay to trasmit a frame in the
physical medium (seconds)." "SHOW"
param queue "NODE_7" "QUEUE" "Distance" "double" 1 1 10 "Length of the connection physycal link (meters)."
"SHOW"
param queue "NODE_7" "QUEUE" "Generate Delay File" "int" 1 1 0 "Generate the output file with cell delays."
"SHOW"
param queue "NODE_7" "QUEUE" "Generate Statistics" "int" 1 1 0 "Switch ON(1) or OFF(0) the generation of the
statistics output file" "SHOW"
param queue "NODE_7" "QUEUE" "Trigger by Service Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of
buffer service events." "SHOW"
param queue "NODE_7" "QUEUE" "Trigger by Event" "int" 1 1 0 "Switch ON(1) or OFF(0) the sampling of buffer
events." "SHOW"
param layer "SOURCE_0" "SOURCE CONSTANT" "Source Rate" "double" 1 1 2.048e+06 "Constant source rate in (bits /
```

```
seconds)." "SHOW"
param layer "SOURCE_0" "SOURCE_CONSTANT" "Start Delay" "double" 1 1 0 "Delay to start packet transmition."
"SHOW"
param layer "SOURCE_0" "SOURCE_CONSTANT" "Packet Limited" "int" 1 1 1 "Finish the transmition when it reaches
Max Number Of Packets For Trasmition." "SHOW"
param layer "SOURCE_0" "SOURCE_CONSTANT" "Time Limited" "int" 1 1 0 "Finish the transmition at Final
Transmition Time." "SHOW"
param layer "SOURCE_0" "SOURCE_CONSTANT" "Max Number Of Packets For Trasmition" "double" 1 1 1e+09 "Total
number of packets to be  trasmited  by the source." "SHOW"
param layer "SOURCE_0" "SOURCE_CONSTANT" "Final Transmition Time" "double" 1 1 1e-06 "Transmition end time ( s
) to stop the source activity." "SHOW"
param layer "SOURCE_0" "SOURCE_CONSTANT" "Priority" "int" 1 1 0 "Priority of the packets sent by the source."
"SHOW"
param layer "SOURCE_0" "SOURCE_CONSTANT" "Length" "double" 1 1 40 "Source packet length." "SHOW"
param layer "SOURCE_0" "SOURCE_CONSTANT" "Number Of Trasmited Packets" "double" 1 1 0 "Total number of packets
trasmited by the source." "SHOW"
param layer "SOURCE_0" "SOURCE_CONSTANT" "Stop Time" "double" 1 1 0 "End time of the source activity." "SHOW"
param block "RECEIVER_0" "OutPut Files" "string" 6 4 "output_prj1/RECEIVER_0_END_packetjitter.dat" "Receiver
packet jitter log file." "DATA" "OFF" "output_prj1/RECEIVER_0_END_celljitter.dat" "Receiver cell jitter log
file." "DATA" "OFF" "output_prj1/RECEIVER_0_END_packetdelay.dat" "Receiver packet delay log file." "DATA" "OFF"
"output_prj1/RECEIVER_0_END_celldelay.dat" "Receiver cell delay log file." "DATA" "OFF"
"output_prj1/RECEIVER_0_END_stat.dat" "Receiver statistics." "TEXT" "OFF"
"output_prj1/RECEIVER_0_END_timedelay.dat" "Time sampling of delay" "DATA" "OFF" "List of Block Output Files."
"SHOW"
param layer "RECEIVER_0" "END" "Sampling Interval" "double" 1 1 0.1 "Sampling Interval for the time sampling
analysis" "SHOW"
param layer "RECEIVER_0" "END" "LogVarianceStep" "int" 1 1 1 "Sampling step of variance values in the cell and
packet jitter files" "SHOW"
param layer "RECEIVER_0" "END" "LogPacketDelayVariance" "int" 1 1 0 "Set (1) or reset(0) the log of packet
delay variance in the statistcs file" "SHOW"
param layer "RECEIVER_0" "END" "LogCellDelayVariance" "int" 1 1 0 "Set (1) or reset(0) the log of cell delay
variance in the statistcs file" "SHOW"
param layer "RECEIVER_0" "END" "Generate Delay Time Sample" "int" 1 1 0 "Set (1) or reset(0) the generation of
delay sampling with constant time interval" "SHOW"
param layer "RECEIVER_0" "END" "Generate Packet Jitter File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_0" "END" "Generate Cell Jitter File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_0" "END" "Generate Packet Delay File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_0" "END" "Generate Cell Delay File" "int" 1 1 0 "Set (1) or reset(0) the generation of
packet delay log file" "SHOW"
param layer "RECEIVER_0" "END" "Generate Statistics File" "int" 1 1 1 "Set (1) or reset(0) the generation of
statistics log file" "SHOW"
param layer "RECEIVER_0" "END" "Number Of Received Cells" "long" 1 1 0 "Total number of cells received." "SHOW"
param layer "RECEIVER_0" "END" "Number Of Received Packets" "long" 1 1 0 "Total number of packets received."
"SHOW"
param block "RECEIVER_1" "OutPut Files" "string" 6 4 "output_prj1/RECEIVER_1_END_packetjitter.dat" "Receiver
packet jitter log file." "DATA" "OFF" "output_prj1/RECEIVER_1_END_celljitter.dat" "Receiver cell jitter log
file." "DATA" "OFF" "output_prj1/RECEIVER_1_END_packetdelay.dat" "Receiver packet delay log file." "DATA" "OFF"
"output_prj1/RECEIVER_1_END_celldelay.dat" "Receiver cell delay log file." "DATA" "OFF"
"output_prj1/RECEIVER_1_END_stat.dat" "Receiver statistics." "TEXT" "OFF"
"output_prj1/RECEIVER_1_END_timedelay.dat" "Time sampling of delay" "DATA" "OFF" "List of Block Output Files."
"SHOW"
param layer "RECEIVER_1" "END" "Sampling Interval" "double" 1 1 0.1 "Sampling Interval for the time sampling
analysis" "SHOW"
param layer "RECEIVER_1" "END" "LogVarianceStep" "int" 1 1 1 "Sampling step of variance values in the cell and
packet jitter files" "SHOW"
param layer "RECEIVER_1" "END" "LogPacketDelayVariance" "int" 1 1 0 "Set (1) or reset(0) the log of packet
delay variance in the statistcs file" "SHOW"
param layer "RECEIVER_1" "END" "LogCellDelayVariance" "int" 1 1 0 "Set (1) or reset(0) the log of cell delay
variance in the statistcs file" "SHOW"
param layer "RECEIVER_1" "END" "Generate Delay Time Sample" "int" 1 1 0 "Set (1) or reset(0) the generation of
```

```
delay sampling with constant time interval" "SHOW"
param layer "RECEIVER_1" "END" "Generate Packet Jitter File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_1" "END" "Generate Cell Jitter File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_1" "END" "Generate Packet Delay File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_1" "END" "Generate Cell Delay File" "int" 1 1 0 "Set (1) or reset(0) the generation of
packet delay log file" "SHOW"
param layer "RECEIVER_1" "END" "Generate Statistics File" "int" 1 1 1 "Set (1) or reset(0) the generation of
statistics log file" "SHOW"
param layer "RECEIVER_1" "END" "Number Of Received Cells" "long" 1 1 0 "Total number of cells received." "SHOW"
param layer "RECEIVER_1" "END" "Number Of Received Packets" "long" 1 1 0 "Total number of packets received."
"SHOW"
param block "RECEIVER_2" "OutPut Files" "string" 6 4 "output_prj1/RECEIVER_2_END_packetjitter.dat" "Receiver
packet jitter log file." "DATA" "OFF" "output_prj1/RECEIVER_2_END_celljitter.dat" "Receiver cell jitter log
file." "DATA" "OFF" "output_prj1/RECEIVER_2_END_packetdelay.dat" "Receiver packet delay log file." "DATA" "OFF"
"output_prj1/RECEIVER_2_END_celldelay.dat" "Receiver cell delay log file." "DATA" "OFF"
"output_prj1/RECEIVER_2_END_stat.dat" "Receiver statistics." "TEXT" "OFF"
"output_prj1/RECEIVER_2_END_timedelay.dat" "Time sampling of delay" "DATA" "OFF" "List of Block Output Files."
"SHOW"
param layer "RECEIVER_2" "END" "Sampling Interval" "double" 1 1 0.1 "Sampling Interval for the time sampling
analysis" "SHOW"
param layer "RECEIVER_2" "END" "LogVarianceStep" "int" 1 1 1 "Sampling step of variance values in the cell and
packet jitter files" "SHOW"
param layer "RECEIVER_2" "END" "LogPacketDelayVariance" "int" 1 1 0 "Set (1) or reset(0) the log of packet
delay variance in the statistcs file" "SHOW"
param layer "RECEIVER_2" "END" "LogCellDelayVariance" "int" 1 1 0 "Set (1) or reset(0) the log of cell delay
variance in the statistcs file" "SHOW"
param layer "RECEIVER_2" "END" "Generate Delay Time Sample" "int" 1 1 0 "Set (1) or reset(0) the generation of
delay sampling with constant time interval" "SHOW"
param layer "RECEIVER_2" "END" "Generate Packet Jitter File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_2" "END" "Generate Cell Jitter File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_2" "END" "Generate Packet Delay File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_2" "END" "Generate Cell Delay File" "int" 1 1 0 "Set (1) or reset(0) the generation of
packet delay log file" "SHOW"
param layer "RECEIVER_2" "END" "Generate Statistics File" "int" 1 1 1 "Set (1) or reset(0) the generation of
statistics log file" "SHOW"
param layer "RECEIVER_2" "END" "Number Of Received Cells" "long" 1 1 0 "Total number of cells received." "SHOW"
param layer "RECEIVER_2" "END" "Number Of Received Packets" "long" 1 1 0 "Total number of packets received."
"SHOW"
param block "RECEIVER_3" "OutPut Files" "string" 6 4 "output_prj1/RECEIVER_3_END_packetjitter.dat" "Receiver
packet jitter log file." "DATA" "OFF" "output_prj1/RECEIVER_3_END_celljitter.dat" "Receiver cell jitter log
file." "DATA" "OFF" "output_prj1/RECEIVER_3_END_packetdelay.dat" "Receiver packet delay log file." "DATA" "OFF"
"output_prj1/RECEIVER_3_END_celldelay.dat" "Receiver cell delay log file." "DATA" "OFF"
"output_prj1/RECEIVER_3_END_stat.dat" "Receiver statistics." "TEXT" "OFF"
"output_prj1/RECEIVER_3_END_timedelay.dat" "Time sampling of delay" "DATA" "OFF" "List of Block Output Files."
"SHOW"
param layer "RECEIVER_3" "END" "Sampling Interval" "double" 1 1 0.1 "Sampling Interval for the time sampling
analysis" "SHOW"
param layer "RECEIVER_3" "END" "LogVarianceStep" "int" 1 1 1 "Sampling step of variance values in the cell and
packet jitter files" "SHOW"
param layer "RECEIVER_3" "END" "LogPacketDelayVariance" "int" 1 1 0 "Set (1) or reset(0) the log of packet
delay variance in the statistcs file" "SHOW"
param layer "RECEIVER_3" "END" "LogCellDelayVariance" "int" 1 1 0 "Set (1) or reset(0) the log of cell delay
variance in the statistcs file" "SHOW"
param layer "RECEIVER_3" "END" "Generate Delay Time Sample" "int" 1 1 0 "Set (1) or reset(0) the generation of
delay sampling with constant time interval" "SHOW"
param layer "RECEIVER_3" "END" "Generate Packet Jitter File" "int" 1 1 0 "Set (1) or reset(0) the generation of
cell delay log file" "SHOW"
param layer "RECEIVER_3" "END" "Generate Cell Jitter File" "int" 1 1 0 "Set (1) or reset(0) the generation of
```

```
cell delay log file" "SHOW"
param layer "RECEIVER_3" "END" "Generate Packet Delay File" "int" 1 1 0 "Set (1) or reset(0) the generation of cell delay log file" "SHOW"
param layer "RECEIVER_3" "END" "Generate Cell Delay File" "int" 1 1 0 "Set (1) or reset(0) the generation of packet delay log file" "SHOW"
param layer "RECEIVER_3" "END" "Generate Statistics File" "int" 1 1 1 "Set (1) or reset(0) the generation of statistics log file" "SHOW"
param layer "RECEIVER_3" "END" "Number Of Received Cells" "long" 1 1 0 "Total number of cells received." "SHOW"
param layer "RECEIVER_3" "END" "Number Of Received Packets" "long" 1 1 0 "Total number of packets received." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Source Rate" "double" 1 1 3.4e+07 "Constant source rate in (bits / seconds)." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Start Delay" "double" 1 1 0 "Delay to start packet transmition." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Packet Limited" "int" 1 1 1 "Finish the transmition when it reaches Max Number Of Packets For Trasmition." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Time Limited" "int" 1 1 0 "Finish the transmition at Final Transmition Time." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Max Number Of Packets For Trasmition" "double" 1 1 1e+09 "Total number of packets to be  trasmited  by the source." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Final Transmition Time" "double" 1 1 1e-06 "Transmition end time ( s ) to stop the source activity." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Priority" "int" 1 1 0 "Priority of the packets sent by the source." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Length" "double" 1 1 40 "Source packet length." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Number Of Trasmited Packets" "double" 1 1 0 "Total number of packets trasmited by the source." "SHOW"
param layer "SOURCE_1" "SOURCE_CONSTANT" "Stop Time" "double" 1 1 0 "End time of the source activity." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Source Rate" "double" 1 1 15000 "Constant source rate for packet generation in (bits / seconds)." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Start Delay" "double" 1 1 0 "Delay to start packet transmition." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Packet Limited" "int" 1 1 1 "Finish the transmition when it reaches Max Number Of Packets For Trasmition." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Time Limited" "int" 1 1 0 "Finish the transmition at Final Transmition Time." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Max Number Of Packets For Trasmition" "double" 1 1 1e+09 "Total number of packets to be  trasmited  by the source." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Final Transmition Time" "double" 1 1 1e-06 "Transmition end time ( s ) to stop the source activity." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "External File Name" "string" 1 1 "output_prj1/h08_1.dat" "Name of the source input file." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Use Path" "int" 1 1 1 "Use the path entered by the user to search the file." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Path" "string" 1 1 "C:\Hydragyrum\projects\sample3\project_3\final\Release\" "Use the path to search the file (end path with \)." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Priority" "int" 1 1 0 "Priority of the packets sent by the source." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Length" "double" 1 1 40 "Source packet length." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Number Of Trasmited Packets" "double" 1 1 0 "Total number of packets trasmited by the source." "SHOW"
param layer "SOURCE_FILE_TIME_1" "SOURCE_FILE_TIME_SRC" "Stop Time" "double" 1 1 0 "End time of the source activity." "SHOW"
```

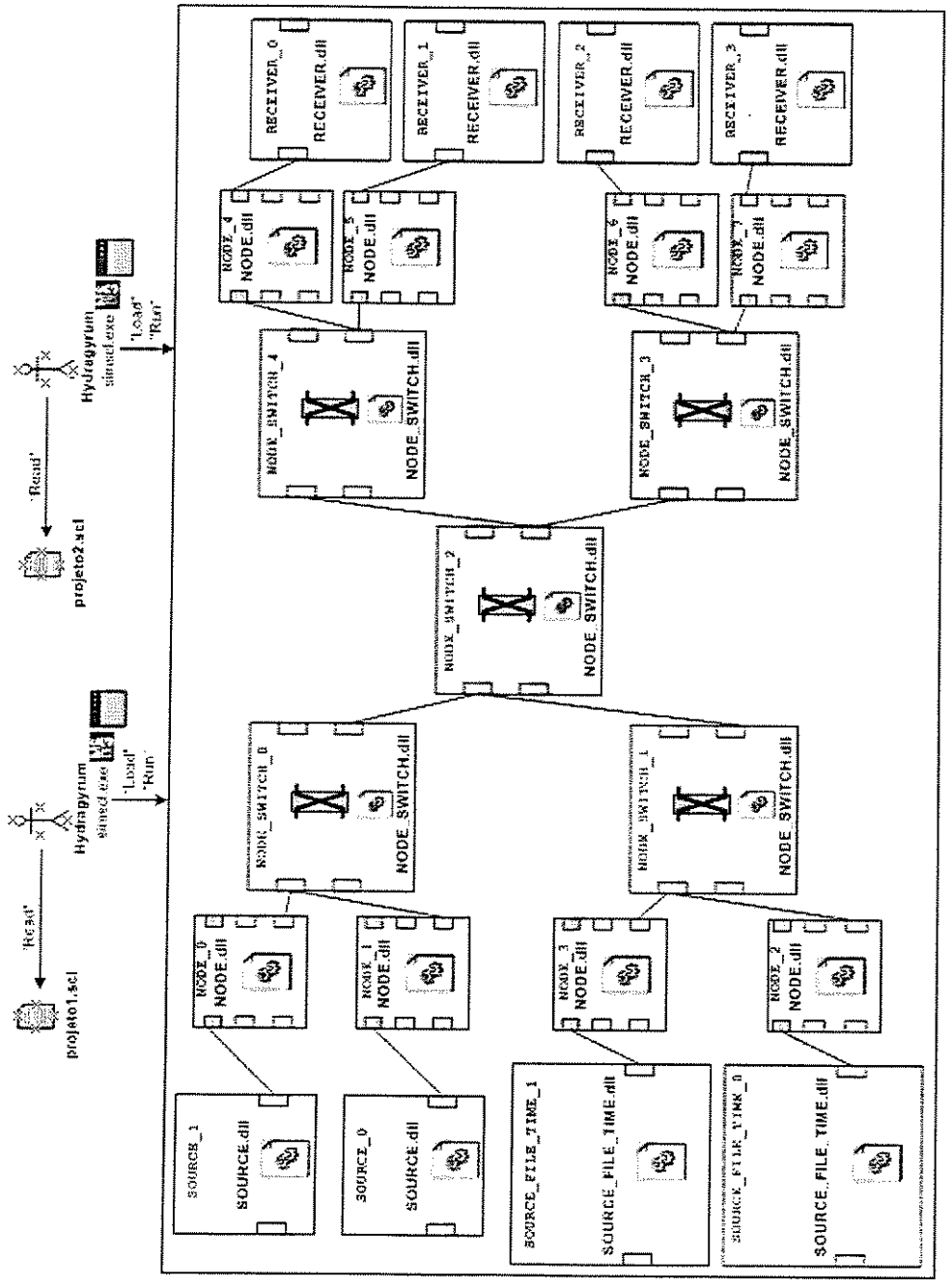

Figura 4-14: Modelos definidos nos descritores “projeto1.scl” e “projeto2.scl”

4.3.3 Resultados da Simulação

Para a realização da simulação foi usada a seguinte configuração dos computadores: CPU *PIII-733 MHz, 512 Mb RAM, Sistema Operacional Microsoft Windows™ 2000 Professional*. Os resultados numéricos obtidos das simulações estão descritos a seguir.

Relógio interno Simulador	Resultados obtidos no modelo "RECEIVER_0"
1	Número de pacotes recebidos: 4829 Máxima latência dos pacotes (s): 0.000217536357569714 Variação estimada de atraso dos pacotes: 1.05204428317398e-012
2	Número de pacotes recebidos: 9659 Máxima latência dos pacotes (s): 0.000217536357621562 Variação estimada de atraso dos pacotes: 1.05111764730566e-012
3	Número de pacotes recebidos: 14489 Máxima latência dos pacotes (s): 0.00021753635767352 Variação estimada de atraso dos pacotes: 1.05062798568502e-012
4	Número de pacotes recebidos: 19319 Máxima latência dos pacotes (s): 0.00021753635772237 Variação estimada de atraso dos pacotes: 1.04995362051402e-012
5	Número de pacotes recebidos: 24149 Máxima latência dos pacotes (s): 0.00021753635772237 Variação estimada de atraso dos pacotes: 1.05048322905524e-012
10	Número de pacotes recebidos: 48300 Máxima latência dos pacotes (s): 0.000218006515517288 Variação estimada de atraso dos pacotes: 1.05094825755415e-012

- Simulação através de processamento seqüencial:

Relógio interno Simulador	Resultados "Projeto-1"	Resultados "Projeto-2"
1	Tempo de execução: 91 s Eventos processados: 1.44528e+006 15882.2 (eventos/s)	Tempo de execução: 91 s Eventos processados: 1.44528e+006 15882.2 (eventos/s)
2	Tempo de execução: 183 s Eventos processados: 2.8906e+006 15795.6 (eventos/s)	Tempo de execução: 183 s Eventos processados: 2.8906e+006 15795.6 (eventos/s)
3	Tempo de execução: 276 s Eventos processados: 4.33592e+006 15709.9 (eventos/s)	Tempo de execução: 276 s Eventos processados: 4.33592e+006 15709.9 (eventos/s)
4	Tempo de execução: 368 s Eventos processados: 5.78124e+006 15709.9 (eventos/s)	Tempo de execução: 368 s Eventos processados: 5.78124e+006 15709.9 (eventos/s)

5	Tempo de execução: 455 s Eventos processados: 7.22655e+006 15882.5 (eventos/s)	Tempo de execução: 455 s Eventos processados: 7.22655e+006 15882.5 (eventos/s)
10	Tempo de execução: 909 s Eventos processados: 1.44532e+007 15900.1 (eventos/s)	Tempo de execução: 909 s Eventos processados: 1.44532e+007 15900.1 (eventos/s)

- Simulação através de processamento paralelo:

Relógio interno Simulador	Resultados "Projeto-1"	Resultados "Projeto-2"
1	Tempo de execução: 189 s Eventos processados: 1.44528e+006 7647 (eventos/s)	Tempo de execução: 188 s Eventos processados: 1.44528e+006 7687.68 (eventos/s)
2	Tempo de execução: 381 s Eventos processados: 2.8906e+006 7586.88 (eventos/s)	Tempo de execução: 381 s Eventos processados: 2.8906e+006 7586.88 (eventos/s)
3	Tempo de execução: 582 s Eventos processados: 4.33592e+006 7450.04 (eventos/s)	Tempo de execução: 582 s Eventos processados: 4.33592e+006 7450.04 (eventos/s)
4	Tempo de execução: 765 s Eventos processados: 5.78124e+006 7557.17 (eventos/s)	Tempo de execução: 765 s Eventos processados: 5.78124e+006 7557.17 (eventos/s)
5	Tempo de execução: 954 s Eventos processados: 7.22655e+006 7575 (eventos/s)	Tempo de execução: 955 s Eventos processados: 7.22655e+006 7567.07 (eventos/s)
10	Tempo de execução: 1928 s Eventos processados: 1.44532e+007 7496.45 (eventos/s)	Tempo de execução: 1927 s Eventos processados: 1.44532e+007 7500.34 (eventos/s)

- Simulação através de processamento distribuído:

Relógio interno Simulador	Resultados obtidos na simulação dos Projetos
1	Tempo de execução: 91 s
2	Tempo de execução: 183 s
3	Tempo de execução: 276 s
4	Tempo de execução: 368 s
5	Tempo de execução: 455 s
10	Tempo de execução: 909 s

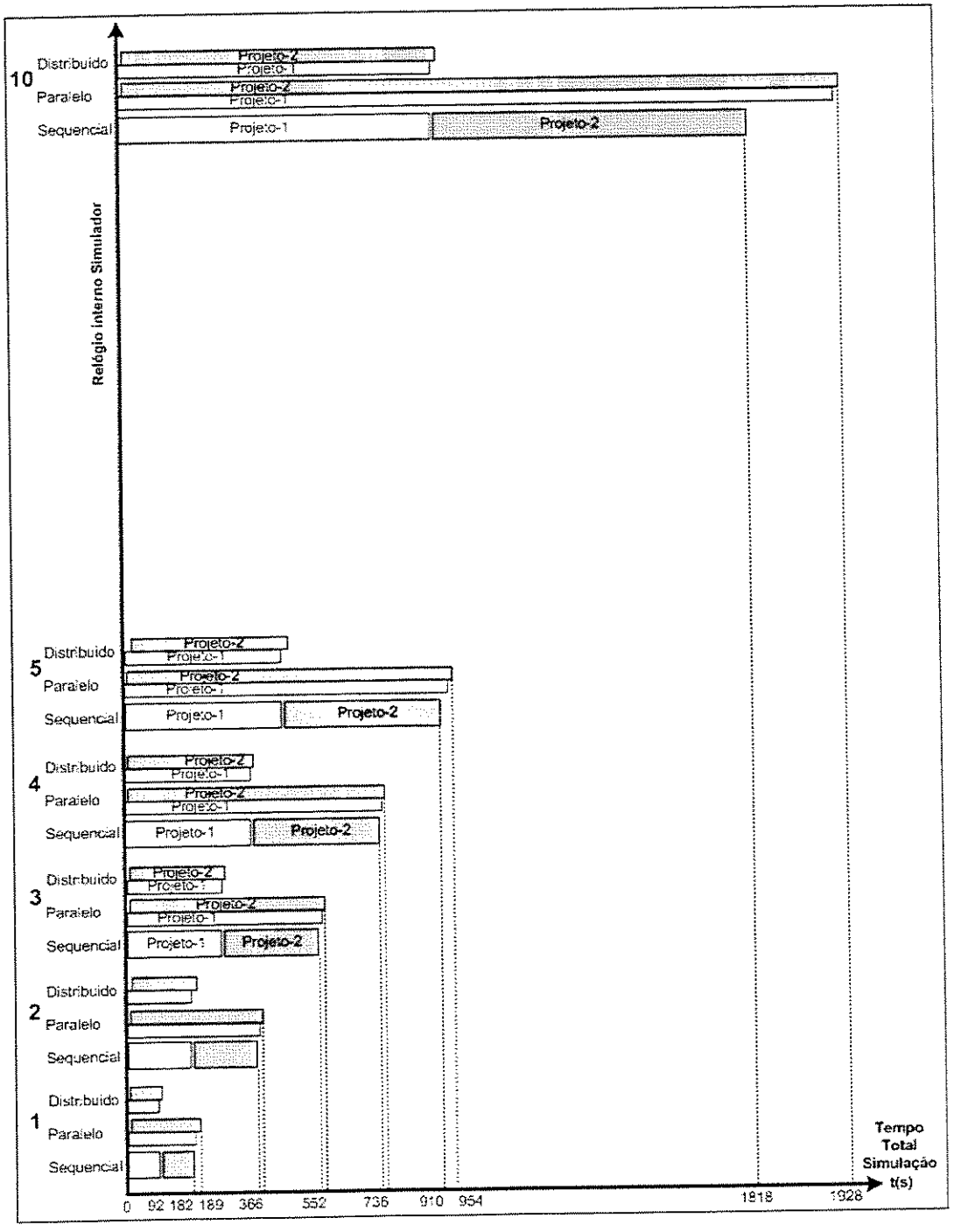

Figura 4-15: Gráfico comparativo da Simulação Seqüencial x Simulação Distribuída

Baseado nos requisitos propostos para este projeto, evidenciou-se uma significativa redução no tempo total gasto para alcançar o resultado utilizando a simulação através de processamento distribuído com dois computadores quando comparado à simulação através do processamento seqüencial onde apenas um computador foi utilizado, conforme ilustrado na Figura 4-15.

4.4 Conclusão

Os modelos de simulação distribuídos precisam ser elaborados numa nova filosofia de modelagem. Nesta nova filosofia, não basta particionar um projeto em múltiplos sub-projetos de forma a permitir a simulação distribuída, porque os modelos matemáticos precisam ser repensados para atender requisitos de desempenho tais como problemas relacionados à latência de comunicação e aos tempos de simulação (local e no contexto da simulação distribuída), e precisam ser projetados para atender aos requisitos de um modelo de simulação distribuído a fim de minimizar o tempo da simulação. Um plano para avaliar quais modelos podem ser distribuídos precisa ser elaborado em virtude da função conceitual do modelo (papel a ser desempenhado) e de cálculos matemáticos (dependência de outros cálculos). Neste capítulo, os resultados alcançados no trabalho comprovam o objetivo inicialmente proposto. As considerações finais são apresentadas no capítulo a seguir.

(Página Intencionalmente em Branco)

Capítulo 5

Conclusões e Considerações Finais

5.1 Conclusões

O desenvolvimento da arquitetura proposta exigiu não só a busca de novos conhecimentos, mas também a absorção e integração de novas tecnologias. A interoperabilidade entre projetos de simulação executando no ambiente de simulação Hydragyrum, através de uma extensão ao ambiente, implementada neste trabalho, foi alcançada com sucesso. Como resultado deste trabalho, foi gerado um *CD* contendo: o componente de *software* (distrib*)* que permite a troca de dados entre projetos de simulação, implementado em linguagem C^{++} (Microsoft Visual C++ 6.0); o módulo de *Software* para operar como servidor de aplicação, implementado em linguagem Java; o manual básico de utilização do usuário; o ambiente de simulação *Hydragyrum cliente/servidor;* uma ferramenta para a edição de modelos de simulação distribuídos para executar como aplicativo; e um exemplo trivial de utilização da arquitetura *cliente/servidor* para o ambiente de simulação *Hydragyrum*.

5.2 Dificuldades Encontradas

A maior dificuldade encontrada no trabalho foi desenvolver uma solução para possibilitar o envio de mensagens distribuídas, a fim de causar o mínimo de alterações no ambiente *Hydragyrum*. O problema consistiu em determinar uma forma em que as *threads* (paralelismo de processos em software) já implementadas no ambiente *Hydragyrum* não entrassem em conflito (*threads* bloqueando outras *threads* e aspectos de compilação do projeto) com as *threads* utilizadas nas bibliotecas dinâmicas do *ORB*

Orbacus. Outro aspecto importante a ressaltar foi a dificuldade na compilação de todos os códigos-fontes do trabalho.

5.3 Projeções Futuras

Como continuidade do trabalho apresentado, algumas possibilidades são vislumbradas:

- Criar mecanismos visuais para realizar a administração pelo usuário moderador: gerenciamento de usuários cadastrados; obtenção de relatórios estatísticos e gráficos visuais dos resultados obtidos nas simulações; agendamento de simulações; notificação eletrônica via email dos resultados da simulação;
- Criar mecanismos visuais para construir modelos de rede fazendo o uso de um formato padrão para troca de mensagens (*XML*) e um protocolo padrão para o envio e recebimento destas mensagens (*SOAP*), de forma que a interoperabilidade entre as mensagens empacotadas e transportadas usando *HTTP* seja mantida, com o propósito de abrir novos leques para a distribuição através da *WEB*.

A modelagem de problemas que requerem simulação distribuída são possibilidades de trabalhos futuros, a fim de extrair dados provenientes de testes de carga/*stress* e testes de desempenho. Na *Figura 5-1*, é ilustrado uma proposta de trabalho futuro para ser realizada sob o ambiente de simulação *Hydragyrum cliente/servidor*. Esta proposta consiste basicamente na simulação de dois laboratórios de pesquisa denominados "Sala 2" e "Sala 3". Cada laboratório é constituído de computadores e dispositivos móveis tais como celular ou PDA (*Personal Digital Assistant*) que podem estar realizando ações distintas. Os laboratórios comunicam-se através de um equipamento de *Hardware* denominado *Backbone (presente na "Sala 1")* que é responsável por comutar as mensagens entre os laboratórios. No escopo do ambiente de simulação *Hydragyrum cliente/servidor*, cada laboratório poderia ser representado por um projeto

distinto de simulação. A fim de prover maior processamento computacional através do uso otimizado dos recursos computacionais de *hardware* existentes, o uso de *"Grid Computing"* através da plataforma de *software* "*Sun Grid Engine*" [11] pode ser uma interessante estratégia de solução computacional.

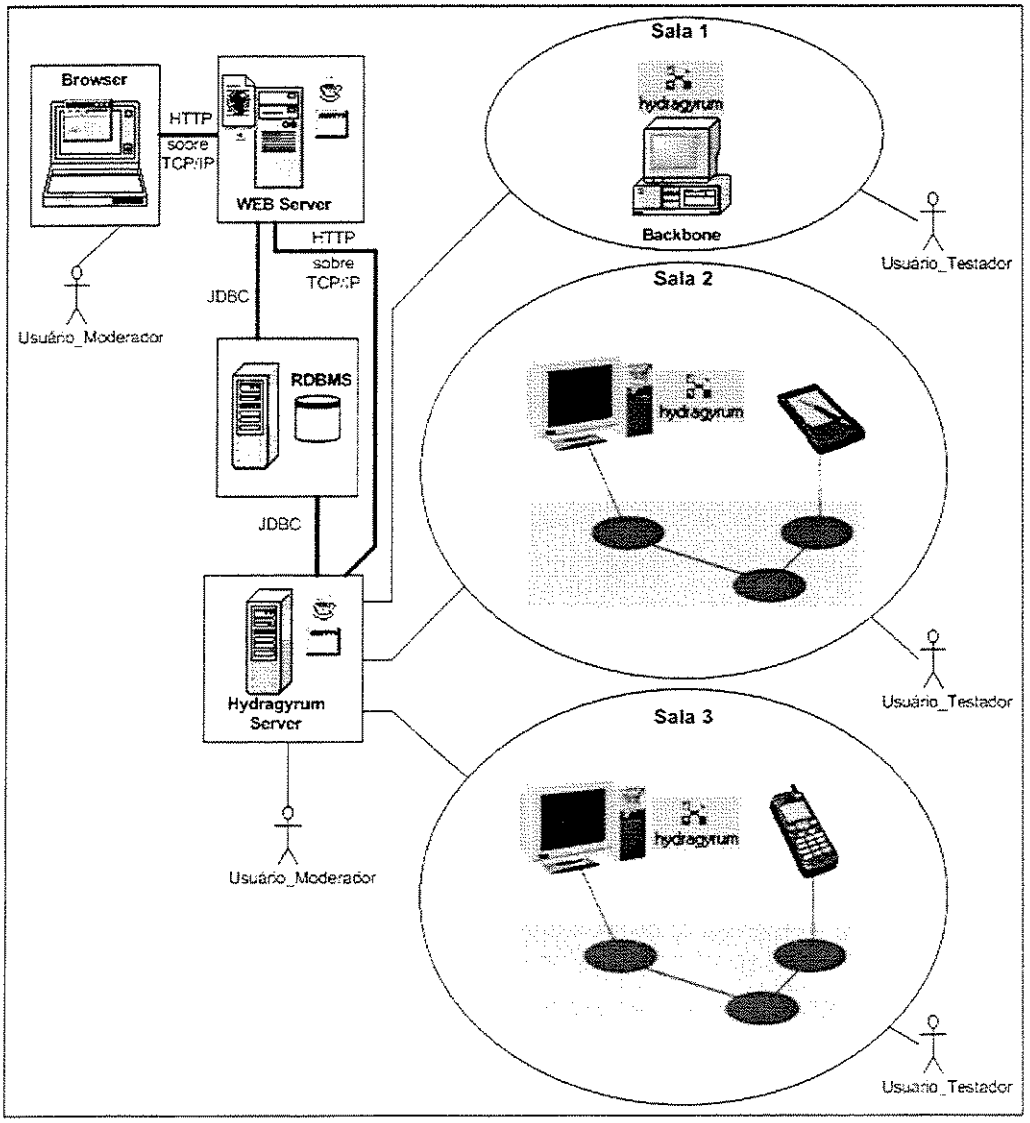

Figura 5-1: Proposta de trabalho futuro – simulação de laboratórios

(Página Intencionalmente em Branco)

Referências Bibliográficas

[1] Klein, J.. **SIMNT - Uma ferramenta para a simulação de sistemas de comunicação**. Tese de Mestrado. Faculdade de Engenharia Elétrica e de Computação. UNICAMP, Agosto, 1995.

[2] Klein, J.. **Desenvolvimento de uma ferramenta CAD/CAM para Simulação, Projeto e Ensino de Sistemas de Comunicação**. Tese de Doutorado. Faculdade de Engenharia Elétrica e de Computação. UNICAMP, Março, 1999.

[3] Alberti, A. M.. **SimATM: Um Ambiente para a Simulação de Redes ATM**. Tese de Mestrado. Faculdade de Engenharia Elétrica e de Computação. UNICAMP, Abril, 1998.

[4] Andrade Neto, Ernesto Luiz. **Ambiente de simulação de redes a eventos discretos**. Tese de Doutorado. Universidade Estadual de Campinas - UNICAMP, Faculdade de Engenharia Elétrica e de Computação, 2001.

[5] __. Referência oficial para *Java*. Disponível por WWW em http://java.sun.com/. 21 abr 2002.

[6] Natan, Ron Bem. **CORBA**. A guide to common object request broker architecture. EUA: McGraw-Hill, 1995.

[7] __. **The ORBacus home page**. Disponível por WWW em http://www.ooc.com/ob/. 10 nov 2002.

[8] __. **The Common Object Request Broker Architecture**: architecture and specification, revision 2.3.1. Disponível por WWW em http://www.omg.org/cgi-bin/doc?formal/99-10-07. 15 fev 2003.

[9] __. **IDL/Java Language Mapping**. Disponível por WWW em http://www.omg.org/. Documento da OMG de 01 mar 1997.

[10] Mowbray, Thomas J., Malveau, Raphael C.. **Corba Design Patterns**. EUA: John Wiley, 1997.

[11] __. Referência oficial para *Sun Grid Computing*. Disponível por WWW em http://www.sun.com/software/gridware/. 16 dez 2003.

[12] Fowler, Martin. **Analysis Patterns: Reusable Object Models**. EUA: Addison Wesley, 1997.

[13] Buschmann, Frank, et al. **Pattern-Oriented Software Architecture**. A system of patterns. New York: John Wiley and Sons, Inc., 1996.

[14] Gamma, Erich, Helm, Richard, Johnson, Ralph, Vlissides, John. **Design Patterns**. Elements of reusable object-oriented software. EUA: Addison Wesley, 1994.

[15] Overeinder, B., Hertzberger, B., Sloot, P. **Parallel Discrete Event Simulation**. EUA: Addison Wesley, 1991.

[16] Thondugulam, N. V. **Unsynchronized Parallel Discrete Event Simulation**. Division of Research and Advanced Studies of the University of Cincinnati, 1998.

[17] Thondugulam, N. V, Rao, D. M., Radhaknshnan, R., Wilsey, P. **Relaxing casual constraints in PDES**. In Proceedings of the 13th International Parallel Processing Symposium, 1999, 696-700.

[18] Fujimoto, R. M. **Parallel Discrete Event Simulation**. Communications of the ACM, v. 33, n. 10, 1990, 31-53.

[19] Vee, V.-Y., Hsu, W.-J.. **Parallel Discrete Event Simulation: A Survey**. *Center for Advanced Information Systems, SAS – Nanyang Technological University*, 1999.

(Página Intencionalmente em Branco)

Bibliografia Suplementar

ABNT. **NBR 6023**: **informação e documentação: elaboração: referências**. Rio de Janeiro, 2002. 24 p.

ABNT. **NBR 14724**: **informação e documentação: trabalhos acadêmicos: apresentação**. Rio de Janeiro, 2002. 6 p.

PATEL, Pratik; MOSS, Karl. **Java database programming with JDBC**. Scottsdale: Coriolis Group, 1996.

FLANAGAN, David. **Java in a Nutshell**. EUA: O' Reilly & Associates, Inc., 1996.

BERSON, Alex. **Client / Server architecture**. 2º ed. EUA, McGraw-Hill, 1996.

ORFALI, Robert. **Client / Server programming with Java / CORBA**. EUA, McGraw-Hill, 1998.

MOWBRAY, Thomas J., ZAHAVI, Ron. **The essential CORBA**. EUA, John Wiley & Sons, Inc., 1995.

SELIC, Bran, GULLEKSON, Garth, WARD, Paul T.. **Real-Time object-oriented modeling**. EUA, John Wiley & Sons, Inc., 1994.

BOOCK, Grady, RUMBAUGH, James, JACOBSON, Ivar. **UML**. Brasil, Editora Campus, 2000.

RUMBAUGH, James, et al. **Object-Oriented modeling and design**. EUA, Prentice-Hall, Inc., 1991.

LEE, Richard C., TEPFENHART, William M. **UML and C++**. EUA, Prentice-Hall, Inc., 1997.

CHANG, Dan, HARKEY, Dan. **Client / Server data access with Java and XML**. EUA, Addison Wesley, 1999.

ALUR, Deepak, CRUPI, John, MALKS, Dan. **Core J2EE patterns**. EUA, Sun Microsystems, Inc., 2001.

ALEXANDER, Chistopher. **The timeless way of building**. New York, Oxford University Press, 1979.

ALEXANDER, Chistopher, et al. **A pattern language**. New York, Oxford University Press, 1977.

BOOCH, Grady, RUMBAUGH, James, JACOBSON, Ivar. **The unified modeling language user guide**. EUA, Wiley, 1998.

BRITTON, Chris. **IT architectures and middleware**. EUA, Addison Wesley, 2001.

Metodologia Rational (RUP) para processos de *software.* Disponível por WWW em http://www.rational.com/. 21 abr 2002.

Fundamentos de *Sun Grid Computing*. Sun Microsystems, Inc., EUA, mai 2002.

Referência oficial para o banco de dados *MySQL*. Disponível por WWW em http://www.mysql.com/. 2 mai 2002.

Glossário

Análise. A parte do processo de desenvolvimento de *software* cuja principal finalidade é formular um modelo do domínio do problema. A análise enfoca o que fazer, enquanto o *design* enfoca como fazê-lo.

Analista. Membro da equipe do projeto, responsável por identificar e interpretar as necessidades dos principais envolvidos, comunicando-as à equipe.

API (interface de programação de aplicativos). Uma interface de *software* que permite que aplicativos se comuniquem entre si. Uma *API* é o conjunto de sentenças e construções de linguagem de programação que podem ser codificadas em um programa de aplicativo a fim de obter as funções e os serviços específicos fornecidos por um programa de serviços ou sistema operacional básico.

Arquitetura. O conceito de nível mais alto de um sistema em seu ambiente (IEEE). A arquitetura de um sistema de *software* (em determinado momento) é sua organização ou estrutura das partes significativas interagindo através de interfaces, componentes e subsistemas.

Artefato. Uma informação que é produzida, modificada ou usada por um processo de desenvolvimento de *software*, define uma área de responsabilidade e está sujeita a controle de versão. Um artefato pode ser um documento ou um *software*. Sinônimo: produto.

Assinatura. O nome e os parâmetros de uma característica comportamental (operação, mensagem ou evento). Uma assinatura pode incluir um parâmetro retornado opcional.

Ator. Alguém ou algo fora do sistema que interage com ele.

Biblioteca de classes. Uma coleção de classes.

Caso-de-teste. A definição (geralmente formal) de um conjunto específico de *inputs* de teste, condições de execução e resultados esperados, identificados com a finalidade de avaliar um determinado aspecto de um item de teste-alvo.

Caso-de-uso. Uma descrição de comportamento do sistema em termos de seqüências de ações que um sistema (ou outra entidade) pode executar, interagindo com atores do sistema.

Cenário. Uma sucessão específica de ações que ilustra comportamentos. Um cenário pode ser usado para ilustrar uma interação.

Cenário de teste. Uma instância específica de um *script* de teste que fornece valores de dados específicos e, normalmente, um único caminho de execução.

Classe. Uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, métodos, relacionamentos e semântica. Uma classe pode usar um conjunto de interfaces para especificar coleções de operações que ela fornece para seu ambiente.

Componente. Uma parte substituível e praticamente independente de um sistema que cumpre uma função clara no contexto de uma arquitetura bem definida. Um componente fornece e se adapta à realização física de um conjunto de interfaces, representa uma parte física da implementação de um sistema, incluindo código de *software* (fonte, binário ou executável). Pontos fundamentais que caracterizam componentes são: é uma entidade 'comercializável'; não é uma aplicação completa; é um objeto interoperável e é um objeto passível de extensão.

Container (contêiner). (1) Um objeto que existe para conter outros objetos, e para isso provê operações para ter acesso ou interagir sobre seu conteúdo. Por exemplo: vetôres, conjuntos e dicionários. (2) Um componente que existe para conter outros componentes.

Desenvolvedor. Uma pessoa responsável pelo desenvolvimento da funcionalidade necessária de acordo com os procedimentos e os padrões adotados no projeto. Isso inclui a realização de atividades em qualquer uma das disciplinas de requisitos, análise e design, implementação e teste.

Evento. Uma ocorrência de significância. Um evento tem uma localização no tempo e no espaço e pode ter parâmetros. No contexto do diagrama de estados, um evento é uma ocorrência que pode ativar uma transição de estado.

Fat-Client. Em uma arquitetura *cliente/servidor*, este termo é empregado para caracterizar uma aplicação onde a *"Client Tier"* requer grande quantidade de processamento local do computador do usuário-final, onde a aplicação está instalada para executar o volume de operações.

GUI (interface gráfica de usuário). Um tipo de interface que permite a comunicação dos usuários com o programa através da manipulação de recursos gráficos, em vez da digitação de comandos. Normalmente, a *GUI* inclui uma combinação de elementos gráficos, dispositivos apontadores, barras de menu e outros menus, janelas sobrepostas e ícones.

Grid. Conceitualmente, um *grid* [11] é uma coleção de recursos computacionais conectados através de uma rede, para tratar enormes tarefas da computação pela distribuição do trabalho através dos recursos de processamento.

Grid Computing. Também chamado de computação distribuída ou "distributed computing", o termo é muito amplo, e vários tipos distintos de *grids* são comumente usados na indústria hoje. *Grid computing* pode prover alguns benefícios não disponíveis com os modelos tradicionais da computação:

- Melhor utilização de recursos – *Grid computing* usa recursos distribuídos mais eficientemente e disponibiliza maior poder de uso da computação. Isto pode minimizar o tempo de entrega do resultado (*time-to-market*) ou então incluir

inovações ao resultado ou disponibilizar testes adicionais e simulações a fim de prover qualidade do produto.

- Incremento da produtividade do usuário – é providenciado através do acesso transparente aos recursos e desta forma, o trabalho pode ser completado mais rapidamente.
- Escalabilidade – *Grids* permitem centenas de processadores integrados dentro de um *cluster*. Componentes podem ser atualizados independentemente e recursos adicionais podem ser adicionados de acordo com a necessidade.
- Flexibilidade – *Grid computing* provém capacidade de processamento onde é mais necessário, ajudando da melhor maneira a dinamicamente realizar o balanceamento das cargas de trabalho.

Herança. Mecanismo que torna possível a generalização, através do qual elementos específicos incorporam estrutura e comportamento de elementos gerais relacionados por comportamento.

IDE (ambiente de desenvolvimento integrado). Um *software* composto por um editor, um compilador e um depurador.

Interface. Uma coleção de operações usadas para especificar um serviço de uma classe ou de um componente. Um conjunto nomeado de operações que caracterizam o comportamento de um elemento.

Instância. Um objeto descrito por uma classe. De acordo com uma interpretação rígida do metamodelo um membro individual de um tipo é uma instância e um membro de uma classe é um objeto. Em um uso menos formal é aceitável se referir a um membro de uma classe como um objeto ou uma instância. Instâncias podem ser criadas (instânciadas) ou destruidas (apagadas) em tempo de execução.

Instanciação. É um mecanismo no paradigma orientado a objetos pelo qual pode-se criar instâncias de classes. Estas instâncias (objetos) são os guardiões dos dados que

irão fazer a aplicação/sistema trabalhar. Este mecanismo é um dos veículos que se usa para fazer os modelos dinâmicos.

Layer (camada). Um modo específico de agrupar serviços, todos no mesmo nível de abstração.

Mensagem. Uma comunicação entre objetos que leva informação com a expectativa que resultará em uma atividade. O recebimento de uma mensagem é normalmente considerado um evento.

Mensagem assíncrona. Uma mensagem onde o objeto emissor não tem sua execução interrompida para esperar por resultados.

Mensagem síncrona. Uma mensagem onde o objeto emissor pausa seu processamento para esperar por resultados.

Modelo. (1) Uma abstração semanticamente fechada de um sistema. (2) Desenho ou imagem que representa o que se pretende reproduzir. (3) Aquele a quem se procura imitar nas ações e maneiras.

Módulo. Uma unidade de armazenamento e manipulação do *software*. Os módulos incluem módulos de código-fonte, de código binário e de código executável.

Nó. Um nó é um objeto físico em tempo de execução que representa um recurso computacional, possuindo, geralmente, pelo menos uma memória, bem como, uma capacidade de processo. Objetos em tempo de execução e componentes podem residir em nó.

Objeto. Uma entidade com uma fronteira bem-definida e uma identidade, que encapsula estado e comportamento. Estado é representado por atributos e relacionamentos; o comportamento é representado por operações, métodos e máquinas de estado. Um objeto é uma instância de uma classe. Entre as características fundamentais, objetos suportam herança, poliformismo e encapsulamento.

Processo. (1) Uma linha de controle (por exemplo uma *Thread*) que pode executar simultaneamente com outras linhas de controle (por exemplo um processo de sistema operacional). (2) Conjunto de passos parcialmente ordenados que visam atingir uma meta. Na engenharia de *software*, a meta é criar um produto ou aprimorar um já existente; na engenharia de processos, a meta é desenvolver ou aprimorar um modelo de processo; corresponde a um caso de uso de negócios na engenharia de negócios.

Produto. Os artefatos resultantes do desenvolvimento tais como mídia de *release do software*, códigos e documentação.

Processamento distribuído. O processamento distribuído é um modelo de sistemas ou aplicativos no qual a função e os dados podem ser distribuídos para vários recursos de computação em uma *LAN*.

Projeto. Projetos são realizados por pessoas, restringidos por recursos limitados, planejados, executados e controlados. Um projeto é um esforço temporário empreendido para criar um serviço ou produto. Temporário significa que todo projeto tem começo e fim definidos.

Requisito. (1) Uma característica, uma propriedade ou um comportamento desejado de um sistema. (2) Um requisito descreve uma condição ou capacidade à qual um sistema deve se adaptar, seja ela derivada diretamente de necessidades dos usuários ou declarada em um contrato, um padrão, uma especificação ou outro documento formalmente imposto.

Requisito de software. Uma especificação de um comportamento do sistema que pode ser observado externamente; por exemplo, entradas e saídas do sistema, funções e atributos do sistema ou atributos do ambiente do sistema.

Speed-up. (1) O fator de aceleração (*speed-up*) é uma medida que indica o ganho de velocidade obtido na execução de um programa utilizando vários processadores em

relação a sua execução seqüencial. (2) O fator de aceleração (*speed-up*) mede a razão entre o tempo gasto para execução de um algoritmo ou aplicação em um único processador e o tempo gasto na execução com *"n"* processadores;

$$S(n) = T(1) / T(n)$$

Para uma máquina paralela com *"n"* processadores, *speed-up* ideal seria *"n"* (*speed-up* linear). A eficiência para o *speed-up* do processador calculado, pode ser determinado por:

$$E(n) = S(n) / n$$

SOAP. *Simple Object Access Protocol* é um protocolo, baseado na linguagem *XML,* que permite conexões *RPC* sobre *HTTP* tornando-se fácil alcançar computadores através de um *"firewall"*. Por outro lado, como as informações que chegam ao servidor estão em modo texto, torna-se fácil bloquear chamadas *SOAP* se necessário, pela análise das informações *HTTP* ou sobre o cabeçário *SOAP* de uma mensagem. *SOAP* implementa um padrão de arquitetura *cliente/servidor*. Processos são inicializados pelo cliente e o servidor responde as requisições. O protocolo pode ser utilizado para fins de envio de mensagens distribuídas através de redes distintas de computadores.

Subsistema (*subsystem*). Um sistema subordinado dentro de um sistema maior. Na *UML* um subsistema é modelado como um pacote de componentes.

Stub. Um componente que contém funcionalidade para fins de teste. Um *stub* é "fictício", simplesmente retornando alguns valores predefinidos, ou está "simulando" um comportamento mais complexo.

Tempo (*time*). Um valor que representa um momento absoluto ou relativo no tempo.

Tempo de execução (*run time*). O período de tempo durante o qual um programa de computador é executado.

Thin-Client. Em uma arquitetura cliente/servidor, este termo é empregado para caracterizar uma aplicação, onde a *"Client Tier"* foi projetada para ser especificamente pequena no processamento do volume de dados, que ocorrem no *servidor*.

Tier. Uma camada lógica de um sistema distribuído, onde as partes de um programa podem ser separadas em várias *tier´s*, cada podendo estar localizada em um computador diferente em uma rede. O modelo de aplicação denominado *three-tier* provavelmente é o modo mais comum de organizar um programa em uma rede. O particionamento de um projeto, em várias *"Tier",* permite a escolha de tecnologias especializadas ditadas para os requisitos do negócio da aplicação de *software* a ser desenvolvida. Cada *"Tier"* representa um nível de abstração do projeto, permitindo desenvolvedores a focar sua atenção em um nível sem ter que se preocupar com detalhes de outros níveis.

UML (Linguagem Unificada de Modelagem). É uma linguagem padrão destinada a visualizar, especificar, construir e documentar os artefatos de um sistema fundamentado em *software*. Cada um dos pontos de vista (cenário) traz contribuições próprias ao projeto e observa o sistema de maneira distinta em momentos diferentes ao longo do desenvolvimento do projeto.

XML. A Linguagem de Marcação Estendida (*eXtensible Markup Language*) é um subconjunto da *SGML* (*Standard Generalized Markup Language*, ou Linguagem de Marcação Padrão Generalizada) permite que uma marcação especifica seja criada para especificar idéias e compartilhá-las na rede. Sua padronização é regulado pelo WC3 (o World Wide Web Consortium).

(Página Intencionalmente em Branco)

Apêndice A

Diagramas de Classes Simplificado do Ambiente Hydragyrum

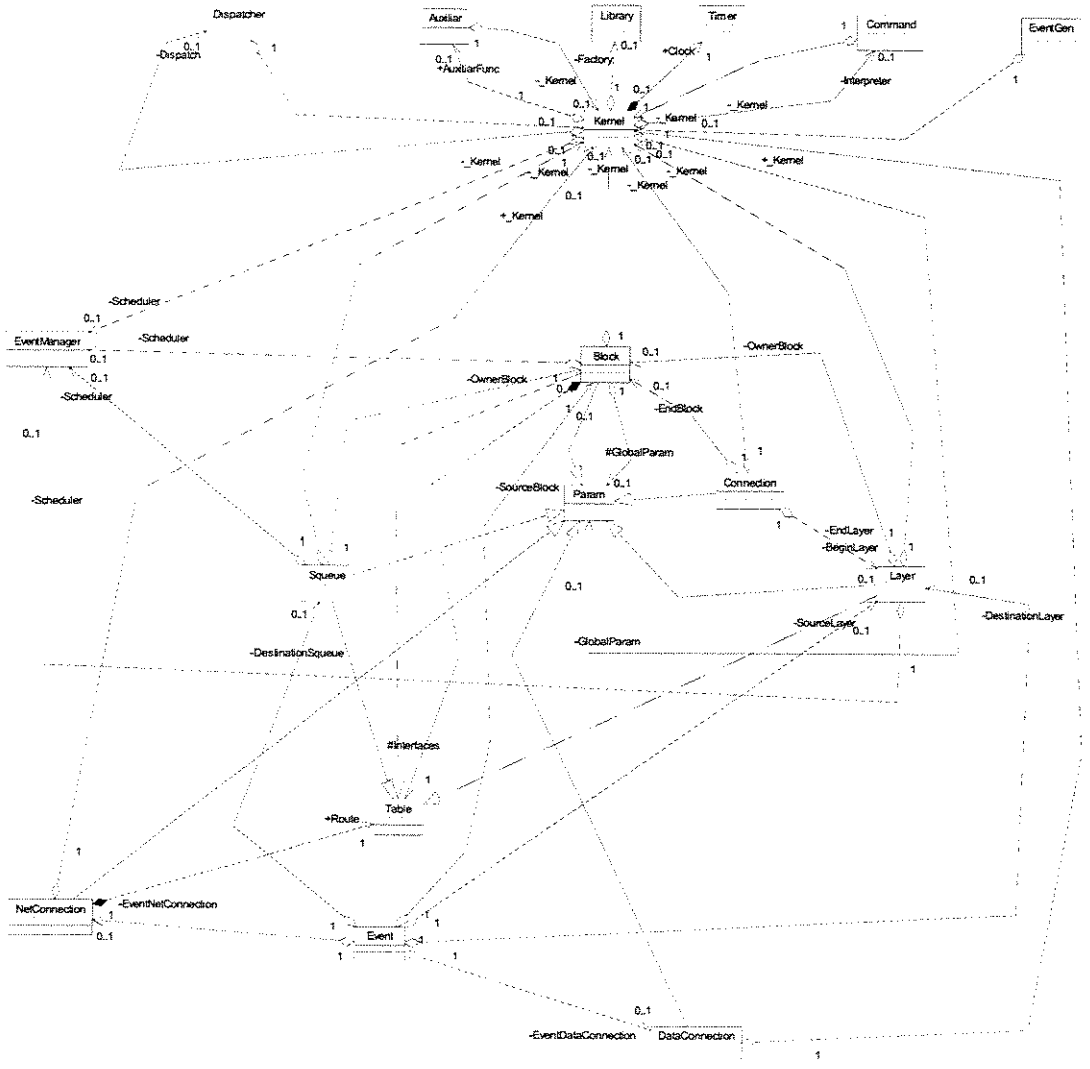

Figura AA-1: Cenário de relacionamento para a classe "Kernel"
Diagrama de Classes - Notação Gráfica UML

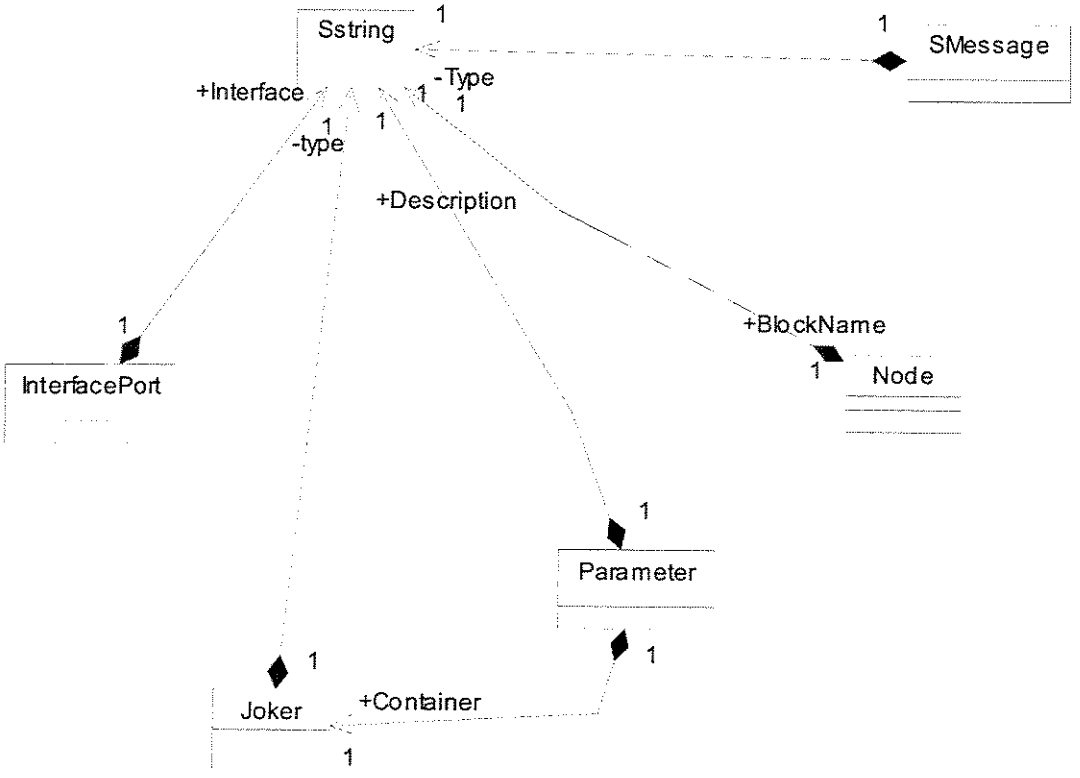

Figura AA-2: Cenário de relacionamento para a classe "Sstring"
Diagrama de Classes - Notação Gráfica UML

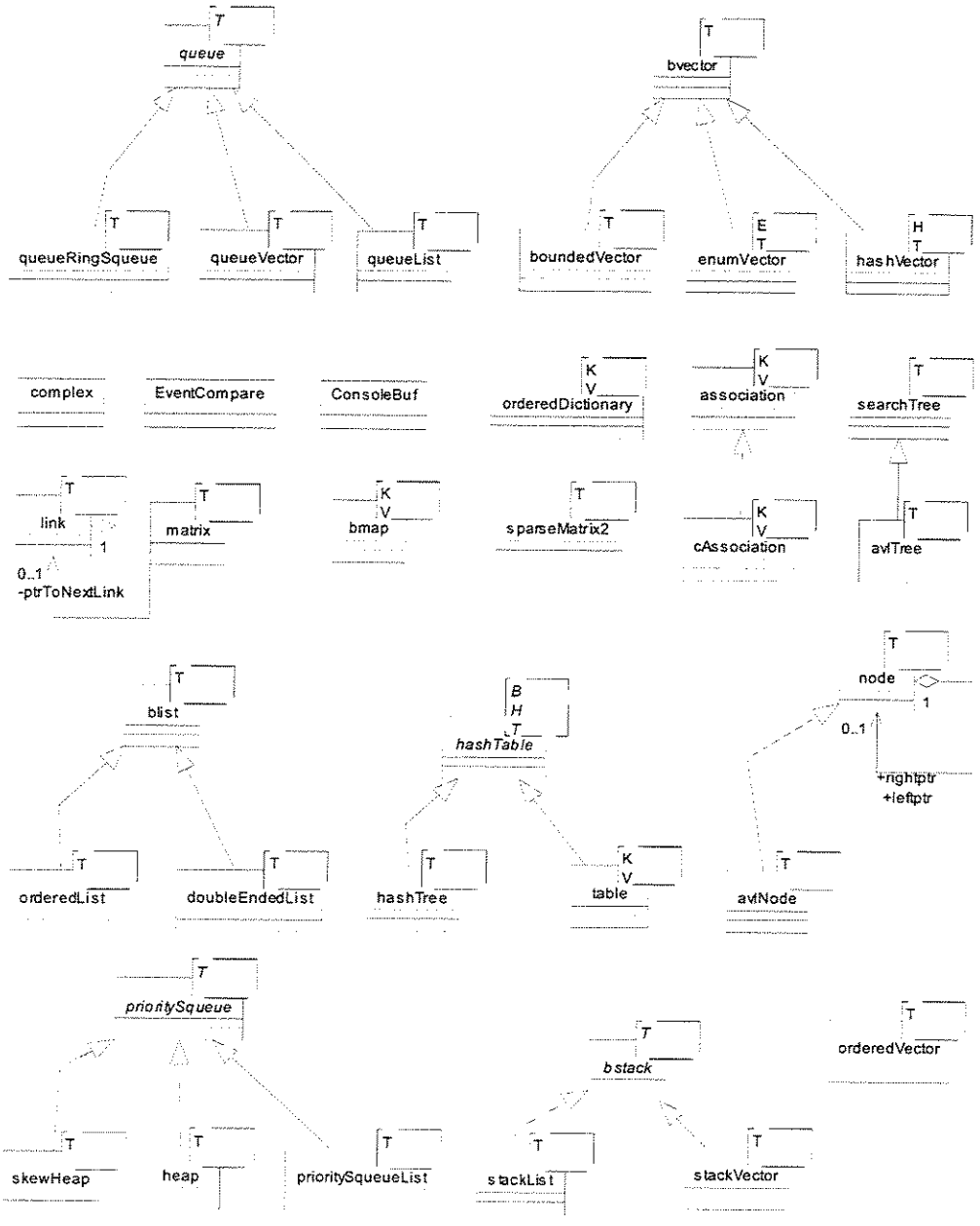

Figura AA-3: Cenário 1 para classes diversas
Diagrama de Classes - Notação Gráfica UML

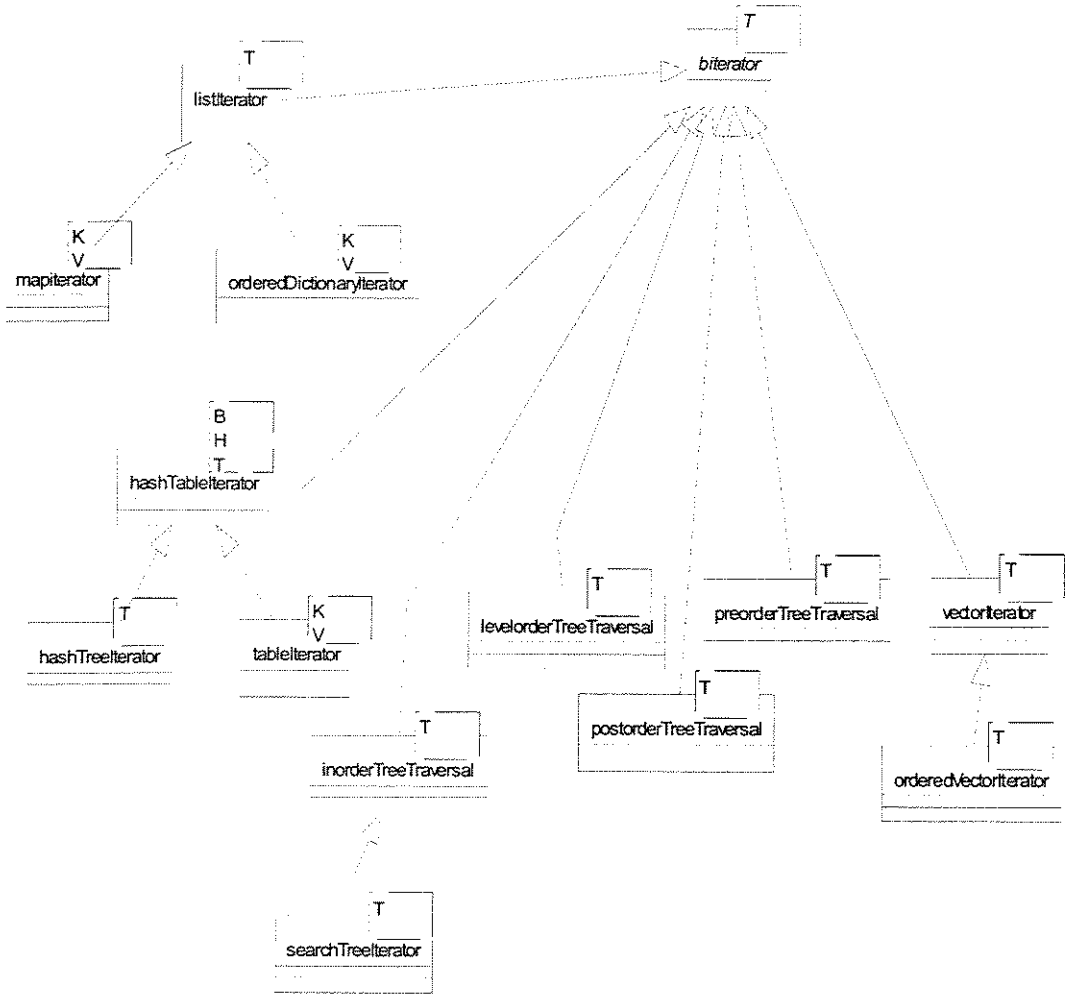

Figura AA-4: Cenário 2 para classes diversas

Diagrama de Classes - Notação Gráfica UML

Apêndice B

Métricas de Software coletadas do Ambiente Hydragyrum

Neste apêndice, é apresentado a métrica de linhas-de-código coletada para o Ambiente de Simulação *Hydragyrum*.

Nome do Arquivo	Linhas (código) (SLOC)	Linhas (comentários)	Linhas (espaços em branco) (WLOC)	Total (LOC)
auxiliar.cpp	189	3	61	253
auxiliar.h	28	1	17	46
bhash.h	213	103	127	443
biterator.h	12	19	10	41
blist.h	437	146	192	775
block.cpp	397	9	272	678
block.h	338	108	277	723
blockb.cpp	484	10	279	773
blockc.cpp	1084	7	476	1567
blockmcr.cpp	422	1	5	428
blockmcr.h	35	0	5	40
bpriorque.h	229	104	145	478
bqueue.h	168	56	43	267
bstack.h	121	50	72	243
btable.h	417	163	232	812
btree.h	851	262	379	1492
bvector.h	448	194	196	838
command.cpp	1018	7	321	1346
command.h	45	10	46	101
commandb.cpp	874	14	317	1205

connection.cpp	506	1	333	840
connection.h	123	19	102	244
dataconnec.cpp	554	6	297	857
dataconnec.h	111	22	91	224
dispatcher.cpp	47	0	21	68
dispatcher.h	18	1	11	30
event.cpp	444	14	428	886
event.h	109	14	102	225
eventgen.h	28	2	24	54
eventmng.cpp	92	2	57	151
eventmng.h	37	1	28	66
joker.cpp	1251	13	63	1327
joker.h	117	34	36	187
kernel.h	333	135	247	715
kernela.cpp	571	37	374	982
kernelb.cpp	1128	61	597	1786
kernelc.cpp	582	96	292	970
kernelmcr.cpp	391	1	5	397
kernelmcr.h	29	0	5	34
layer.cpp	521	41	281	843
layer.h	292	71	238	601
layerb.cpp	931	12	557	1500
layermcr.cpp	423	1	5	429
layermcr.h	35	0	3	38
library.cpp	39	6	29	74
library.h	20	2	12	34
netconnec.cpp	575	11	332	918
netconnec.h	105	27	89	221
param.cpp	488	35	105	628
param.h	133	39	43	215

parameter.h	20	3	11	34
sim_complex.cpp	276	3	73	352
sim_complex.h	66	0	18	84
sim_console.h	29	22	10	61
sim_iostream.cpp	38	20	36	94
sim_iostream.h	34	15	26	75
simdefs.h	26	7	15	48
simscl.cpp	54	2	35	91
smessage.cpp	11	1	7	19
smessage.h	28	1	15	44
squeue.cpp	927	32	536	1495
squeue.h	235	65	182	482
squeuemcr.cpp	423	1	5	429
squeuemcr.h	35	0	3	38
sstring.cpp	353	34	80	467
sstring.h	73	5	31	109
stimer.cpp	44	0	35	79
stimer.h	34	1	22	57
table.cpp	582	56	108	746
table.h	110	48	39	197
Total: 70 arquivos	**21241**	**2287**	**9566**	**33094**

Apêndice C

Listagem dos códigos-fonte exemplo para um projeto Hydragyrum

Para facilitar o reaproveitamento de modelos por diversos projetos no contexto distribuído, uma organização física dos diretórios é importante, conforme ilustrado em *Figura AC-1*, *Figura AC-2*, *Figura AC-3*. A pasta denominada "sample1" define a raiz onde os projetos estarão sendo armazenados e caracteriza o nome geral dos projetos interoperáveis. A pasta denominada "project_1" armazena o primeiro projeto, a pasta denominada "project_2" armazena o segundo projeto e a pasta denominada "project_3" armazena o terceiro projeto que é o relacionamento do primeiro com o segundo. Internamente ao primeiro e segundo projetos, contém uma pasta denominada "common" que é responsável por conter os elementos que serão comuns somente ao projeto internamente. Tanto o primeiro quanto o segundo projeto, podem conter diversos modelos que são armazenados, cada um em sua pasta internamente ao projeto, iniciando o nome da pasta por "model_" e mais o nome dado ao modelo. A pasta denominada "common" no terceiro projeto armazena todos os elementos (estruturas de dados e classes) que são comuns ao primeiro e segundo projetos para a realização de troca de mensagens entre projetos.

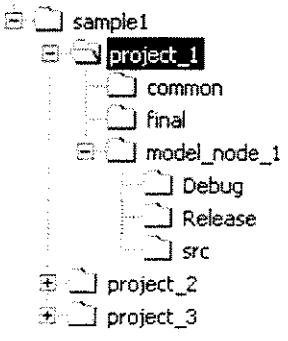

Figura AC-1: Hierarquia Projeto-1

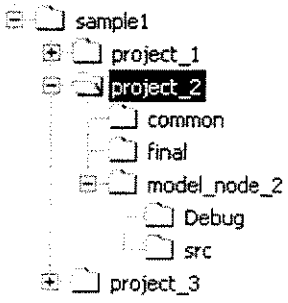

Figura AC-2: Hierarquia Projeto-2

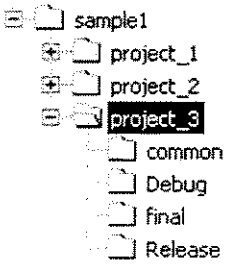

Figura AC-3: Hierarquia Projeto-3

Códigos-Fonte para o Projeto-1

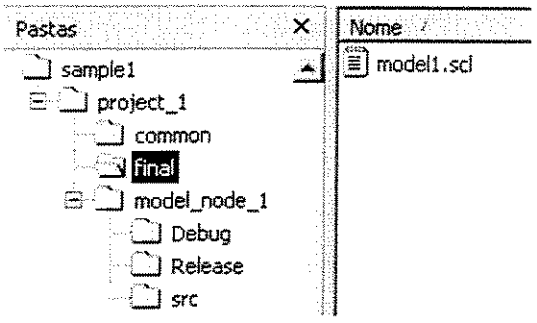

Figura AC-4: Artefatos do Projeto-1

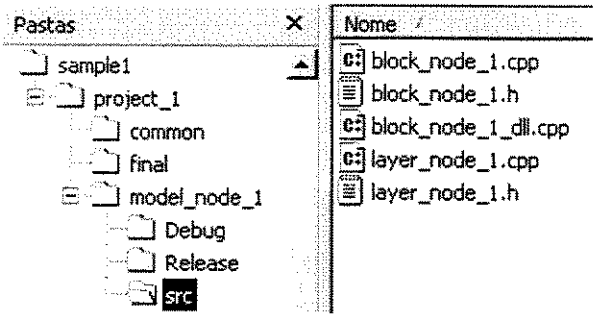

Figura AC-5: Artefatos do Projeto-1

model1.scl

```
block "BLOCK_NODE_1" model_node_1.dll
```

block_node_1.cpp

```
/* ------------------------------------------------------------------------
   BLOCK_NODE_1.CPP
   ------------------------------------------------------------------------
*/

#include "block_node_1.h"

int BLOCK_NODE_1::Setup()
{
  // Set the model name that will be shown in ther GUI
  Sstring N1("BLOCK_NODE_1");
  SetModel(N1);

  // Set the general information about the model
  Author      = "Ramme";
  Version     = "1.0";
  Date        = "28/06/2004";
  Copyright   = "MC21-DECOM-FEEC-UNICAMP";
  Description = "Modelo 1.";

  // Create the layers that will compose the model
  LAYER_NODE_1  * PLNMSC = new LAYER_NODE_1;

  // Seta camadas
  Sstring N2("LAYER_NODE_1");
  Sstring N3("LAYER_NODE_1");
  PLNMSC->SetName(N2);
  PLNMSC->SetType(N3);

  // Add the layers to the model
  AddLayer(PLNMSC);
```

```
  // Definicao das coneccoes permitidas:
  // tipo_layer_neste_bloco, tipo_layer_permitido_para_conectar_neste_bloco,
  // tipo_coneccao_resultante_entre_2layers
  AddInterface("LAYER_NODE_1","LAYER_NODE_2","EV2");

  // Configure the GUI appearence of the model
  // Configure the model block size in the GUI
  ConfigureBlock(54,54,"BORDER","BLACK",1); // BORDER // NOBORDER

  // Configure the ports that will be shown in the GUI
  // associate a layer to each port and choose the port look

  // ports in the right side of the block
  //      | layer name     | layer position     |color|size|type |
  AddInterfacePort("LAYER_NODE_1","RIGHT","GREEN",0.15,"OUTPUT_ARROW");
  //Tipo de seta: OUTPUT_ARROW,AREA
  AddInterfacePort("LAYER_NODE_1","RIGHT","RED",0.15,"INPUT_ARROW");

  // same ports repeated in the left side of the block
  //AddInterfacePort("LAYER_NODE_1","LEFT","RED",0.15,"INPUT_ARROW");
  //AddInterfacePort("LAYER_NODE_1","LEFT","GREEN",0.15,"OUTPUT_ARROW");

  return 0;
}

// -------------------- Metodos da Interface Bloco ---------------------------------
// ---------------------------------------------------------
int BLOCK_NODE_1::OnChangeParameter(Sstring ParamName)
{
  MESSAGE<<"-01-> OnChangeParameter "<<GetName()<<" Parameter "<<ParamName<<endl;
  return 0;
}

// ---------------------------------------------------------
int BLOCK_NODE_1::OnStop(Sstring Stop, double Time, int StopCode)
{
  MESSAGE<<"-04-> OnStop "<<GetName()<<endl;
  return 0;
}

// ---------------------------------------------------------
int BLOCK_NODE_1::Initiate()
{
  MESSAGE<<"-02-> Initiate "<<GetName()<<endl;
  return 0;
}

// ---------------------------------------------------------
int BLOCK_NODE_1::Run( Event *_Event )
{
  MESSAGE<<endl<<setprecision(16)<<"("<<_Event->GetEventTime()<<")"<<endl;
  MESSAGE<<"-03-> Running event ..................................................... "
         <<_Event->GetEventType()<<" at "<<GetName()<<endl;
  return 0;
}

// ---------------------------------------------------------
int BLOCK_NODE_1::Post( Event * _Event )
{
  MESSAGE<<"-MY-> Post "<<GetName()<<endl;
  return 0;
}
```

```
// ------------------------------------------------------
int BLOCK_NODE_1::PosProcessing()
{
  MESSAGE<<"-05-> Pos Processing "<<GetName()<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_1::Terminate()
{
  MESSAGE<<"-06-> Terminate "<<GetName()<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_1::OnConnect(Sstring ConnecName)
{
  MESSAGE<<"-07-> OnConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_1::OnDisconnect(Sstring ConnecName)
{
  MESSAGE<<"-08-> OnDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_1::OnNetworkConnect(Sstring ConnecName)
{
  MESSAGE<<"-09-> OnNetworkConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_1::OnNetworkDisconnect(Sstring ConnecName)
{
  MESSAGE<<"-10-> OnNetworkDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_1::OnDataConnect(Sstring ConnecName)
{
  MESSAGE<<"-11-> OnDataConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_1::OnDataDisconnect(Sstring ConnecName)
{
  MESSAGE<<"-12-> OnDataDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ============================== end! ==================================
```

block_node_1.h

```
#ifndef __BLOCK_NODE_1_H
#define __BLOCK_NODE_1_H

#ifndef  __BLOCK_H
#include "block.h"
#endif

#ifndef  __LAYER_NODE_1_H
#include "layer_node_1.h"
#endif

class BLOCK_NODE_1 : public Block {

public:

   // Interface
   int Setup();
   int OnChangeParameter(Sstring ParamName);
   int OnStop(Sstring Stop, double Time, int StopCode);
   int Initiate();
   int Run( Event * _Event );
   int Post( Event * _Event );
   int PosProcessing();
   int Terminate();
   int OnConnect(Sstring ConnecName);
   int OnDisconnect(Sstring ConnecName);
   int OnNetworkConnect(Sstring ConnecName);
   int OnNetworkDisconnect(Sstring ConnecName);
   int OnDataConnect(Sstring ConnecName);
   int OnDataDisconnect(Sstring ConnecName);

};

//Block * _export GetModel10();
__declspec(dllexport)  Block * GetModel10();

#endif
```

block_node_1_dll.cpp

```
/* ------------------------------------------------------------------------
   BLOCK_NODE_1_DLL.CPP
   Criacao da DLL do Bloco.
   ------------------------------------------------------------------------
*/

#include "block_node_1.h"

//Block * _export GetModel10()
__declspec(dllexport) Block * GetModel10()
{
      return new BLOCK_NODE_1;
}

// ============================== end! ==================================
```

layer_node_1.cpp

```
/* ----------------------------------------------------------------------
   LAYER_NODE_1.CPP
   ----------------------------------------------------------------------
*/

// Exporta implementação dos métodos do modelo para a DLL. Não mexa.
//#define _EXPORT __declspec(dllexport)


#ifndef __LAYER_NODE_1_H
#include "layer_node_1.h"
#endif

#ifndef __PACKET_H
#include "packet.h"
#endif

#ifndef __SMESSAGE_H
#include "SMessage.h"
#endif

#include <Distrib.h>


// ----------------------------------------------------------------------------
// Nesta função o desenvolvedor de modelos deve implementar a:
//     - Configuração dos valores default das variáveis de manipulação dos
//   parâmetros do modelo.
// - Configuração dos valores default das variáveis auxiliares do modelo.
//     - Gravação dos valores default dos parâmetros do modelo.
int LAYER_NODE_1::Setup()
{
  // Mostra na tela do simulador que foi executada a função Setup. Não mexa.
  MESSAGE<<"-->  Setup "<<GetName()<<endl;
  Time = 0;
  return 0;
}

// ----------------------------------------------------------------------------
  // Esta função é executada toda vez que um parâmetro é modificado em algum
  // modelo ou no kernel do simulador. Permite a camada responder a
  // modificações nos seus parâmetros e a modificações nos parâmetros
  // do kernel e de outros modelos. Não é obrigatório implementar esta função.
int LAYER_NODE_1::OnChangeParameter(Sstring ParamName)
{
  // Mostra na tela do simulador que foi executada a função OnChangeParameter.
  MESSAGE<<"--> OnChangeParameter "<<GetName()<<endl;
  return 0;
}

// ----------------------------------------------------------------------------
  // Esta função permite a camada responder a uma parada de simulação.
  // Não é obrigatório implementar esta função.
int LAYER_NODE_1::OnStop(Sstring Stop, double Time, int StopCode)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnStop "<<GetName()<<endl;
  return 0;
}
```

```
// ------------------------------------------------------------------------------
int LAYER_NODE_1::Post( Event * _Event )
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> Post "<<GetName()<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Nesta função o desenvolvedor de modelos deve implementar a leitura
// dos valores dos parâmetros da camada e dos parâmetros globais.
// Carrega variáveis locais para a manipulação de parâmetros com o valor
// atual destes. Isto deve ser feito porque o valor do parâmetro pode ter
// sido modificado desde o Setup, por exemplo pelo usuário na Interface Gráfica.
int LAYER_NODE_1::Initiate()
{
  // Mostra na tela do simulador que foi executada a função Initiate.
  MESSAGE<<"--> Initiate "<<GetName()<<endl;
  GetParam("Nome_do_Parametro_1",Parametro_1);
  CreateLoopEvent(Time,"LOOP_EVENT");
  return 0;
}

// ------------------------------------------------------------------------------
int LAYER_NODE_1::Run( Event * _Event )
{
  // Mostra na tela do simulador que foi executada a função Run.
  // Mostra também o tipo do evento, quem está rodando e a que instante de tempo.
  MESSAGE<<endl<<setprecision(16)<<"("<<_Event->GetEventTime()<<")"<<endl;
  MESSAGE<<"--> Running event ..................................................... "
         <<_Event->GetEventType()<<" at "<<GetName()<<" of block "<<GetBlockName()<<endl;

  if (_Event->GetEventType() == "LOOP_EVENT") {
    double time = _Event->GetEventTime();
    Packet *pPacket = new Packet;
    pPacket->SetID(1);
    MESSAGE<<"--> Criando envento... "<< pPacket->ID <<endl;
    CreateLayerControlEvent( (time+1),"EV1","BLOCK_NODE_2","LAYER_NODE_2", (SMessage*) pPacket);

    // ---------------------------- envio de mensagem remota --------------------
    MESSAGE<<"--> enviando mensagem para SERVER... "<<endl;

    Distrib *d = new Distrib();

    int status = EXIT_SUCCESS;
    try {
      status = d->run("localhost","1038");
    } catch(const CORBA::Exception& /* ex */) {
      //cerr << ex << endl;
      status = EXIT_FAILURE;
    }
  }

  if (_Event->GetEventType() == "EV2") {
    MESSAGE<<"--> Chegou evento EV2... "<<_Event->GetEventType()<<" at "<<GetName()
           <<" of block "<<GetBlockName()<<endl;
    Packet *PPacketTMP=NULL;
    PPacketTMP=(Packet *)_Event->GetEventMessage();
    MESSAGE<< PPacketTMP->ID <<endl;
  }
  return 0; // 0: indica que nao houve erro na execucao da funcao Run. 1: erro na execucao.
}
```

```
// ------------------------------------------------------------------------------
// Esta função pode ser usada para executar qualquer processamento
// adicional em uma camada após o final de uma simulação.
int LAYER_NODE_1::PosProcessing()
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> Pos Processing "<<GetName()<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função pode ser usada para executar a limpeza da memória
// alocada pela camada durante a execução de outras funções.
// Não é obrigatório implementar esta função.
int LAYER_NODE_1::Terminate()
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> Terminate "<<GetName()<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma nova conexão de blocos já tenha sido estabelecida.
// ConnecName é o nome da conexão recém criada.
// Não é obrigatório implementar esta função.
int LAYER_NODE_1::OnConnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma conexão de blocos já tenha sido removida do kernel.
// ConnecName é o nome da conexão removida.
// Não é obrigatório implementar esta função.
int LAYER_NODE_1::OnDisconnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma nova conexão de rede já tenha sido estabelecida.
// ConnecName é o nome da conexão recém criada.
// Não é obrigatório implementar esta função.
int LAYER_NODE_1::OnNetworkConnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnNetworkConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}
```

```
// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma conexão de rede já tenha sido removida do kernel.
// ConnecName é o nome da conexão removida.
// Não é obrigatório implementar esta função.
int LAYER_NODE_1::OnNetworkDisconnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnNetworkDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma nova conexão de dados já tenha sido estabelecida.
// ConnecName é o nome da conexão recém criada.
// Não é obrigatório implementar esta função.
int LAYER_NODE_1::OnDataConnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnDataConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma conexão de dado já tenha sido removida do kernel.
// ConnecName é o nome da conexão removida.
// Não é obrigatório implementar esta função.
int LAYER_NODE_1::OnDataDisconnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnDataDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// ------------ Implementacao das suas funcoes para o Layer --------


// ================================ end! ====================================
```

layer_node_1.h

```
/* ------------------------------------------------------------
   LAYER_NODE_1.H
   ------------------------------------------------------------
*/
#ifndef __LAYER_NODE_1_H
#define __LAYER_NODE_1_H

// Esta função permite exportar este Layer para a DLL.
#ifndef _EXPORT
#define _EXPORT
#endif

// Bibliotecas de manipulacao do formato da sáida dos resultados.
#include <iomanip.h>
```

```
#include <stdio.h>

// ---- Hydragyrum ----
#ifndef __BLOCK_H
#include "block.h"
#endif

using namespace std;

class _EXPORT LAYER_NODE_1 : public Layer {
  // Dados e métodos privados do modelo.
  private:
   double      Parametro_1;
   double    Time;

  // Dados e métodos públicos do modelo.
  public:
    // ------- Dados --------------
    // ------- Metodos ------------
    // Construtor do modelo.
    LAYER_NODE_1(){}

    // Destrutor do modelo.
    ~LAYER_NODE_1(){}

    // ------ Metodos padronizados da Interface Hydragyrum ----------
    int Setup();
    int OnChangeParameter(Sstring ParamName);
    int OnStop(Sstring Stop, double Time, int StopCode);
    int Initiate();
    int Run( Event * _Event );
    int Post( Event * _Event );
    int PosProcessing();
    int Terminate();
    int OnConnect(Sstring ConnecName);
    int OnDisconnect(Sstring ConnecName);
    int OnNetworkConnect(Sstring ConnecName);
    int OnNetworkDisconnect(Sstring ConnecName);
    int OnDataConnect(Sstring ConnecName);
    int OnDataDisconnect(Sstring ConnecName);

    // ----- MY Metodos da Layer ------------
};

#endif
```

Artefatos do Projeto-2

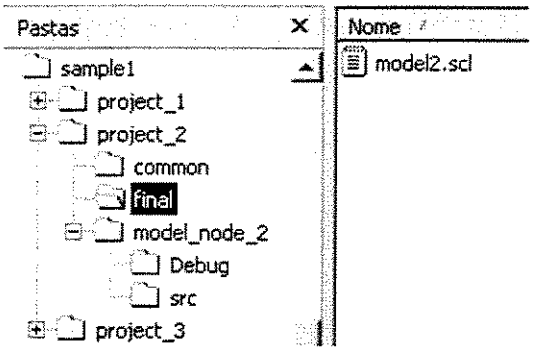

Figura AC-6: Artefatos do Projeto-2

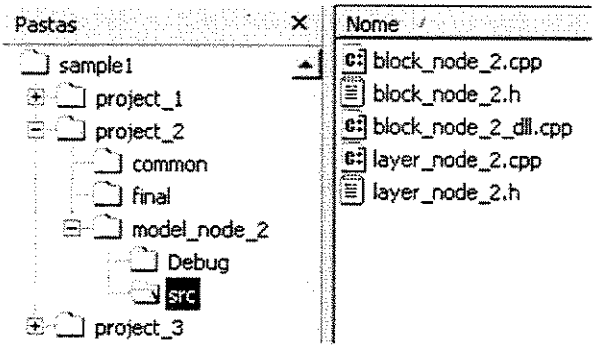

Figura AC-7: Artefatos do Projeto-2

model2.scl

```
block "BLOCK_NODE_2" model_node_2.dll
```

block_node_2.cpp

```
/* ----------------------------------------------------------------------------
   BLOCK_NODE_2.CPP
   ----------------------------------------------------------------------------
*/

#include "block_node_2.h"

int BLOCK_NODE_2::Setup()
{
  // Set the model name that will be shown in ther GUI
  Sstring N1("BLOCK_NODE_2");
  SetModel(N1);

  // Set the general information about the model
  Author      = "Ramme";
  Version     = "1.0";
  Date        = "28/06/2004";
  Copyright   = "MC21-DECOM-FEEC-UNICAMP";
  Description = "Modelo 2.";

  // Create the layers that will compose the model
  LAYER_NODE_2  * PLNMSC = new LAYER_NODE_2;

  // Seta camadas
  Sstring N2("LAYER_NODE_2");
  Sstring N3("LAYER_NODE_2");
  PLNMSC->SetName(N2);
  PLNMSC->SetType(N3);
```

```
   // Add the layers to the model
   AddLayer(PLNMSC);

   // Definicao das coneccoes permitidas:
   // tipo_layer_neste_bloco, tipo_layer_permitido_para_conectar_neste_bloco,
   // tipo_coneccao_resultante_entre_2layers
   AddInterface("LAYER_NODE_2","LAYER_NODE_1","EV1");

   // Configure the GUI appearence of the model
   // Configure the model block size in the GUI
   ConfigureBlock(54,54,"BORDER","BLACK",1); // BORDER // NOBORDER

   // Configure the ports that will be shown in the GUI
   // associate a layer to each port and choose the port look

   // ports in the right side of the block
   //      | layer name     | layer position     |color|size|type |
   AddInterfacePort("LAYER_NODE_2","LEFT","RED",0.15,"INPUT_ARROW");
   AddInterfacePort("LAYER_NODE_2","LEFT","GREEN",0.15,"OUTPUT_ARROW");

   return 0;
}

// -------------------- Metodos da Interface Bloco --------------------------------
// --------------------------------------------------------
int BLOCK_NODE_2::OnChangeParameter(Sstring ParamName)
{
  MESSAGE<<"-01-> OnChangeParameter "<<GetName()<<" Parameter "<<ParamName<<endl;
  return 0;
}

// --------------------------------------------------------
int BLOCK_NODE_2::OnStop(Sstring Stop, double Time, int StopCode)
{
  MESSAGE<<"-04-> OnStop "<<GetName()<<endl;
  return 0;
}

// --------------------------------------------------------
int BLOCK_NODE_2::Initiate()
{
  MESSAGE<<"-02-> Initiate "<<GetName()<<endl;
  return 0;
}

// --------------------------------------------------------
int BLOCK_NODE_2::Run( Event *_Event )
{
  MESSAGE<<endl<<setprecision(16)<<"("<<_Event->GetEventTime()<<")"<<endl;
  MESSAGE<<"-03-> Running event ..................................................... "
         <<_Event->GetEventType()<<" at "<<GetName()<<endl;
  return 0;
}

// --------------------------------------------------------
int BLOCK_NODE_2::Post( Event * _Event )
{
  MESSAGE<<"-MY-> Post "<<GetName()<<endl;
  return 0;
}
```

```
// ------------------------------------------------------
int BLOCK_NODE_2::PosProcessing()
{
  MESSAGE<<"-05-> Pos Processing "<<GetName()<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_2::Terminate()
{
  MESSAGE<<"-06-> Terminate "<<GetName()<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_2::OnConnect(Sstring ConnecName)
{
  MESSAGE<<"-07-> OnConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_2::OnDisconnect(Sstring ConnecName)
{
  MESSAGE<<"-08-> OnDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_2::OnNetworkConnect(Sstring ConnecName)
{
  MESSAGE<<"-09-> OnNetworkConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_2::OnNetworkDisconnect(Sstring ConnecName)
{
  MESSAGE<<"-10-> OnNetworkDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_2::OnDataConnect(Sstring ConnecName)
{
  MESSAGE<<"-11-> OnDataConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------
int BLOCK_NODE_2::OnDataDisconnect(Sstring ConnecName)
{
  MESSAGE<<"-12-> OnDataDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ============================== end! ==================================
```

block_node_2.h

```
#ifndef __BLOCK_NODE_2_H
#define __BLOCK_NODE_2_H

#ifndef  __BLOCK_H
#include "block.h"
#endif

#ifndef  __LAYER_NODE_2_H
#include "layer_node_2.h"
#endif


class BLOCK_NODE_2 : public Block {

public:

   // Interface
   int Setup();
   int OnChangeParameter(Sstring ParamName);
   int OnStop(Sstring Stop, double Time, int StopCode);
   int Initiate();
   int Run( Event * _Event );
   int Post( Event * _Event );
   int PosProcessing();
   int Terminate();
   int OnConnect(Sstring ConnecName);
   int OnDisconnect(Sstring ConnecName);
   int OnNetworkConnect(Sstring ConnecName);
   int OnNetworkDisconnect(Sstring ConnecName);
   int OnDataConnect(Sstring ConnecName);
   int OnDataDisconnect(Sstring ConnecName);

};

//Block * _export GetModel10();
__declspec(dllexport)  Block * GetModel10();

#endif
```

block_node_2_dll.cpp

```
/* ------------------------------------------------------------------------
   BLOCK_NODE_2_DLL.CPP
   Criacao da DLL do Bloco.
   ------------------------------------------------------------------------
*/

#include "block_node_2.h"

//Block * _export GetModel10()
__declspec(dllexport) Block * GetModel10()
{
      return new BLOCK_NODE_2;
}

// ================================ end! ================================
```

layer_node_2.cpp

```
/* ------------------------------------------------------------------------
   LAYER_NODE_2.CPP
   ------------------------------------------------------------------------
*/

// Exporta implementação dos métodos do modelo para a DLL. Não mexa.
//#define _EXPORT __declspec(dllexport)


#ifndef __LAYER_NODE_2_H
#include "layer_node_2.h"
#endif

#ifndef __PACKET_H
#include "packet.h"
#endif

// ---------------------------------------------------------------------------
// Nesta função o desenvolvedor de modelos deve implementar a:
//     - Configuração dos valores default das variáveis de manipulação dos
//   parâmetros do modelo.
// - Configuração dos valores default das variáveis auxiliares do modelo.
//     - Gravação dos valores default dos parâmetros do modelo.
int LAYER_NODE_2::Setup()
{
  // Mostra na tela do simulador que foi executada a função Setup. Não mexa.
  MESSAGE<<"-->  Setup "<<GetName()<<endl;
  return 0;
}

// ---------------------------------------------------------------------------
// Esta função é executada toda vez que um parâmetro é modificado em algum
// modelo ou no kernel do simulador. Permite a camada responder a
// modificações nos seus parâmetros e a modificações nos parâmetros
// do kernel e de outros modelos. Não é obrigatório implementar esta função.
int LAYER_NODE_2::OnChangeParameter(Sstring ParamName)
{
  // Mostra na tela do simulador que foi executada a função OnChangeParameter.
  MESSAGE<<"--> OnChangeParameter "<<GetName()<<endl;
  return 0;
}

// ---------------------------------------------------------------------------
// Esta função permite a camada responder a uma parada de simulação.
// Não é obrigatório implementar esta função.
int LAYER_NODE_2::OnStop(Sstring Stop, double Time, int StopCode)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnStop "<<GetName()<<endl;
  return 0;
}
```

```
// ------------------------------------------------------------------------------
int LAYER_NODE_2::Post( Event * _Event )
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> Post "<<GetName()<<endl;
  return 0;
}


// ------------------------------------------------------------------------------
// Nesta função o desenvolvedor de modelos deve implementar a leitura
// dos valores dos parâmetros da camada e dos parâmetros globais.
// Carrega variáveis locais para a manipulação de parâmetros com o valor
// atual destes. Isto deve ser feito porque o valor do parâmetro pode ter
// sido modificado desde o Setup, por exemplo pelo usuário na Interface Gráfica.
int LAYER_NODE_2::Initiate()
{
  // Mostra na tela do simulador que foi executada a função Initiate.
  MESSAGE<<"--> Initiate "<<GetName()<<endl;
  GetParam("Nome_do_Parametro_1",Parametro_1);
  return 0;
}


// ------------------------------------------------------------------------------
int LAYER_NODE_2::Run( Event * _Event )
{
  // Mostra na tela do simulador que foi executada a função Run.
  // Mostra também o tipo do evento, quem está rodando e a que instante de tempo.
  MESSAGE<<endl<<setprecision(16)<<"("<<_Event->GetEventTime()<<")"<<endl;
  MESSAGE<<"--> Running event ..................................................... "
         <<_Event->GetEventType()<<" at "<<GetName()<<" of block "<<GetBlockName()<<endl;

  if (_Event->GetEventType() == "EV1") {
    MESSAGE<<"--> Chegou evento EV1 ... "<<_Event->GetEventType()<<" at "<<GetName()
           <<" of block "<<GetBlockName()<<endl;
    // Captura instante de tempo atual, ou seja instante de tempo do evento recebido
    double Time = _Event->GetEventTime();
    Packet *PPacket;
    PPacket=(Packet *)_Event->GetEventMessage(); // * cuidar para o pacote nao estar vazio.
    // * pressupondo que o pacote nao esteja vazio ...
    CreateLayerControlEvent( (Time+1),"EV2","BLOCK_NODE_1","LAYER_NODE_1", (SMessage*) PPacket);
  }
  return 0; // 0: indica que nao houve erro na execucao da funcao Run. 1: erro na execucao.
}


// ------------------------------------------------------------------------------
// Esta função pode ser usada para executar qualquer processamento
// adicional em uma camada após o final de uma simulação.
int LAYER_NODE_2::PosProcessing()
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> Pos Processing "<<GetName()<<endl;
  return 0;
}
```

```
// ------------------------------------------------------------------------------
// Esta função pode ser usada para executar a limpeza da memória
// alocada pela camada durante a execução de outras funções.
// Não é obrigatório implementar esta função.
int LAYER_NODE_2::Terminate()
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> Terminate "<<GetName()<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma nova conexão de blocos já tenha sido estabelecida.
// ConnecName é o nome da conexão recém criada.
// Não é obrigatório implementar esta função.
int LAYER_NODE_2::OnConnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma conexão de blocos já tenha sido removida do kernel.
// ConnecName é o nome da conexão removida.
// Não é obrigatório implementar esta função.
int LAYER_NODE_2::OnDisconnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma nova conexão de rede já tenha sido estabelecida.
// ConnecName é o nome da conexão recém criada.
// Não é obrigatório implementar esta função.
int LAYER_NODE_2::OnNetworkConnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnNetworkConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}


// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma conexão de rede já tenha sido removida do kernel.
// ConnecName é o nome da conexão removida.
// Não é obrigatório implementar esta função.
int LAYER_NODE_2::OnNetworkDisconnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnNetworkDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}
```

```
// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma nova conexão de dados já tenha sido estabelecida.
// ConnecName é o nome da conexão recém criada.
// Não é obrigatório implementar esta função.
int LAYER_NODE_2::OnDataConnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnDataConnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// Esta função permite modificar registros em tabelas e executar outras
// funções sempre que uma conexão de dado já tenha sido removida do kernel.
// ConnecName é o nome da conexão removida.
// Não é obrigatório implementar esta função.
int LAYER_NODE_2::OnDataDisconnect(Sstring ConnecName)
{
  // Mostra na tela do simulador que a função foi executada.
  MESSAGE<<"--> OnDataDisconnect "<<GetName()<<" Connection "<<ConnecName<<endl;
  return 0;
}

// ------------------------------------------------------------------------------
// ------------ Implementacao das suas funcoes para o Layer --------


// ================================ end! ====================================
```

layer_node_2.h

```
/* ------------------------------------------------------------
   LAYER_NODE_2.H
   ------------------------------------------------------------
*/

#ifndef __LAYER_NODE_2_H
#define __LAYER_NODE_2_H

// Esta função permite exportar este Layer para a DLL.
#ifndef _EXPORT
#define _EXPORT
#endif

// Bibliotecas de manipulacao do formato da sáida dos resultados.
#include <iomanip.h>
#include <stdio.h>

// ---- Hydragyrum ----
#ifndef __BLOCK_H
#include "block.h"
#endif

using namespace std;

class _EXPORT LAYER_NODE_2 : public Layer {
  // Dados e métodos privados do modelo.
  private:
```

```
    double      Parametro_1;


  // Dados e métodos públicos do modelo.
  public:
    // ------- Dados --------------

    // ------- Metodos ------------
    // Construtor do modelo.
    LAYER_NODE_2(){}

    // Destrutor do modelo.
    ~LAYER_NODE_2(){}

    // ------ Metodos padronizados da Interface Hydragyrum ----------
    int Setup();
    int OnChangeParameter(Sstring ParamName);
    int OnStop(Sstring Stop, double Time, int StopCode);
    int Initiate();
    int Run( Event * _Event );
    int Post( Event * _Event );
    int PosProcessing();
    int Terminate();
    int OnConnect(Sstring ConnecName);
    int OnDisconnect(Sstring ConnecName);
    int OnNetworkConnect(Sstring ConnecName);
    int OnNetworkDisconnect(Sstring ConnecName);
    int OnDataConnect(Sstring ConnecName);
    int OnDataDisconnect(Sstring ConnecName);

    // ----- MY Metodos da Layer ------------
};

#endif
```

Artefatos do Projeto-3

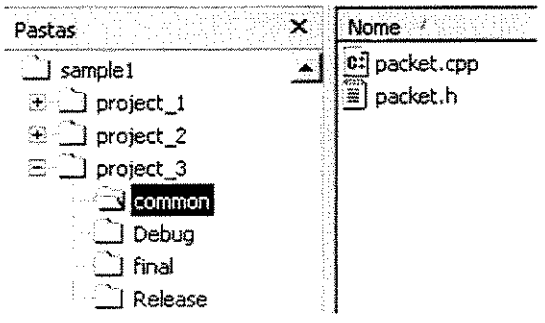

Figura AC-8: Artefatos do Projeto-3

packet.cpp

```
#ifndef __PACKET_H
#include "packet.h"
#endif

Packet::Packet()
{
}

Packet::~Packet()
{
}

int Packet::GetID()
{
  return ID;
}

void Packet::SetID(int _ID)
{
  ID = _ID;
}
```

packet.h

```
#ifndef __PACKET_H
#define __PACKET_H

#ifndef __SMESSAGE_H
#include "smessage.h"
#endif

#include <iostream.h>

class Packet : public SMessage {

public:
    Packet();
   ~Packet();

   // Class Data
   int  ID;

   // Class Interface
   int  GetID();
   void SetID(int _ID);

};

#endif
```