SIMILE - Um Sistema Orientado a Objetos para Simulação de Redes Locais: Projeto, Implementação e Comparações com Simuladores Tradicionais

Maria Madalena Dias
FUEM/CTC/DIN - Maringá, Pr

José Antão Beltrão Moura
UFPb/CCT/DSC-COPIN

Maria Izabel Cavalcanti Cabral
UFPb/CCT/DSC-COPIN

Marcos Antonio Gonçalves Brasileiro
UFPb/CCT/DEE
58100 - Campina Grande - Pb
Fone: (083)333-1929
FAX : (083)333-1945
E-Mail: copin@brfapesp.bitnet

Resumo

Este artigo apresenta o projeto e a implementação de um Sistema de Simulação Orientado a Objetos para Redes Locais, adotando o Padrão IEEE 802. Esse sistema, denominado SIMILE, por ser modular, extensível e reutilizável, pode ser facilmente adaptado para simular outras sub-redes de comunicação além daquelas especificadas pelo comitê IEEE 802 e pode, também, ser incorporado em simuladores de protocolos de níveis mais altos. A eficiência e custos do sistema em termos de tempos de execução e linhas de código, são comparados com outros simuladores tradicionais escritos em linguagens de programação de propósito geral e de propósito específico como C e GPSS, respectivamente.

Abstract

This paper presents the project and implementation of an Object-Oriented Simulation System for Local Area Networks adopting the IEEE 802 standards. The system, called SIMILE, being modular, extensible and reusable can be easily adapted to simulate communications subnetwork protocols other than those specified by the IEEE 802 committee and it can also be incorporated in simulators for higher level protocols. The system's efficiency and costs in terms of execution times and lines of code, are compared to those presented by traditional simulators written in general and specific purpose programming languages such as C and GPSS, respectively.

1 Introdução

No desenvolvimento de simuladores para avaliação de desempenho de Redes Locais de Computadores (RLCs), normalmente usam-se linguagens de simulação como GPSS [20], SIMULA [2], RESQ [15] e SIMSCRIPT [5], ou linguagens de programação de propósito geral como C, no simulador em [17] por exemplo.

A programação orientada a objetos oferece uma alternativa para o ambiente tradicional de simulação e suas linguagens. Na verdade, a essência da programação orientada a objetos é a simulação; ela fornece uma forma mais natural de abstrair o modelo do que é possível nas linguagens de simulação atuais. O problema é decomposto em um conjunto de objetos, onde cada objeto representa um objeto do modelo de simulação [1].

Pode-se observar através de experiências que projetos de simulação são evolutivos por natureza [9]. As características de modularidade, extensibilidade e reutilização de software, propiciadas pela programação orientada a objetos, facilitam a manutenção de programas de simulação.

As linguagens de programação orientadas a objetos oferecem muitas vantagens sobre linguagens de programação tradicionais que facilitam a construção e a manutenção de programas de simulação. Tais vantagens são: encapsulamento (*information hiding*), abstração de dados, herança e amarração dinâmica (*dynamic binding*) [8].

A linguagem de programação C++ [21] foi desenvolvida especificamente para solucionar problemas de simulação. C++ suporta completamente os recursos de uma linguagem de programação orientada a objetos, além de reter os altos níveis de compactação e a velocidade da linguagem de programação C [22].

Este artigo apresenta um simulador reutilizável, escrito em C++, para avaliar o desempenho de RLCs, denominado SIMILE.

O SIMILE, projetado para RLCs com Protocolos de Acesso ao Meio conforme Padrão IEEE 802 (Anel com Passagem de Ficha, Barra com Passagem de Ficha e Barra com CSMA/CD) [12], objetiva, também, servir como protótipo de simulação de RLCs que possa ser facilmente expandido para o estudo e a avaliação de outros protocolos de comunicação. O SIMILE é, também, parte integrante do módulo solução do projeto denominado "SAVAD - Uma Ferramenta Inteligente para Avaliação de Desempenho de Sistemas Distribuídos" [4].

Aqui, pretende-se ilustrar e apresentar os principais aspectos da metodologia orientada a objetos no projeto e implementação de simuladores. Para tanto, o SIMILE será discutido em detalhes na seção 2, no que se refere a identificação de suas classes de objetos e do modelo de relacionamento entre os objetos. Mais informações sobre o simulador desenvolvido podem ser obtidos em [7]. Espera-se que a seção 2 contribua para a área de avaliação de desempenho de Redes Locais já que expõe e exemplifica os passos necessários para o projeto orientado a objetos de simuladores.

Após discutir brevemente a implementação e validação do SIMILE na seção 3, comparam-se as vantagens e limitações da programação orientada a objetos face às alternativas tradicionais de desenvolvimento de simuladores em C e em GPSS. As observações feitas correlacionam-se relativamente bem com aquelas apresentadas por Doyle em [9]. As comparações e conclusões da seção 4 contribuem para a Engenharia de Software já que oferecem subsídios para estimar consumo de recursos (dedicação de

analistas/programadores, UCP, etc.) em projetos de simuladores seguindo a metodologia orientada a objetos.

Conclusões e informações sobre a continuidade do trabalho com o SIMILE são apresentadas na seção 5.

2 Apresentação do Projeto Orientado a Objetos do SIMILE

O projeto orientado a objetos, como outras metodologias de projeto orientado a informação, cria uma representação do domínio do problema do mundo real e mapeia este problema em um domínio de solução que é o software [19].

O projeto de um sistema de software orientado a objetos é construído em torno de um conjunto de classes que caracterizam o comportamento dos objetos do mundo real relativos ao sistema. Em geral, uma classe é a descrição de um conjunto de objetos que possuem os mesmos atributos e comportamentos e, um objeto é uma instância específica de uma classe com comportamento bem definido. Os objetos de cada classe são manipulados através do envio de mensagens. Estas mensagens representam o conjunto de ações que são tomadas sobre o conjunto de objetos.

O primeiro passo, e o mais importante, no projeto orientado a objetos é a identificação das classes de objetos que compõem o sistema do mundo real. O segundo é a definição das mensagens que estabelecem as interfaces dos objetos [18].

2.1 Classes de Objetos

As principais classes identificadas no SIMILE são sucintamente descritas a seguir:

1) Controle: O objeto desta classe controla todo o processo de simulação (inicialização, processamento e apresentação dos resultados).
2) Protocolo: É uma classe virtual que engloba os aspectos gerais dos protocolos de acesso ao meio, conforme padrão IEEE 802.
3) PassagemFicha: As operações desta classe definem o controle da simulação quando os protocolos de acesso ao meio são Anel com Passagem de Ficha e Barra com Passagem de Ficha.
4) CsmaCD: As operações desta classe definem o controle da simulação quando o protocolo de acesso ao meio é em Barra com CSMA/CD.
5) Evento: Contém informações sobre cada evento.
6) ListaEventos: As operações desta classe permitem o gerenciamento dos eventos que ocorrem durante a simulação. Contém apontadores para os eventos.
7) FichaLivre: Define a ficha livre que é passada de interface a interface nos protocolos Anel com Passagem de Ficha e Barra com Passagem de Ficha.
8) Classe: Contém informações gerais sobre cada classe de pacotes[1] (que pode ser: voz, dados, imagens, etc.).
9) ListaClasses: Contém apontadores para as classes.
10) Interface: Contém informações sobre cada interface.

---

1. O termo classe ou classe de objetos, usado anteriormente, representa um conjunto de objetos com as mesmas características. O termo classe de pacotes diz respeito aos tipos de pacotes (voz, dados, imagens, etc.) que são especificados nos modelos de RLCs.

11) ListaInterfaces: Contém apontadores para as interfaces.
12) Pacote: Contém informações sobre cada pacote.
13) FilaPacotes: As operações desta classe permitem o controle da entrada e da saída de pacotes nas filas das interfaces. Contém apontadores para os pacotes.
14) MeioTrans: Define o meio de transmissão.
15) Relógio: Define o tempo simulado.
16) AnelLógico: Controla a ordem de passagem de ficha entre as interfaces, quando o protocolo sendo simulado é do tipo Barra com Passagem de Ficha. Contém apontadores para as interfaces.
17) Função: Operações desta classe permitem a geração de amostras de intervalos de tempo segundo funções de distribuição de probabilidade definidas no SIMILE.
18) TWindow: Esta é uma classe definida pela Zortech [23]. Ela controla a abertura, edição e fechamento de janelas.
19) Menu: Esta é uma classe definida pela Zortech [23]. Ela controla a escolha de opções em um menu.

2.2 Modelo Cliente-Servidor

O projeto de um sistema orientado a objetos não é realizado apenas em termos dos objetos, mas também dos serviços que eles fornecem a outros objetos. O modelo cliente-servidor dá uma visão geral do relacionamento entre os objetos. Nesse modelo cada objeto interage com os outros objetos através de mensagens que passam ou buscam informações, solicitam aos objetos a implementação de um procedimento, etc. [11].

A figura 2.1 mostra a notação usada no modelo cliente-servidor apresentado nas figuras 2.2 e 2.3.

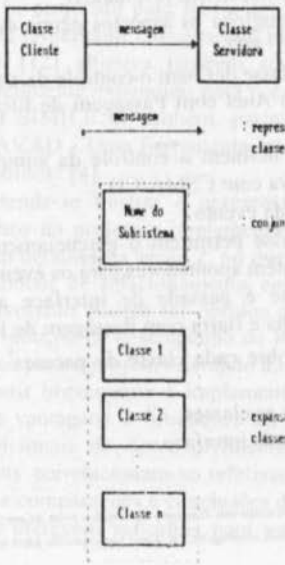

Fig. 2.1 Notação do Modelo Cliente Servidor

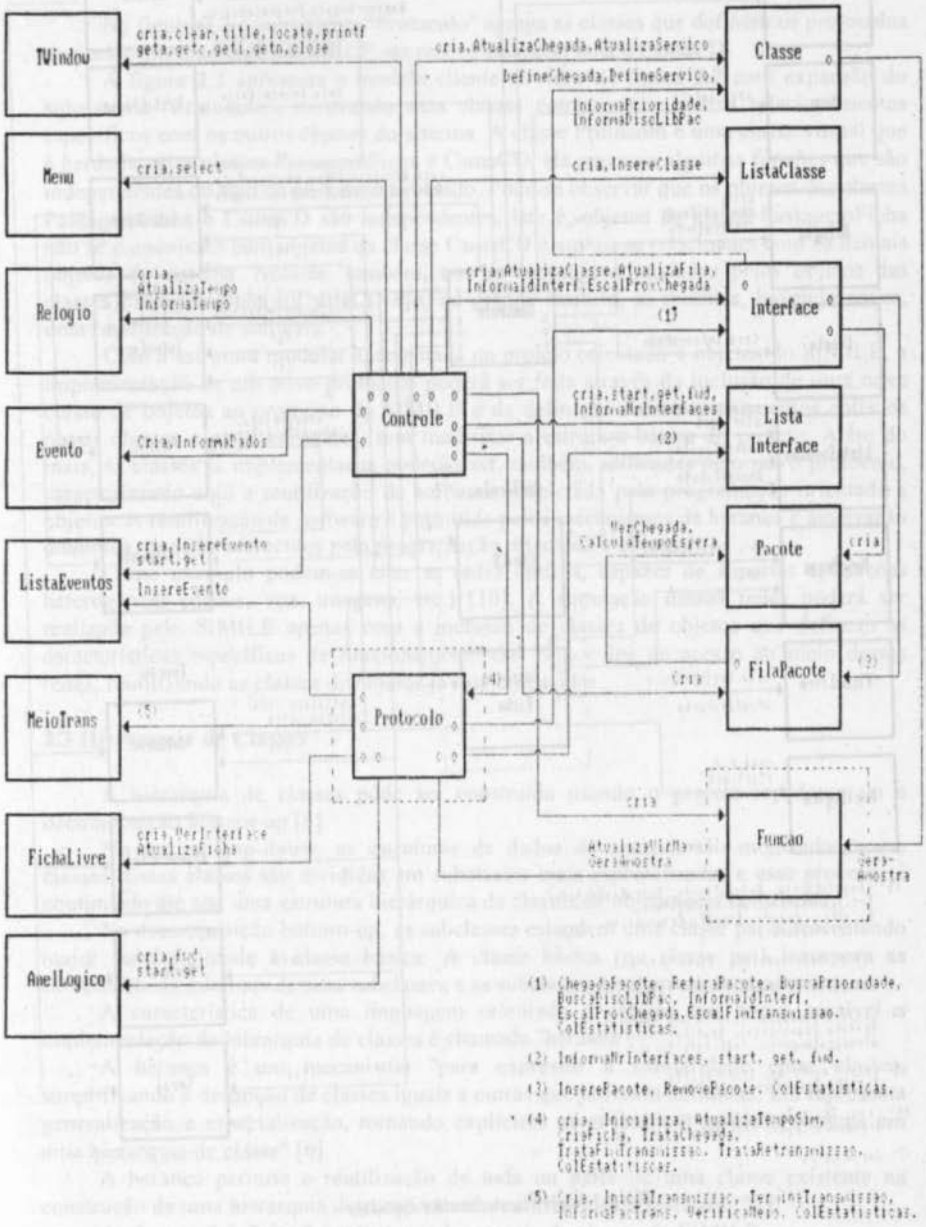

Fig. 2.2 Modelo Cliente-Servidor

Na figura 2.2 o subsistema "Protocolo" agrega as classes que definem os protocolos de acesso simulados pelo SIMILE, no caso, PassagemFicha e CsmaCD.

A figura 2.3 apresenta o modelo cliente-servidor da figura 2.2 com expansão do subsistema "Protocolo", mostrando suas classes componentes e seus relacionamentos específicos com os outros objetos do sistema. A classe Protocolo é uma classe virtual que é herdada pelas classes PassagemFicha e CsmaCD, ela gerencia algumas funções que são independentes do tipo de protocolo simulado. Pode-se observar que os objetos das classes PassagemFicha e CsmaCD são independentes, isto é, objetos da classe PassagemFicha não se comunicam com objetos da classe CsmaCD e ambos se relacionam com os demais objetos do sistema. Nota-se, também, que as mensagens enviadas pelos objetos das classes PassagemFicha e CsmaCD são, na grande maioria, as mesmas, havendo assim, uma reutilização de software.

Com a estrutura modular apresentada no projeto orientado a objetos do SIMILE, a implementação de um novo protocolo poderá ser feita através da inclusão de uma nova classe de objetos ao protótipo do SIMILE e da definição dos relacionamentos entre os novos objetos e os já existentes, sem modificar a estrutura básica do projeto. Além do mais, as classes já implementadas poderão ser, também, utilizadas pelo novo protocolo, caracterizando aqui a reutilização de software propiciada pela programação orientada a objetos. A reutilização de software é permitida pelos mecanismos de herança e amarração dinâmica, que são oferecidos pela programação orientada a objetos.

Como exemplo podem-se citar as redes ópticas, capazes de suportar aplicações heterogêneas (dados, voz, imagens, etc.) [10]. A simulação dessas redes poderá ser realizada pelo SIMILE apenas com a inclusão de classes de objetos que definem as características específicas de funcionamento dos protocolos de acesso ao meio dessas redes, reutilizando as classes de objetos já implementadas.

2.3 Hierarquia de Classes

A hierarquia de classes pode ser construída usando o projeto top-down ou a decomposição bottom-up [6].

No projeto top-down, as estruturas de dados do sistema são modeladas como classes. Essas classes são divididas em subclasses mais especializadas e esse processo é continuado até que uma estrutura hierárquica de classes de objetos seja construída.

Na decomposição bottom-up, as subclasses estendem uma classe pai acrescentando maior funcionalidade à classe básica. A classe básica (ou classe pai) incorpora as características mínimas de suas subclasses e as subclasses herdam essas características.

A característica de uma linguagem orientada a objetos que torna possível a implementação da hierarquia de classes é chamada "herança".

A herança é um mecanismo "para expressar a similaridade entre classes, simplificando a definição de classes iguais a outras que já foram definidas. Ela representa generalização e especialização, tornando explícitos os atributos e serviços comuns em uma hierarquia de classe" [6].

A herança permite a reutilização de toda ou parte de uma classe existente na construção de uma hierarquia de componentes reutilizáveis de software [22].

As figuras 2.4, 2.5 e 2.6 mostram a hierarquia de classes do SIMILE.

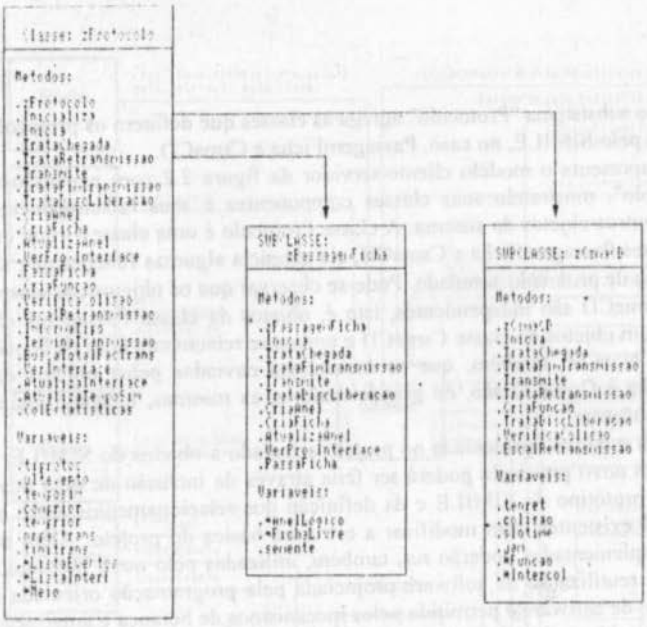

Fig. 2.4 Hierarquia da Classe zProtocolo

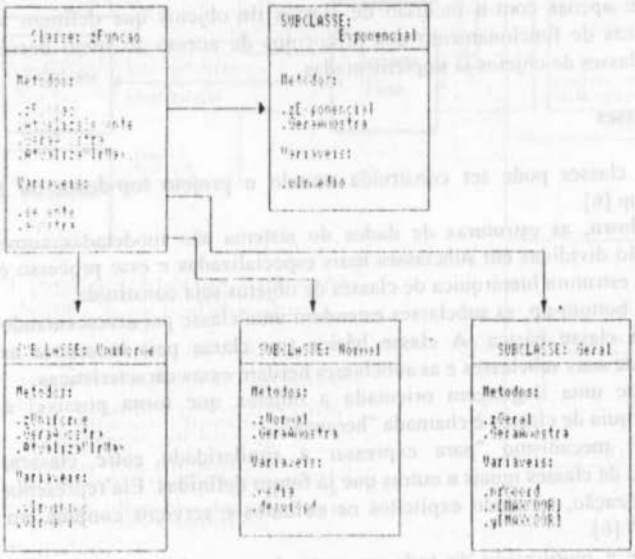

Fig. 2.5 Hierarquia da Classe zFuncao

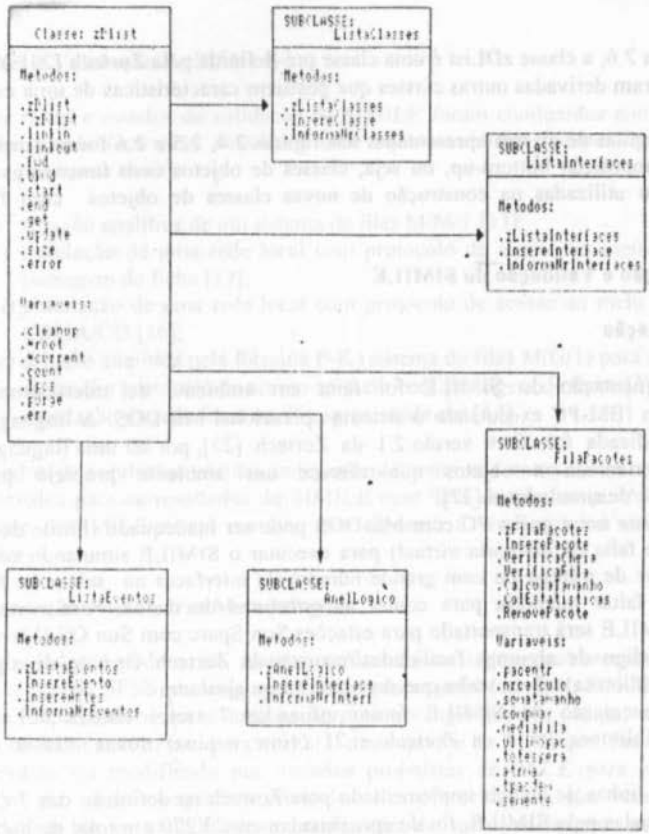

Fig. 2.6 Hierarquia da Classe zDList

As figuras 2.4, 2.5 e 2.6 foram baseadas nas representações de hierarquia de classes sugeridas por Pressman em [19]. Nessas figuras, a direção da seta (-->) indica a hierarquia das classes. Como pode ser visto nessas figuras, as subclasses herdam todas ou quase todas as características das superclasses.

As figuras 2.4 e 2.5 apresentam métodos das subclasses que possuem os mesmos nomes de métodos das superclasses (ex.: o método TrataChegada é definido na classe zProtocolo e em suas classes derivadas zPassagemFicha e zCsmaCD), isso indica que os métodos das subclasses redefinem os métodos de suas superclasses, por possuírem funcionamentos diferentes. A característica de amarração dinâmica da programação orientada a objetos determina qual dos objetos das subclasses receberá a mensagem (método da subclasse ou superclasse), somente em tempo de execução, possibilitando desta forma o polimorfismo.

O polimorfismo permite que uma mesma mensagem seja enviada a todos os objetos dentro de uma classe e suas subclasses, como se cada objeto soubesse como responder a essa mensagem, talvez de uma maneira diferente dos objetos de outras subclasses [22].

Na figura 2.6, a classe zDList é uma classe pré-definida pela Zortech [23]. A partir dessa classe foram derivadas outras classes que possuem características de uma estrutura do tipo lista.

As hierarquias de classes apresentadas nas figuras 2.4, 2.5 e 2.6 foram construídas usando a decomposição bottom-up, ou seja, classes de objetos com funções gerais, já existentes, são utilizadas na construção de novas classes de objetos com funções específicas.

3 Implementação e Validação do SIMILE

3.1 Implementação

A implementação do SIMILE foi feita em ambiente de microcomputador compatível com IBM-PC executando o sistema operacional MS-DOS. A linguagem de programação utilizada foi C++ versão 2.1 da Zortech [23], por ser uma linguagem de programação orientada a objetos que oferece um ambiente propício para o desenvolvimento de simuladores [22].

É importante notar que o PC com MS-DOS pode ser inadequado (limite de RAM de 640 kbytes e falta de memória virtual) para executar o SIMILE simulando cenários com alto volume de tráfego ou com grande número de interfaces na sub-rede. Nesses cenários, pode faltar memória para conter as estruturas de dados. Para evitar este problema, o SIMILE será transportado para estações Sun Sparc com Sun OS (Unix). No transporte, o código de algumas facilidades/recursos da Zortech C++ (e de algumas classes da sua biblioteca) talvez tenha que ser reescrito ou ajustado.

Na implementação do SIMILE foram utilizadas 7 (sete) classes de objetos existentes na biblioteca fonte da Zortech e 21 (vinte e uma) novas classes foram implementadas.

O total de linhas de código implementado pela Zortech na definição das 7 classes de objetos, utilizadas pelo SIMILE, foi de aproximadamente 3.270 e o total de linhas de código que foi necessário na definição das 21 novas classes implementadas foi de aproximadamente 5.250. Em ambos os casos foram consideradas as linhas de comentários e as linhas em branco.

A construção do SIMILE no que se refere às fases de especificação do projeto, implementação com documentação, testes e validação, consumiu um total de 7 meses. Isto resulta em uma taxa média de produção de 750 linhas documentadas por mês, com a seguinte distribuição pelas fases:

i) 3 meses no estudo de metodologias de projeto orientado a objetos e especificação/estruturação do SIMILE;
ii) 2,5 meses na codificação em C++; e,
iii) 1,5 meses em testes, verificação e validação.

A proficiência em C++ e a disponibilidade de módulos reutilizáveis do SIMILE (por exemplo, rotinas para enfileiramento de "pacotes" em filas) poderão abreviar futuros esforços na fase ii) para simuladores de redes locais.

3.2 Testes e Validação do SIMILE

Os testes e estudos de validação do SIMILE foram conduzidos comparando-se os resultados produzidos pelo simulador com aqueles obtidos independentemente pelas formas e nos casos abaixo:

i) solução analítica de um sistema de filas M/M/1 [13];
ii) simulação de uma rede local com protocolo de acesso ao meio em anel com passagem de ficha [17];
iii) simulação de uma rede local com protocolo de acesso ao meio em barra com CSMA/CD [16];
iv) solução analítica pela fórmula P-K (sistema de filas M/G/1) para uma rede local em anel e protocolo de acesso ao meio com passagem de ficha [3]; e,
v) simulação da rede local em iv) apresentada em [14].

Todos os resultados obtidos independentemente nos casos i) a v) ficaram dentro dos intervalos para os resultados do SIMILE com 95% de segurança. A rigor, isto não prova que o SIMILE esteja correto, mas aumenta a confiança na validade de seus resultados.

4 Comparação com Outras Metodologias

Simuladores podem ser escritos utilizando-se linguagens/sistemas de propósito específico (como GPSS) ou linguagens de propósito geral (como C, FORTRAN ou C++). O SIMILE apresenta-se como uma opção mais abrangente desta última, constituindo-se de um simulador reutilizável para avaliar o desempenho de RLCs que, facilmente, poderá ser estendido ou modificado por usuários projetistas de RLCs, para modelar outras aplicações ou outros requisitos além daqueles oferecidos na sua versão inicial. As características de extensibilidade e reutilização de software surgem naturalmente no projeto do SIMILE, devido à abordagem orientada a objetos utilizada no seu desenvolvimento.

Segundo Doyle em [9], a principal característica a ser analisada na escolha de uma ferramenta para construção de programas de simulação é a velocidade de execução, pelo fato de programas de simulação causarem, tradicionalmente, uma alta utilização da UCP.

Apesar das linguagens de simulação de propósito geral serem consideradas de fácil utilização, propiciarem um melhor entendimento das relações causa-efeito do sistema sendo modelado e, poderem ser utilizadas para desenvolver outras aplicações por programadores que nelas se tornem proficientes, os simuladores desenvolvidos, normalmente são mais extensos. Por outro lado, também são de execução mais rápida. O objetivo aqui é comparar os custos-benefícios de se adotar C, C++ ou GPSS na construção de simuladores, tomando-se uma Rede Local com Protocolo em Anel com Passagem de Ficha como ambiente ou sistema de interesse.

Adiante-se que em análises quantitativas e comparativas feitas por Doyle em [9] com algumas linguagens de programação orientadas a objetos e outras não orientadas a objetos, constatou-se que C é a linguagem que possui melhor desempenho. A linguagem de programação C++ ficou em segundo lugar.

No estudo comparativo feito aqui, toma-se o SIMILE (em C++), o simulador apresentado em [17] (em C) e o simulador mostrado em [16] (em GPSS). A comparação leva em conta o número de linhas de código e o tempo de execução de cada programa simulador. Os dois primeiros simuladores foram executados em um PC com processador iAPX 80386 SX de 16 MHz e o último em um Honeywell H66/60. Os tempos de execução no Honeywell foram reduzidos por um fator de 20% para efeito de normalização de capacidade de processamento (estimam-se o 386 SX de 16 MHz com 1 MIPS e o H66/60 com 0,8 MIPS). Com exceção do SIMILE, os outros simuladores não apresentam interface amigável. Por isto, na contagem de linhas de código, foram desprezadas as rotinas de interface de todos os simuladores. A tabela 1 traz os números da comparação.

Da tabela 1, vê-se que C++ exige mais esforço do programador. Na verdade, o simulador em C++ inclui funcionalidade adicional para permitir a agregação de outros protocolos de acesso e a reutilização de seus módulos em simuladores mais elaborados de camadas superiores de protocolos. Os outros dois simuladores foram desenvolvidos especificamente para o ambiente de interesse, sem preocupação de oferecer modularidade ou extensibilidade. Das contagens de linhas, foram excluídas as declarações de funções das bibliotecas de cada linguagem (ex.: rand() do C e o código das classes de objetos disponíveis na Zortech). A vantagem de GPSS deve ser ponderada contra sua limitação de ser empregada para simuladores de sistemas que não possam ser facilmente caracterizados como sistemas de filas.

	C	C++	GPSS
Numero de Linhas de Codigo	240	1200	149
Tempo de Execucao (s)	75	80-135	360-720

Tabela 1: Comparacao entre simuladores em C, C++ e GPSS para Rede Local em Anel com Passagem de Ficha (16 interfaces ativas e utilizacao do meio de 40%)

A estrutura mais pesada de suporte do GPSS, seguida daquela de C++, é responsável pelo maior tempo de execução. A variação nos tempos dos simuladores nestas duas linguagens é devido à passagem da ficha de uma interface para a próxima (os tempos menores são para o caso de se assumir interferência desprezível da transmissão da ficha). O simulador escrito em C, adicionando o tempo de transmissão da ficha ao tempo de transmissão dos pacotes de dados, aproxima satisfatoriamente os resultados para baixa intensidade de tráfego (utilização de 40%) e sofre apenas um pequeno acréscimo no seu tempo de execução. Deve-se notar que os tempos de execução aumentam com a intensidade de tráfego, número de interfaces e detalhes a serem simulados. Os tempos de C++ são ligeiramente maiores que os de C. Com a disponibilidade de processadores dedicados, a questão de eficiência na execução torna-se secundária, considerando, principalmente, as facilidades oferecidas ao programador.

Devido as suas características de reutilização de código e à metodologia de programação implícita em C++ (simuladores em C, quando bem escritos e modulares, terminam sendo desenvolvidos seguindo a metodologia de programação orientada a objetos), e devido a inerentes limitações de GPSS e sua estreita gama de aplicações, parece vantajoso investir-se no desenvolvimento de simuladores usando C++. É importante destacar, também, que o maior número de linhas em C++ apresentado na tabela 1 significará provavelmente menos esforço de programação futura para extensão e/ou alteração do simulador construído.

5 Conclusão

O SIMILE foi projetado para avaliar o desempenho de Redes Locais com Protocolos de Acesso ao Meio, conforme Padrão IEEE 802, podendo servir como protótipo de desenvolvimento de simuladores para avaliação de desempenho de RLCs.

A inclusão de um novo protocolo de acesso ao meio ao protótipo do SIMILE implica, apenas, no acréscimo de classes de objetos e na definição de relacionamentos entre os novos objetos com os já existentes, podendo o novo protocolo reutilizar as classes de objetos já implementadas. Essa facilidade deve-se às características de modularidade, extensibilidade e reutilização de software da Abordagem Orientada a Objetos, que foram exploradas no desenvolvimento do SIMILE.

As comparações apresentadas na seção 4 mostraram que o SIMILE, implementado na linguagem de programação C++, apresentou tempo de execução, em determinadas condições, bem próximo ao apresentado pelo simulador implementado na linguagem de programação C. Essa constatação está de acordo com os estudos apresentados em literaturas que mostram que o desempenho de C++ é quase tão bom quanto o da linguagem de programação C.

Pretende-se dar continuidade a este trabalho incluindo outros protocolos de acesso ao meio ao protótipo do SIMILE, refinando a apresentação dos resultados do SIMILE de forma gráfica e desenvolvendo uma metodologia orientada a objetos para a construção de simuladores de sistemas de redes de filas.

Agradecimentos

Agradecemos ao CNPq pelo apoio financeiro.

Referências Bibliográficas

[1] ADELSBERGER, H.H., POOCH, U.W., SHANOON, R.E., WILLIAMS, G.N. *Rule based object-oriented simulation systems*. In: SIMULATION SERIES, Vol. 17, Nr. 1, 1986. p. 107-112.

[2] BIRTWISTLE, G.M. *Simula begin*. AUERBACH Publisher Inc. Philadelphia, Pa. 1973.

[3] BRASILEIRO, M.A.G. & MOURA, J.A.B. *Modelagem de aplicações críticas no tempo em redes locais com passagem de ficha*. In: SEMINAIRE FRANCO-BRASILEIRO SUR LES SYSTEMES INFORMATIQUES REPARTIS. Florianópolis, 1989. p. 257-265.

[4] Brasileiro, M.A.G., Cabral, M.I.C. & Silva, H.M. *SAVAD - Uma ferramenta para avaliar o desempenho de sistemas distribuídos*. SIMPÓSIO FRANCO-BRASILEIRO EM SISTEMAS DISTRIBUÍDOS. Florianópolis, 1989.

[5] BULGREN, W.G. *Discrete system simulation*. Prentice-Hall, In., Englewood Cliffs, N.J., 1982.

[6] COAD, Peter & YOURDON, Edward. *Análise baseada em objetos*. Trad. CTI Informática. 2a. Edição. Editora Campus. 1992. 252 p.

[7] DIAS, M.M., CABRAL, M.I.C. & BRASILEIRO, M.A.G. *Abordagem orientada a objetos no desenvolvimento de simuladores de Redes Locais*. 10o. SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES. Recife, 1992. p. 247-262.

[8] DIESCH, K. H. & BARTA, Thomas A. *Object-oriented discrete-event simulation in a strongly-typed procedural language*. Object-Oriented Simulation. In: SIMULATION SERIES, Vol. 23, No. 3, 1991. p. 43-49.

[9] Doyle, R.J. *Object-oriented simulation programming*. In: SCS MULTICONFERENCE ON OBJECT-ORIENTED SIMULATION. San Diego, Califórnia, 1990.

[10] Galdino, J.F., Giozza, W.F. *Redes locais de altíssima velocidade*. 9o.SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES. São Paulo, 1991.

[11] HANDERSON-SELLERS, B., EDWARDS, J.M. *The object-oriented systems life cycle*. Communication of the ACM. Vol. 33, No. 9, 1990.

[12] IEEE. *Project 802. Local area network*. IEEE Computer Society. IEEE 802.3 - CSMA/CD, IEEE 80.4 - Token Bus, IEEE 802.5 - Token Ring, IEEE 802.2 - LLC, 1985.

[13] KLEINROCK, L. *Queueing system*. Vol. 1: Theory. John-Wiley & Sons. 1985.

[14] KUEHN, P.J. *Multiqueue systems with nonexhaustive cyclic service*. B.S.T.J. Vol. 58, no. 3, March 1979. p. 671-698.

[15] MACNAIR, E.A. & SAUER, C.H. *Elements of practical performance modeling*. Prentice-Hall, Inc., 1981.

[16] MOURA, J.A.B. *Loops and ethernet: Evaluantion and comparison of performance and complexity*. MASc Thesis, University of Waterloo, Ontario, Canadá, 1979. 181 p.

[17] MOURA, J.A.B., SAUVÉ, J.P., GIOZZA, W.F. & ARAÚJO, J.F.M. *Redes locais de computadores (protocolos de alto nível e avaliação de desempenho)*. Editora McGraw-Hill, São Paulo, 1986.

[18] PINSON, L.J., WIENER, R.S. *An introdution to object-oriented programming and Smalltalk*. Addison-Wesley Publishing Company, 1988.

[19] PRESSMAN, R.S. *Software engineering*. Second Edition, McGraw-Hill International Editions, 1987.

[20] SCHRIBER, T.J. *Simulation using GPSS*. John Wiley & Sons. 1974.

[21] STROUSTRUP, B. *The C++ Programming Language. Reading*. Mass.: Addison-Wesley, 1986.

[22] WEINER, R.S. & PINSON, L.J. *Programação orientada para objeto e C++*. Trad. Andréa Nastri. Makron Books do Brasil Editora Ltda. 1991. 333 p.

[23] ZORTECH C++ COMPILER V2.1. *C++ Tools*. 1990. 379 p.