Núcleo *Multithreaded* para Agentes de Gerenciamento OSI: Implementação e Avaliação de Desempenho

Rivalino Matias Júnior[1] Paulo José de Freitas Filho[2]

Iwens Gervasio Sene Junior[3] Elizabeth S. Specialski[4]

{rivalino, freitas, iwens, beth}@ inf.ufsc.br

Curso de Pós-Graduação em Ciência da Computação

Universidade Federal de Santa Catarina

Cx. Postal: 476 - Campus Universitário - CEP: 88040-900

Florianópolis-SC/Brasil

Phone: +55 (48) 231-9738 Fax: +55 (48) 231-9770

ABSTRACT

Nos últimos anos, os requisitos de gerenciamento para redes de computadores, tem exigido que plataformas de gerenciamento sejam construídas levando-se em conta os atuais avanços tecnológicos, em função da grande variedade e capacidade dos elementos disponíveis nestes ambientes. Este trabalho apresenta a introdução de tecnologias de computação de alta performance no desenvolmento de aplicações de gerenciamento de redes.

Neste artigo, é apresentada uma proposta de um núcleo *multithreaded* para agentes de gerenciamento OSI/ISO [1], juntamente com sua modelagem, simulação e avaliação de desempenho.

Para efeitos de comparação com outros modelos de agentes, uma versão *unthreaded* será utilizada, a qual segue o mesmo modelo dos núcleos atualmente implementados por plataformas de gerenciamento comerciais e/ou de projetos acadêmicos.

Keywords

OSI Network Management, Agent Core, Multithreading, Active Objects, Modeling and Simulation

[1] Professor do Curso de Ciências da Computação da Universidade do Oeste de Santa Catarina (UNOESC), Campus de Chapecó-SC.
[3] Professor da Universidade Católica de Goiás
[2 4] Professores do Departamento de Informática e Estatística (INE) da Universidade Federal de Santa Catarina (UFSC)

1 - Introdução

Os objetivos do gerenciamento de redes são imutáveis, contudo, o processo pelo qual estes objetivos são alcançados, tem-se alterado de maneira a fazer uso dos atuais avanços tecnológicos. Como novas tecnologias e serviços estão sendo introduzidos, sistemas de gerenciamento de redes devem estar prontos para atender tais inovações, executando monitorações em tempo real, e fornecendo suporte para a execução de ações corretivas automáticas, baseadas nos eventos e no estado corrente da rede [2].

Atualmente, devido ao avanço dos sistemas operacionais, linguagens de programação e ambientes de suporte ao desenvolvimento de aplicações, a implementação de aplicações *multithreaded* tem se tornado freqüente em várias áreas (SGDBs, GUIs, Web Servers, etc.).

Este artigo apresenta um núcleo *multithreaded* para a implementação de agentes de gerenciamento OSI, o qual faz parte de um projeto maior que visa a definição e implementação de uma plataforma de gerenciamento de redes.

Com o objetivo de oferecer um núcleo básico para os processos agentes na plataforma, o modelo proposto permitirá que objetos gerenciados sejam implementados como objetos ativos, visto suas características *multithreaded*. Este núcleo, fornece aos processos agentes, características que propiciem seu maior desempenho e segurança no que se refere ao suporte de execução de seus objetos gerenciados, e na realização de atividades intra-agentes (manutenção da MIT (*Management Information Tree*), disseminação de notificações, etc.).

2 - Aplicações Agentes

Os documentos de padronizações, na área de gerenciamento de redes, não definem a organização interna dos processos de gerenciamento, sendo esta uma questão local de cada implementação. Conceitualmente, processos de aplicação de gerenciamento são aquelas aplicações que se utilizam dos serviços providos pelo elemento de serviço de aplicação de gerenciamento de sistemas (SMAE). Este conceito, definido em [3], pode ser ilustrado na Figura 1.

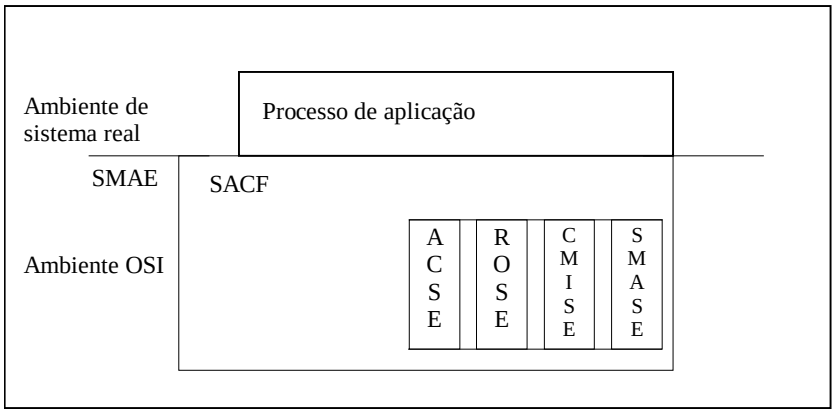


Figura 1 - Gerenciamento e a camada de aplicação [3].

Processos de aplicação são parte do ambiente real, não sendo escopo dos documentos de padronização, os quais somente tratam de componentes definidos dentro do ambiente OSI (elementos de serviços, protocolos de camada, etc.).

Agentes são processos participantes do gerenciamento de sistemas, que tem como objetivo servir e manter informações de gerenciamento a respeito de um determinado sistema gerenciado. Sendo assim, podemos dizer que agentes são servidores de informação de gerenciamento.

Para atender a estes requisitos, agentes devem monitorar e atuar (controlar) sobre recursos do sistema. Monitoração, significa a coleção dinâmica de informações relacionadas com os objetos, de *software* ou *hardware*, que estão sob observação [4]. Controle pode ser entendido como a execução das operações requisitadas pelo gerente sobre os objetos gerenciados.

Processos agentes devem ser estruturados de forma a minimizar os tempos de respostas às requisições dos gerentes e manter suas bases de informações de gerenciamento sempre atualizadas e consistentes.

A organização interna do código de cada componente dos processos agentes, juntamente com suas características de implementação, são particulares à cada plataforma, visto que estas questões não são definidas nos documentos de padronização.

Neste trabalho, aplicações agentes são implementadas como processos *daemons* [5] do sistema operacional UNIX, fazendo parte do seu código os seguintes componentes:

- a implementação da MIB (*Management Information Base*)
- os elementos de serviços ACSE[6], ROSE[6] e CMISE[7];
- o protocolo LPP[6];

No enfoque adotado para a implementação dos processos agentes, ao ser carregado para a memória o agente leva em seu código todas funções necessárias à sua execução. A Figura 2 apresenta a estrutura dos processos agentes na plataforma.

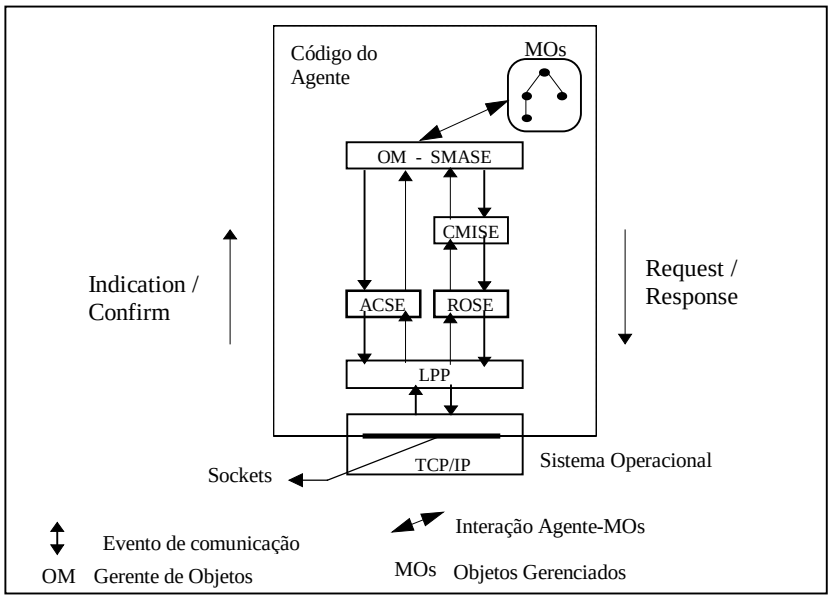


Figura 2 - Estrutura do código dos processos agentes na plataforma.

3 - Agentes e Objetos Gerenciados

Em cada processo agente, foi definido que estarão contidas todas definições das classes dos objetos gerenciados, o que permitirá instanciá-las em tempo de execução. A implementação dos objetos gerenciados, basicamente foi realizada representando-os como instâncias de classes da Linguagem C++.

Para fins de nomeação dos seus objetos gerenciados, os processos agentes implementam uma estrutura interna para a representação da MIT [8]. Esta estrutura permitirá que a representação dos objetos siga o conceito de hierarquia de *containment*, definida no

modelo de gerencia OSI. A implementação da MIT não está incorporada na implementação das classes de objetos gerenciados, sendo esta uma implementação independente, o que proporciona uma maior flexibilidade para sua manutenção (criação, consulta, deleção, etc.). Um objeto de suporte chamado OM (*Object Manager*), foi definido para realizar o gerenciamento da MIT e a manutenção do conjunto de instâncias de classes de objetos gerenciados suportadas pelo agente. Uma descrição mais detalhada do comportamento do OM será apresentada na seção 4.3.

4 - O Núcleo *Multithreaded*

Como foi dito anteriormente, objetos gerenciados são parte integrante do código dos processos agentes. Portanto, processos agentes devem suportar os requisitos básicos para a execução e manutenção destes objetos. O núcleo *multithreaded* que fornecerá suporte à execução dos objetos gerenciados, leva em consideração dois aspectos básicos:

- Aspectos de tratamento das operações de gerenciamento e emissão de notificações;
- Dinâmica (comportamento) dos objetos gerenciados.

Com relação ao tratamento das operações de gerenciamento e emissão de notificações, foi definida uma política de serialização destas operações. O atendimento das operações requisitadas pelo gerente é feito de maneira serializada, de acordo com a ordem de chegada destas operações (FCFS - *First-Come-First-Served*).

Como o núcleo *multithreaded* está inserido na estrutura apresentada na Figura 2, a serialização das operações de gerenciamento se dá no momento em que chegam PDU's do ACSE/CMISE. Neste ponto, a organização da estrutura de entrada do OM se encarrega de agrupar as operações que chegam, seguindo uma política FIFO (*first-in-first-out*). O objeto de suporte OM tem como objetivo dar manutenção na MIT, distribuir as operações que chegam do gerente para seus respectivos objetos gerenciados, realizar as operações de suporte à *multithreading*, implementar o controle de acesso aos objetos da MIB, dentre outras funções de gerenciamento da infra-estrutura.

Com relação ao tratamento das operações emitidas pelo gerente, foram estudadas algumas formas para estruturação dos processos agentes (servidores), sendo que uma das mais utilizadas para aumentar a performance destes processos, assemelha-se à implementação de um servidor de RPC *multithreading* [9]. Neste enfoque, para cada operação que chega uma *thread* é criada para a execução desta operação. Tal organização não foi adotada, visto que a serialização das operações não é garantida, e para os casos de operações realizadas sobre o mesmo objeto, o protocolo CMIP [10] requer a serialização destas operações, ao contrário de operações não sincronizadas sobre objetos diferentes, que podem ser executadas sem nenhum requisito de serialização, no que se refere à ordem de emissão das operações pelo gerente.

Um exemplo que invalida a utilização do enfoque adotado para o servidor de RPC *multithreading*, poderia ser a execução de operações não serializadas sobre o mesmo objeto gerenciado. Um gerente solicitando uma operação SET (*Replace-value*) sobre atributos de vários objetos gerenciados, utilizando escopo e filtro, em seguida emite uma operação GET para recuperar o valor de um atributo que foi alterado em um dos objetos gerenciados afetados na operação SET. A *thread* criada para a operação SET poderia perder o processador antes de ser concluída, possibilitando a *thread* da operação GET tomar o processador e possivelmente retornar o valor do atributo, o qual ainda não foi alterado pela operação SET, criando uma situação de erro.

Em função das exigências de serializações do protocolo CMIP, foi considerado que operações de gerenciamento serão tratadas seqüencialmente, tanto pelo OM quanto pelos objetos gerenciados, sendo que os dois tipos de objetos possuem filas de operações seguindo

uma política FIFO. A Figura 3 apresenta o modelo proposto para o núcleo multithreaded.

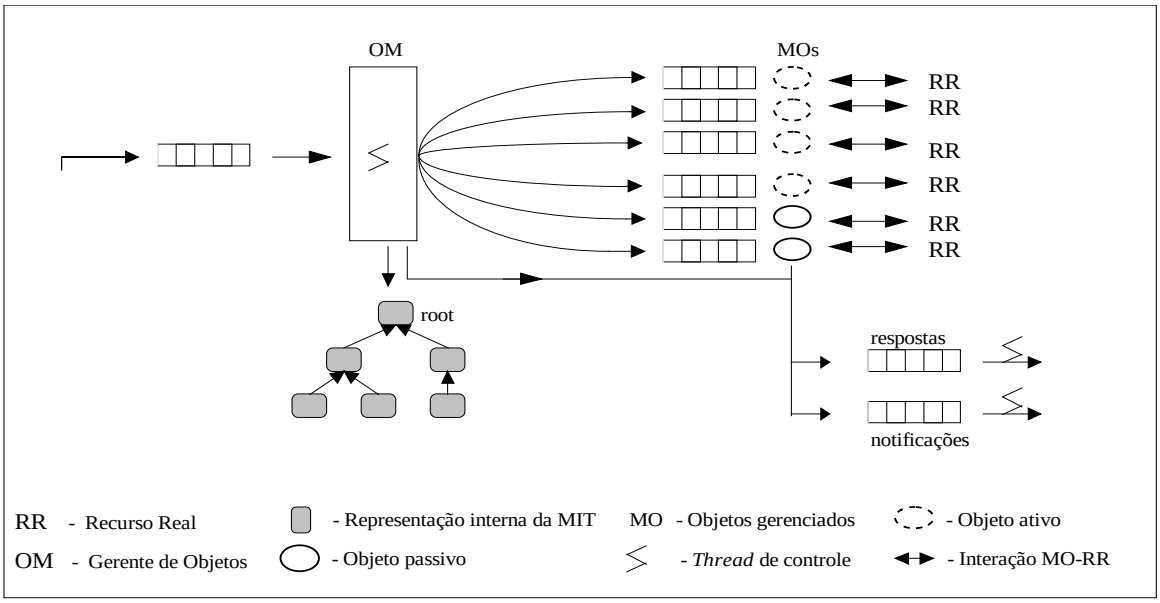


Figura 3 - Núcleo *multithreaded*.

Após chegar mensagens em sua estrutura de entrada, o OM não necessita ser invocado, pois ele possui sua própria *thread* de controle e constantemente estará consumindo mensagens de sua fila. Quando não existem mensagens para serem consumidas, o OM continua executando partes de seu comportamento (mudança de prioridade entre as *threads*, sincronização de operações, etc.) relativas ao gerenciamento dos vários objetos gerenciados de sua MIB.

Cada objeto possuindo sua própria *thread* de controle e fila de operações, possibilita que sua execução seja realizada de forma independente e concorrente com outras atividades do agente, incrementando a performance deste processo.

Para o propósito de independência e concorrência entre os objetos gerenciados, o sistema adota o conceito de objetos ativos (seção 4.1) na implementação dos objetos gerenciados.

Na seção 5, será apresentada uma avaliação de desempenho de duas implementações de núcleo de agentes, uma utilizando objetos ativos e outra que implementa objetos passivos.

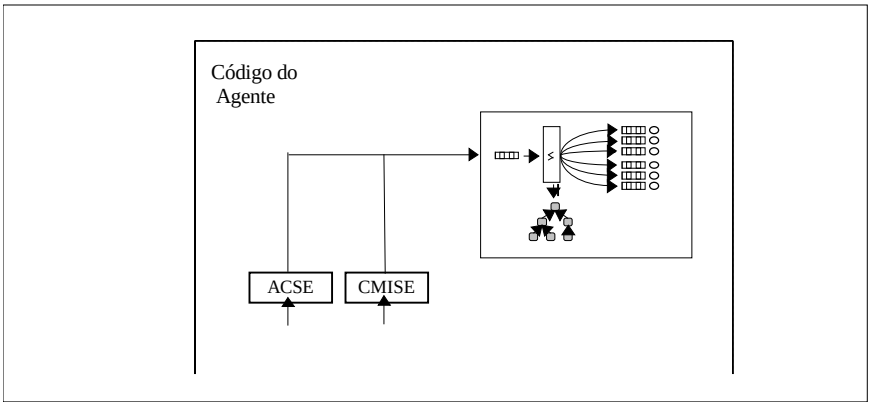


Figura 4-Localização do núcleo *multithreaded* no processo agente.

A Figura 4 apresenta a localização da infra-estrutura *multithreaded* no código do processo agente, de acordo com a estrutura apresentada na Figura 2.2.

4.1 - Objetos Ativos

No paradigma orientado a objetos, podemos classificar os estados de execução dos objetos basicamente em: **Ativos** ou **Passivos** [11]. Segundo Booch [12], concorrência em OOP é a propriedade que distingue um objeto ativo de um objeto não ativo (passivo). Na presença de concorrência, é apropriado que se visualize os objetos como entidades ativas autônomas, as quais executam seu comportamento independente de invocações externas [13].

Utilizando estes conceitos, pode-se dizer que objetos ativos são aqueles que encapsulam sua própria *thread* de controle, a fim de possuírem autonomia para mudarem seu estado interno sem a necessidade de invocações externas. Objetos passivos não encapsulam sua própria *thread* de controle, e só podem mudar seu estado na presença de uma invocação externa, que pode ser tanto por outro objeto ativo ou pela *thread* de controle do processo em que o objeto passivo faz parte. Após ser invocado, um objeto passivo torna-se ativo, realiza alguma computação (seu comportamento) e retorna ao estado passivo.

Algumas linguagens orientadas a objeto [14], fornecem suporte à implementação de objetos ativos, contudo, a linguagem C++ a qual está sendo utilizada na implementação da plataforma, somente implementa o conceito de objetos passivos. Em função desta limitação, foram utilizadas *threads* associadas com a linguagem C++, a fim de resolver esta restrição e permitir que o conceito de objetos ativos seja então implementado.

4.2 - Objetos Gerenciados como Objetos Ativos

Com o objetivo de melhor atender aos requisitos de gerenciamento, adotou-se a utilização de objetos ativos para implementar objetos gerenciados, visto que a representação de recursos reais através de objetos ativos, se encaixa melhor com a abstração de recursos do sistema (entidades de camada, conexões de transporte, hubs, etc.).

Enfoques tradicionais (objetos passivos), limitam a implementação de tais abstrações, devido a falta de dinâmica e independência entre a execução dos objetos. Problemas com um objeto (ex. falha na interação com um recurso real), comprometem a execução dos outros objetos, pois a *thread* de controle do processo está dedicada à execução do objeto que falhou, causando o bloqueio de todo processo agente. Para resolver este problema, algumas plataformas restringem a implementação dos objetos, pois não são permitidos bloqueios em eventos externos ou até mesmo executando uma computação mais complexa. Um exemplo de tais plataformas, é a plataforma de gerência OSIMIS [15], a qual define que implementações de objetos gerenciados não podem se bloquear em pontos externos de comunicação, causando o bloqueio de todo processo agente.

Restrições como esta torna a implementação do comportamento dos objetos mais complexa, visto que de todo processo agente pode ser comprometido em função de uma situação de erro em um de seus objetos. Esta complexidade pode ser aumentada, quando se tem objetos gerenciados representando recursos do sistema os quais estão fracamente acoplados, o que exige maior complexidade na comunicação recurso real e objeto gerenciado.

Implementando objetos gerenciados como objetos ativos, o objeto que se bloqueou não afeta os outros objetos, nem as atividades do próprio processo agente, visto que suas *threads* individuais estão livres para executarem suas computações independentes do objeto que se bloqueou.

Objetos ativos refletem com maior fidelidade a principal função dos objetos gerenciados; representar recursos reais [16].

O núcleo proposto (Figura 3) não impede a utilização de objetos passivos, pois em alguns casos, objetos gerenciados possuem um comportamento estático e serão melhor implementados como objetos passivos. Um exemplo para objetos gerenciados implementados como entidades não ativas, são objetos da classe *log*, os quais somente realizam suas funções mediante invocações externas (gravação, leitura, etc.). Nestes casos, é desejável que o objeto seja implementado como passivo, pois sua atividade constante, somente iria consumir recursos de processamento (CPU) sem necessidade.

Como dito anteriormente, objetos passivos podem se bloquear e comprometer a operação de outros objetos passivos ou da *thread* de controle que o invocou. Para isto, foi proposta uma implementação de objetos passivos, a qual permite que um eventual bloqueio em função de seu comportamento não comprometa a execução de outras atividades no processo agente. Esta implementação foi possível, dadas as características *multithreaded* do núcleo proposto e será abordada na seção 4.5.

4.3 - Dinâmica dos Objetos

A execução concorrente dos objetos gerenciados, proporciona ao sistema gerente, menor tempo de resposta com relação às operações requisitadas, e uma visão mais precisa da MIB, no que se refere aos estados dos objetos gerenciados e os estados dos recursos que estes representam [17].

Basicamente, objetos gerenciados podem obter informações de seus recursos acessando-os de três formas:

- mediante requisição do gerente (acesso por demanda);
- através de *polling* periódicos;
- recebendo alarmes (*traps*) de recursos com funções de gerência embutidas.

No primeiro caso, organizações tradicionais (*single threaded*) causariam o bloqueio temporário de toda aplicação agente, em função do objeto estar interagindo com um recurso naquele dado instante. Caso este recurso estivesse fracamente acoplado ao sistema, várias requisições posteriores vindas do gerente poderiam ficar pendentes por um longo período, esperando o término da comunicação entre o objeto gerenciado e o recurso real. Utilizando o núcleo *multithreaded*, todas as atividades do agente são independentes, proporcionando maior justiça em suas execuções e não permitindo que computações ativas em um dado instante, monopolizem a utilização do processador.

No segundo caso de interação (acesso por *polling*), tanto organizações tradicionais quanto a organização *multithreaded*, fornecem um suporte desejável para sua implementação. Contudo, no enfoque *multithreaded* tem-se a facilidade de fornecer a determinados objetos gerenciados maiores prioridades com relação às suas execuções, causando uma maior freqüência de suas operações de *polling*. Isto é desejável, pois em alguns recursos gerenciados atualizações mais precisas são necessárias a fim de suportarem características de tempo real. Em aplicações agentes, atuando no gerenciamento de faltas, esta facilidade é desejável no sentido de proporcionar uma maior precisão na detecção de problemas que venham a ocorrer com determinados recursos que são vitais para o perfeito funcionamento do sistema.

Na terceira interação (recepção de *traps*), semelhante ao caso anterior, determinados recursos que exigem uma atenção especial (ex. serviço de controle de acesso), podem ser privilegiados em ter suas notificações atendidas de forma prioritária a outros objetos gerenciados que exigem menor atenção com relação à suas notificações.

A função de alterar prioridades entre os objetos gerenciados é delegada ao OM, o qual interfere diretamente nas prioridades do conjunto de *threads* utilizadas pelos objetos gerenciados. O projetista da aplicação agente, será encarregado de definir quais prioridades

serão atribuídas para cada objeto.

Com o objetivo de incrementar ainda mais a performance do processo agente, objetos podem passar a sua vez de execução para outras entidades ativas (*threads*) do sistema, verificando que em um dado instante, seu comportamento não realizará nenhuma computação. Filas vazias, atributos de estado operacional desabilitados (*disable*), recursos gerenciados temporariamente indisponíveis, são alguns exemplos de condições para que os objetos repassem o processador para outras atividades, a fim de não consumirem toda sua fatia de tempo de execução sem nenhuma computação.

A serialização das operações se dá ao nível de operações sobre o mesmo objeto. Requisições que se utilizam de escopo, filtro e sincronização, devem previamente serem tratadas pelo OM, pois além de gerenciar a estrutura que representa a MIT (*scope*, *filter*), este fornece mecanismos para a implementação de sincronizações entre objetos, dentre outras funcionalidades.

No modelo apresentado pela Figura 3, tem-se uma *thread* dedicada à execução do OM, e outras duas para cada uma das filas de respostas e notificações, as quais são dedicadas à emissão destas mensagens. É função do OM realizar a manutenção (criar, deletar, suspender, reativar, alterar prioridades, etc.) destas *threads* e das *threads* de cada objeto gerenciado, ao nível da aplicação agente.

Na criação de um objeto, o OM associa ao objeto criado uma *thread* de controle, atualiza a MIT, inicializa os valores dos atributos deste objeto com algum objeto de referência (se especificado no *Name Binding* [18]), cria uma fila de operações para o objeto, emite um relatório de eventos notificando a criação, e atualiza seu estado interno a fim de suportar o novo objeto criado. A *thread* associada ao novo objeto, executará o método "*behaviour()*" deste objeto, sendo que parte deste comportamento é definido pela plataforma e o restante será aquele definido no construtor (BEHAVIOUR) do *template* da classe de objeto gerenciado [18], o qual será implementado pelo projetista da aplicação. A Parte do comportamento do objeto, fornecido pela plataforma, está relacionado com o tratamento da fila de operações do objeto gerenciado e com interações com objetos de sincronização (*S*), sendo esta parte da implementação denominada "código de suporte aos objetos ativos". A funcionalidade do objeto (*S*) será descrita na seção 4.4.

4.4 - Operações que Exigem Sincronização

Cada objeto tendo sua própria fila de operações e executando de forma independente e concorrente com outros objetos do sistema, cria um problema em situações onde uma operação deve ser executada sobre múltiplos objetos (*scope*) e esta requer sincronização atomica. O protocolo CMIP permite duas formas de sincronização:

- *atomic*;
- *best effort*.

Na sincronização *best effort*, a falha na execução da operação por algum objeto do grupo, não interfere na execução dos outros objetos. A sincronização *atomic* exige que todos objetos do grupo tenham sucesso na execução da operação; caso algum objeto falhe, todos objetos envolvidos abortam a operação mesmo sendo capazes de executá-la. O problema com este tipo de operação, pode ser ilustrado no exemplo a seguir.

Suponha-se que para uma operação de gerenciamento SET (S), solicitada com escopo e filtro, três objetos foram selecionados, tendo sido esta operação requisitada com sincronização *atomic*. A Figura 5 ilustra o cenário do problema.

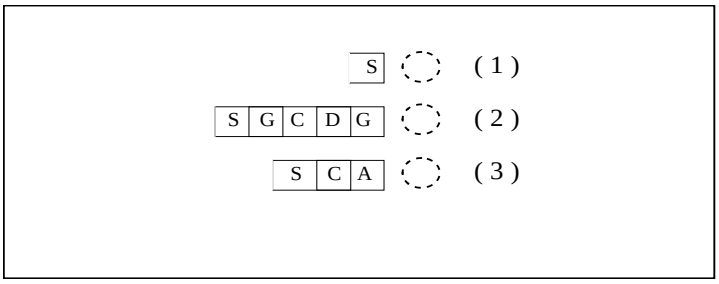
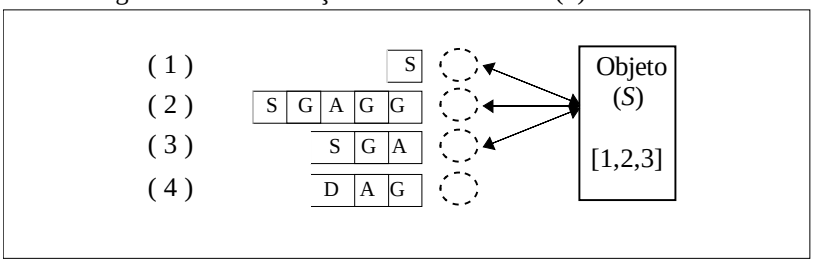


Figura 5 - Cenário para o problema de sincronização *atomic*.

Como o objeto (1) não possui mensagens de operações em sua fila, a operação SET será a única da fila de operações deste objeto. No caso do objeto (2), existem operações pendentes, e o SET será a quinta operação da fila. O terceiro objeto possui somente duas operações em sua fila, e a operação SET será a terceira a ser atendida. Neste cenário, verifica-se o problema de que a operação (S), poderá ser executada, pelos objetos em tempos diferentes, visto que a fila de operações de cada objeto possui tamanhos diferentes. Para resolver o problema, um objeto de suporte à aplicação, denominado objeto sincronizador (*S*), coordenará operações que exijam sincronismo atômico. A Figura 6 apresenta a solução.

Figura 6 - Sincronização com coordenador (*S*) centralizado.



Quando operações que exigem síncronismo *atomic* forem submetidas aos objetos gerenciados, o OM criará um objeto (*S*) o qual conterá os identificadores dos objetos (OID) que estão no grupo da operação. Neste exemplo, somente os objetos 1,2 e 3 estão sendo submetidos à sincronização atômica, sendo que seus identificadores estão contidos no objeto (*S*). Quando o objeto gerenciado retirar uma operação de sua fila, a qual exige sincronização com outros objetos, este notificará ao seu objeto (*S*), que está pronto para executar a operação. A partir deste momento o objeto gerenciado espera por uma mensagem do objeto (*S*), solicitando a sua execução da operação. Esta mensagem somente será emitida pelo objeto (*S*) quando todos objetos pertencentes ao grupo estiverem prontos para executar a operação. Sendo assim, novas operações que chegam na fila do objeto, não são tratadas, até que ele seja liberado pelo objeto (*S*). Outras atividades podem ser executadas pelo objeto gerenciado, a fim de manter sua imagem do recurso real sempre atualizada.

Após o objeto (*S*) receber de todos objetos do grupo, notificações de que estão prontos, este envia uma mensagem para todos objetos executarem a operação. Após executarem a operação, os objetos gerenciados devem enviar ao objeto (*S*) uma resposta de que a operação foi realizada com sucesso ou falha, pois na execução da operação sobre os recursos gerenciados (recursos reais), podem ocorrer problemas na interação (objeto gerenciado-recurso). Caso alguma resposta de falha seja notificada, a transação é cancelada para todos objetos do grupo, que após retornarem para seu estado anterior, são liberados para continuarem a servir suas filas de operações.

4.5 - Objetos Passivos não Bloqueantes

A fim de permitir que a implementação de alguns objetos gerenciados (*log*, registro de *log*, etc.), modelados como objetos passivos, não comprometa outras atividades do sistema caso estes venham a se bloquear, foi definido que cada objeto passivo possuirá sua própria *thread* de controle. Estes objetos somente se tornarão ativos na presença de invocações externas, preservando a semântica de objetos passivos.

Nesta implementação, o OM não acessa diretamente o método do objeto passivo, visto que se o método se bloquear por algum motivo, a *thread* do OM será bloqueada e não poderá atender às novas PDUs que chegam do CMISE/ACSE. Portanto, o OM solicita que a *thread* do objeto passivo invoque seus métodos para atender as operações de sua fila, não afetando assim a execução de outras atividades do agente, caso o objeto passivo venha a se bloquear. A Figura 7 ilustra conceitualmente a semântica do comportamento dos objetos passivos e ativos, ambos com suas próprias *threads* de controle.

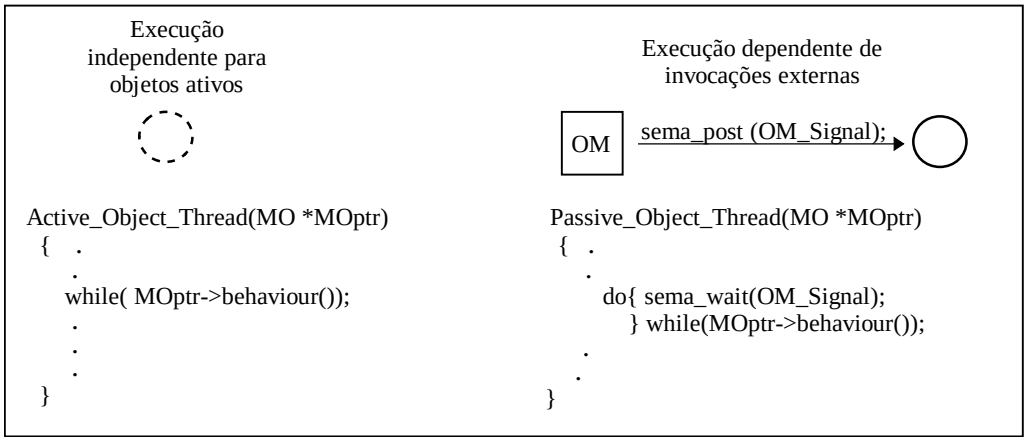


Figura 7 - Comportamento das *threads* em objetos passivos e ativos.

Como pode ser observado, objetos passivos executam seu comportamento mediante invocação do OM, a qual está sendo representada pelo incremento (*sema_post()*) de uma variável semáforo (Sinal_do_OM), utilizada pelo objeto gerenciado passivo, ao contrário de objetos ativos que estão constantemente executando seu comportamento.

5 - Modelagem, Simulação e Avaliação de Desempenho

Esta seção apresenta os resultados da comparação do desempenho de dois modelos de núcleo de agente. Um modelo adota a implementação de objetos gerenciados como objetos passivos, sendo esta a forma adotada pela maioria das plataformas de gerenciamento, hoje implementadas. No segundo modelo, a implementação dos objetos gerenciados segue a abordagem de objetos ativos, a qual é proposta deste trabalho.

A avaliação de desempenho dos agentes de gerênciamento requer a construção de modelos, cuja simulação forneça resultados relativos ao comportamento das variáveis que medem seu desempenho. Tais modelos, quando expostos à cargas artificiais, análogas àquelas do mundo real, permitem a comparação da performance dos sistemas sob investigação. Os principais componentes do modelo são:

- Fontes de emissão de operações de gerênciamento (gerente)
- Sistema gerenciado (agente)
  - Objeto gerenciado (MO)
  - Gerente de objetos (OM)
  - Relatório de eventos (notificações)

5.1 - Parâmetros Utilizados

Os parâmetros utilizados para a modelagem dos componentes acima, foram obtidos a partir da realização de medições efetuadas em protótipos, os quais foram implementados para cada um dos modelos avaliados. As operações de gerenciamento (requisições) são responsáveis por 85% do tráfego de entidades a serem processadas pelo agente. Os 15% remanescentes estão relacionados às notificações geradas pelos objetos gerenciados. Assume-se que as chegadas, tanto de requisições quanto de notificações, aderem a um processo de Poisson. Desta forma, para a modelagem dos processos de geração de notificações e requisições, foram utilizadas distribuições exponenciais.

Para os tempos de processamento do OM, segundo as medições realizadas nas duas implementações, foram adotados os valores 0,073 seg. e 0,075 seg., respectivamente, para os modelos de Objeto Passivo e Objeto Ativo.

Na implementação de ambos os protótipos, foi definido que o tempo de processamento dos MOs tem duração de 2 segundos. As medições de ambos os modelos, forneceram o tempo total de 2,0094 seg. para MOs implementados de forma passiva, e 2,0508 para MOs implementados como objetos ativos.

5.2 - Projeto de Experimentos

O desempenho dos sistemas modelados depende, fundamentalmente, de quatro fatores principais: número de gerentes, número de objetos, formas de interação agente-gerente e nível de carga do sistema. Cada um destes fatores poderá assumir dois possíveis níveis, caracterizando-se, portanto, um projeto de experimentos do tipo fatorial completo [19], com um número de experimentos igual a 2^4. Os quatros fatores e seus respectivos níveis são apresentados na Tabela 1.

Fator	**Nível 1 (+)**	**Nível 2 (-)**
Número de Gerentes	5	1
Número de Objetos	30	10
Nível de Carga	2,2 seg. (objetos ativos)	1,1 seg. (objetos ativos)
	12,44 seg. (objetos passivos)	24,88 seg. (objetos passivos)
Interação Agente-Gerente	Assíncrono	Síncrono

Tabela 1: Fatores e Níveis utilizados nos experimentos

Como pode ser observado na tabela acima, a alocação das cargas, dentro de um mesmo nível, é diferenciada dependendo do modelo adotado. Os valores utilizados, refletem uma compatibilização entre os pontos de estrangulamento diferenciados nos dois modelos, OM no modelo de objetos ativos e MO no modelo de objetos passivos. Desta forma, para caracterizar os modelos em situações de alta e baixa taxa de processamento de suas atividades críticas, adotou-se os parâmetros mostrados na Tabela 1.

Para avaliar o desempenho dos sistemas modelados, as seguintes variáveis de respostas foram adotadas:

- Tempo total gasto por uma requisição no sistema;
- Número de requisições atendidas em ambos modelos;
- Número total de notificações contabilizadas.

5.3 - Resultados das Simulações

Os resultados apresentados aqui foram obtidos apartir das simulações dos modelos

dos sistemas (ativos e passivos). Os modelos foram desenvolvidos usando o ambiente de simulação ARENA 2.0 [20]. Para a análise estatística dos resultados das simulações foi utilizado *Output Analyzer* do ARENA.

Os resultados apresentados são decorrentes de valores obtidos a partir de 10 replicações de cada um dos experimentos. Este número demostrou ser suficiente para que as médias obtidas se encontrem dentro de um intervalo de confiança aceitável ao nível α de 0,05. As simulações foram realizadas considerando-se intervalos de 1000 seg.. Por se tratarem de sistemas não-terminais, suas análises devem ser realizadas com os sistemas sob regime. Para tanto, após procedimentos estatísticos adequados, considerou-se a adoção de um período de *warm-up* equivalente a 10 % do tempo total de cada replicação. Desta forma, foram simulados períodos de 1100 seg., onde as observações referentes aos 100 seg. iniciais foram descartadas.

5.3.1 - Análise dos Resultados do Modelo de Objetos Passivos

Os resultados das simulações, para o modelo de objetos passivos, encontram-se na Tabela 2. Para cada um dos experimentos, apresenta-se os valores adotados para cada fator, bem como, os resultados obtidos pelas variáveis de resposta. Na tabela, NRA significa o número de requisições atendidas pelo OM, NNA o número de notificações atendidas pelo OM, e TTR o tempo total despendido por uma requisição no sistema

		Níveis:	(+)	(-)
Fatores:	1 - N.º Gerentes		5	1
	2 - N.º Objetos		30	10
	3 - Carga de Requisições		12.44 s	24.88 s
	4 - Iteração		assíncrono	síncrono

Experimentos	**1**	**2**	**3**	**4**	**NRA**	**TN**	**TTR**
1	+	-	+	+	**383**	29	2.11
2	+	+	+	+	**397**	**75**	2.18
3	+	+	-	+	223	59	2.37
4	+	-	+	-	**430**	27	13.63
5	+	+	+	-	**403**	**69**	**64.62**
6	+	+	-	-	223	59	4.5
7	-	+	+	+	70	**67**	2.11
8	-	+	+	-	70	**67**	2.23
9	+	-	-	-	206	39	3.03
10	+	-	-	+	183	26	2.11
11	-	-	-	-	31	24	2.26
12	-	-	-	+	31	24	2.11
13	-	-	+	-	70	18	2.26
14	-	-	+	+	70	18	2.11
15	-	+	-	+	104	40	2.11
16	-	+	-	-	104	40	2.16

Tabela 2 – Resultados obtidos do modelo Objetos Passivos

Considerando as observações da Tabela 2 e o Gráfico 1, percebe-se, no experimento 5, um aumento considerável (64.62 seg.) no tempo de resposta. Justifica-se este aumento em função dos níveis adotados pelos diversos fatores. O tempo de resposta de uma requisição para o modelo de objetos passivos síncrono, é dependente do tempo de espera da execução de outras requisições em trânsito pelo sistema. Neste caso, em função do alto nível de

requisições geradas pelos 5 gerentes, pelo grande número de notificações geradas pelos 30 objetos e, pela forma de interação agente-gerente (forma síncrona), o número de operações a espera de atendimento pelo OM, atinge seu ponto máximo, ocasionando um alto tempo de resposta por operação. Nos demais casos, o sistema apresenta-se menos congestionado, principalmente devido a ausência da combinação fator 3, nível (+) com fator 4 nível (-).

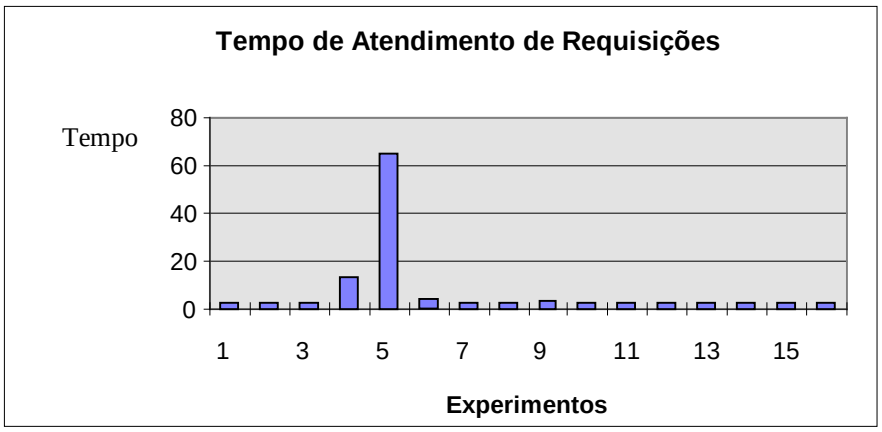


Gráfico 1

Quanto ao número de requisições atendidas, observa-se na Tabela 2 e no gráfico 2, que os ensaios 1, 2, 4 e 5 apresentam máximos (383, 397, 430 e 403, respectivamente). Estes valores, são devidos, principalmente, ao alto número de requisições geradas pelos 5 gerentes com alta taxa de criação de operações (fatores 1 e 3 (+)). No entanto, observa-se, também, que especialmente no experimento 5, devido ao grande número (69) de notificações geradas pelos 30 objetos, o tempo de processamento das operações é também o mais alto entre todos os experimentos.

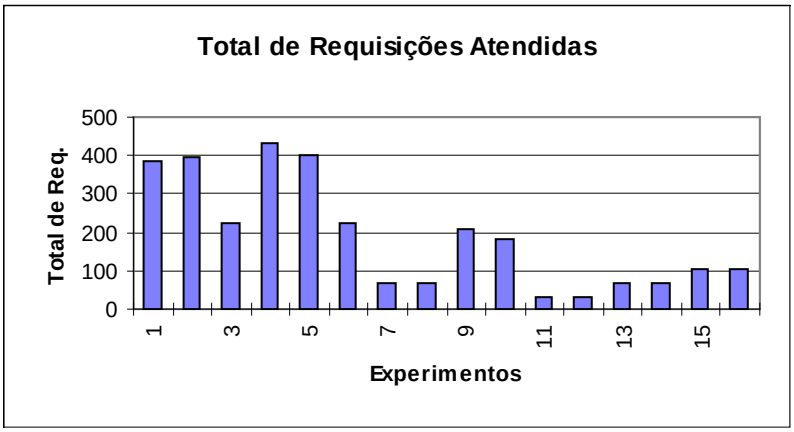


Gráfico 2

5.3.2 - Análise dos Resultados do Modelo de Objetos Ativos

Os resultados das simulações, para o modelo de objetos ativos, encontram-se na Tabela 3. Para cada um dos experimentos, apresenta-se os valores adotados para cada fator, bem como, os resultados obtidos pelas variáveis de resposta. Na tabela, NRA significa o número de requisições atendidas pelo OM, NNA o número de notificações atendidas pelo OM e TTR, o tempo total despendido por uma requisição no sistema

		(+)	(-)
Fatores:	1 - N.º Gerentes	5	1

2 - N.º Objetos				30	10		
3 - Carga de Requisições				1.1. s	2.2 s		
4 - Iteração				assíncrono	síncrono		
Experimentos	**1**	**2**	**3**	**4**	**NRA**	**NNA**	**TTR**
1	+	-	+	+	**3219**	34	15.21
2	+	+	+	+	**3420**	97	3.13
3	+	+	-	+	1656	98	2.57
4	+	-	+	-	1283	34	**157.79**
5	+	+	+	-	1529	97	**135.96**
6	+	+	-	-	1512	98	34.849
7	-	+	+	+	699	97	**2.39**
8	-	+	+	-	344	97	117.42
9	+	-	-	-	1220	102	79.39
10	+	-	-	+	1677	37	3.79
11	-	-	-	-	341	35	16.041
12	-	-	-	+	350	35	**2.41**
13	-	-	+	-	342	32	128.19
14	-	-	+	+	683	32	2.70
15	-	+	-	+	489	**107**	**2.28**
16	-	+	-	-	321	**107**	9.67

Tabela 3 – Resultados obtidos do modelo Objetos Ativos

Observando-se a Tabela 3 e os Gráficos 3 e 4, percebe-se que, de maneira geral o número médio de requisições atendidas cresce em relação ao modelo anterior (passivo). Examinando-se mais acuradamente os resultados, pode-se verificar que fatores mais influenciam no desempenho geral do sistema, considerando-se a combinação das variáveis de resposta NRA e TTR. Neste sentido, o experimento 2 (NRA = 3420 requisições e TTR = 3,13 seg.) pode ser considerado o de maior desempenho. Neste experimento, tem-se um sistema com máxima taxa de utilização dos MO's (em função dos níveis adotados nos diversos fatores) e atuando de forma assíncrona.

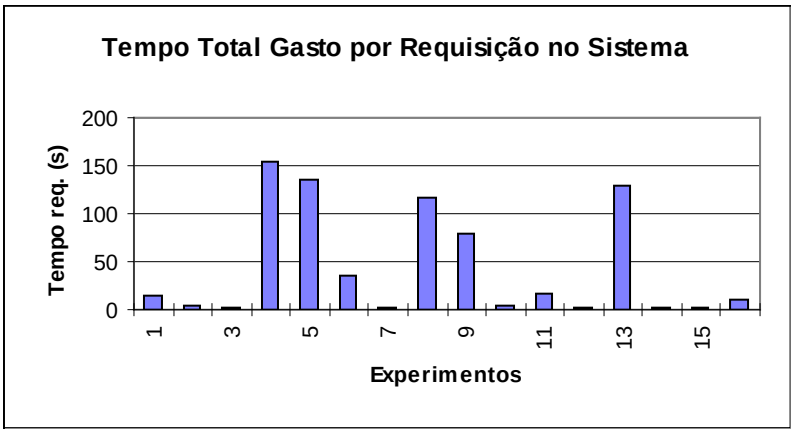


Gráfico 3

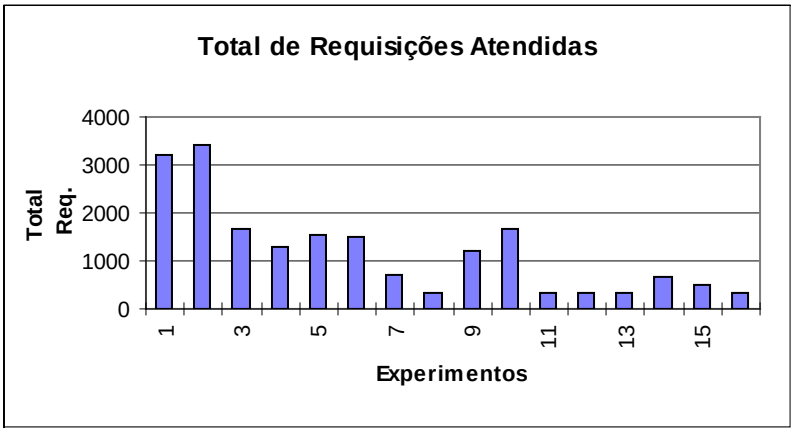


Gráfico 5

6 - Conclusões

O núcleo *multithreaded* apresentadado neste artigo, permite a introdução do conceito de objetos ativos na implementação de objetos gerenciados.Uma vantagem deste núcleo sobre as implementações de núcleos que utilizam o enfoque *single-thread*, é a grande independência entre as execuções dos comportamentos dos objetos gerenciados, o que propicia um alto grau de tolerância a faltas para o processo como um todo, protegendo-o de faltas que possam ocorrer no escopo de um objeto e que venham interferir em outras atividades do agente. A maior performance na execução das atividades intra-agente também é outro importante fator presente na proposta *multithreaded*. Os resultados obtidos a partir dos modelos simulados, comprovam o melhor desempenho desta proposta, quando comparada com o modelo *tradicional* que implementa objetos passivos.

Em máquinas multiprocessadas, a natureza das atividades intra-agente, permite a execuçao paralela das várias *threads* de controle, ocasionando um alto desempenho em suas execuções.

Este núcleo está atualmente implementado como parte de um *Agent Toolkit*, o qual tem como objetivo automatizar o processo de desenvolvimento de processos agentes, ficando a cargo do usuário somente a implementação dos comportamentos (*behaviours*) dos objetos gerenciados. Este *Toolkit* faz parte de um projeto maior, o qual visa a implementação de uma plataforma OSI de gerenciamento de redes. A implementação da plataforma está sendo desenvolvida para o Solaris 2 (SunOS 5.5). Vários recursos (mutex, semaphore, LWPs, etc.) pertencentes a este ambiente estão sendo utilizados. Toda manipulação de *threads* está sendo realizada usando a *pthreads library* [21] de maneira a tornar a implementação compatível com o padrão POSIX P1003.4a.

Futuros trabalhos terão como objetivo, a integração deste núcleo em agentes que utilizam interfaces XOM/XMP, visto a larga utilização destas APIs em plataformas de gerenciamento comerciais.

Referencias

[1] ISO/IEC 10040, Information Technology - Open Systems Interconnection - Systems management overview, 1992.
[2] Bean, A.; Wood, D.; Fairclough, W. "Specifying Goal-Oriented Network Management Systems", IEEE Communications Magazine, 1993.
[3] ISO/IEC 7498-4, Information Technology - Open Systems Interconnection - Management framework, 1992.
[4] Mansouri-Samani, M.; Sloman, M. "Monitoring Distributed Systems (A Survey)", Imperial College Research Report N°. DOC92/23, 1992.
[5] Bach, M. J. "The Design of the UNIX Operating System", 1990.
[6] Rose, M. T. "The Open Book: A practical perspective on OSI", Prentice-Hall, 1990.

[7] ISO/IEC 9595, Information Technology - Open Systems Interconnection - Common management information service definition, 1991.
[8] ISO/IEC 10165-1, Information Technology - Open Systems Interconnection - Structure of management information: Management information model, 1992.
[9] Tanenbaum, A. S. "Distributed Operating Systems", Prentice-Hall, 1995.
[10] ISO/IEC 9596, Information Technology - Open Systems Interconnection - Common management information protocol - part 1: Specification, 1991.
[11] Tsichritzis, D.; Nierstrasz, O.; Gibbs, S. "Beyond Objects: Objects ", IJICIS, vol. 1 no. 1, pp. 43-60, 1992.
[12] Booch, G. "Object-Oriented Analysis and Design with Applications - Second Edition", The Benjamin/Cummings Publishing Company, 1994.
[13] Löhr, K. "Concurrency Annotations", OOPSLA'92, pp 327-340, 1992.
[14] Takashio, K.; Tokoro, M. "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", OOPSLA'92, pp. 276-294, 1992.
[15] Pavlou, G.; McCarthy, K.; Bhatti, S.; Knight. G. "The OSIMIS Platform: Making OSI Management Simple",1994.
[16] Matias Jr., R.; Specialski, E. S. "Managed Objects as Active Objects: A Multithreaded Approach.", IS&N'97 (4th International Conference on Intelligence in Services & Networks), Maio/1997, Como, Italia.
[17] Matias Jr., R.; Specialski, E. S. "A Multithreaded Core for Network Management Agents", ISCC'97 (IEEE Symposium on Computers and Communications), Julho/1997, Alexandria, Egito.
[18] ISO/IEC 10165-4, Information Technology - Open Systems Interconnection - Structure of management information: Guidelines for the definition of managed objects, 1992.
[19] JAIN, R. 1991. *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc.
[20] PEGDEN, C.D., Shannon, R.E. Sadowski, P.P. 1995. *Introduction to Simulation Using SIMAN*, 2nd Ed, McGrall-Hill
[21] SunSoft "Pthreads and Solaris threads: A comparison of two user level threads APIs", Sun Microsystems, 1994.