MomentA: Gerenciamento de Serviços Usando Agentes Móveis em Ambiente CORBA

B.Schulze[1,2], E.R.M.Madeira[1], P.Ropelatto[1]
[schulze | edmundo | ropelatt@dcc.unicamp.br]

1. IC / Unicamp, CP 6176, 13083-970, Campinas, S. Paulo, fax: (019) 788-5847
2. CBPF / CNPq, R.Xavier Sigaud 150, 22290-180, Rio de Janeiro

Resumo

Este trabalho explora o paradigma de mobilidade no contexto de gerenciamento de monitoração e configuração em sistemas distribuídos abertos. Um conjunto de agentes é definido para explorar o ambiente gerenciado utilizando uma abordagem de detalhamento sucessivo de potenciais problemas. A implementação considera sistemas distribuídos baseados em objetos segundo OMG/CORBA. Uma extensão do trabalho é proposta para o ajuste do sistema gerenciado, utilizando o balanceamento dos recursos computacionais com o auxílio de um *serviço de disponibilidade*.

Abstract

This paper explores the mobility paradigm in the context of monitoring and configuration management of open distributed systems. A pool of agents is defined in order to explore the managed environment based on the zooming of potential problems. The implementation considers an open distributed environment based on OMG/CORBA objects. An extension is presented for adjusting the managed system based on the balancing of the computational resources with the help of an *availability service*.

Palavras-chave: gerenciamento distribuído, configuração, mobilidade, disponibilidade.

1. Introdução

O trabalho explora mobilidade de computação no âmbito de gerenciamento de sistemas distribuídos abertos e apresenta uma arquitetura com agentes móveis. As vantagens são consideradas do ponto de simplicidade e adaptabilidade do gerenciamento aos novos ambientes gerenciados ao invés da otimização de desempenho em problemas de soluções bem estabelecidas.

O modelo tradicional envolve processos estáticos transferindo dados e/ou comandos. Os dados e os comandos são a parte móvel de uma computação e os programas são a parte estática. Um crescente número de cenários não pode ser abordado de forma eficaz por tais paradigmas de interação estática. Um paradigma alternativo é o de agentes móveis, i.e., um programa identificado de forma única que pode migrar de máquina para máquina em um ambiente heterogêneo. Uma agência existe em cada máquina destino para receber e executar os agentes. A expectativa de desempenho maior está em redes de largura de banda estreita e de latência elevada.

Do ponto de vista arquitetural, gerenciamento pode variar de componentes inteiramente estáticos a componentes inteiramente móveis. Os esquemas estáticos são baseados em um controle de gerenciamento e recursos distribuídos ou em um controle com agentes distribuídos e associados aos recursos controlados. Os esquemas móveis são baseados em um controle estático e agentes móveis que se conectam aos recursos controlados, ou em um controle móvel com agentes móveis que se conectam a recursos móveis.

São apresentados resultados do ponto de vista de uma implementação e de um estudo de caso de gerenciamento. A implementação é baseada em um *middleware* OMG/CORBA [1,2] e o ambiente controlado é um mundo de objetos CORBA, isto é, objetos de aplicação e objetos de serviço [3,4,5,6]. A definição do problema é apresentada dos pontos de vista topológico, funcional e operacional. Do ponto de vista topológico, a população de objetos se encontra aglomerada em pequenos conjuntos (*clusters*) e distribuída no ambiente controlado. Em sistemas orientados a processo, estes conjuntos são processos. O uso de agentes móveis simplifica a visão topológica de gerenciamento dos objetos distribuídos.

Do ponto de vista funcional, os agentes móveis alteram dinamicamente a funcionalidade de gerenciamento e recolhem informação. Do ponto de vista operacional, uma solução de compromisso entre latência e velocidade de processamento é mover os dados em direção ao código, ou o código em direção aos dados - *caching* inverso [7]. Manter os dados próximos de onde são consumidos é eficaz quando há um grau elevado de localidade no uso e de coerência, sendo impróprio para dados distribuídos altamente voláteis, que se desatualizam rapidamente e tornam a *cache* inconsistente. Em resumo, mover código para perto dos dados pode reduzir excedentes de comunicação de dados enquanto mover o código a destinos remotos pode minimizar a ausência de recursos computacionais limitados.

Soluções baseadas em tecnologia de código não móvel podem sempre ser propostas. Entretanto, tarefas como gerenciamento e recuperação de informação se ajustam naturalmente ao saltar-executar-saltar de agentes móveis. Um agente migra para uma máquina, executa uma tarefa, migra para outra, executa outra tarefa dependente da saída precedente, e assim por diante.

O objetivo do trabalho é gerenciar aplicações baseadas em um middleware objeto-orientado que seja baseado em CORBA. Um conjunto de agentes é definido para explorar o ambiente controlado baseado no *detalhamento* de problemas potenciais. Uma extensão desta abordagem é proposta para permitir ajustes no sistema gerenciado. Este ajuste é baseado no balanceamento dos recursos computacionais com a ajuda de um *serviço de disponibilidade*.

Estrutura do Trabalho. A seção 2 apresenta uma visão geral de metodologias em gerenciamento distribuído e agentes móveis autônomos. A seção 3 descreve a arquitetura de gerenciamento *MomentA* e seus blocos básicos. A seção 4 discute aspectos e tecnologias da implementação. A seção 5 resume resultados enquanto a seção 6 apresenta observações finais.

2. Metodologia

Nesta seção exploramos aspectos dos diferentes esquemas de gerenciamento, o paradigma de agentes móveis em gerenciamento e uma proposta de ajuste usando migração de agentes.

2.1. Esquemas de Gerenciamento

Na Figura 1 apresentamos quatro esquemas de gerenciamento. No esquema 1, a unidade de controle conecta-se diretamente aos recursos controlados. No esquema 2, os agentes estáticos fazem as conexões aos recursos controlados, coletam a informação de gerenciamento e a armazenam na base de informações de gerenciamento [8,9]. No esquema 3, os agentes estáticos são substituídos por agentes móveis e funcionalidade adicional é delegada do gerente aos agentes [10]. No esquema 4, as partes envolvidas podem se mover, isto é, os gerentes, os agentes e os serviços ou recursos. No trabalho atual, exploramos os esquemas com mobilidade, com detalhes de projeto e de implementação para o esquema 3 e considerações de projeto para o esquema 4.

Começamos com o caso mais simples onde somente os agentes são móveis. Nesta situação exploramos um estudo de caso no qual queremos detectar objetos distribuídos que estejam potencialmente causando problemas. A avaliação dos objetos é baseada nas requisições que chegam ou saem dos objetos. Baseado nas requisições estimamos a atividade, isto é, *throughput*, tempo de resposta, volume de dados e mais tarde a comparamos com a ocupação de cpu e de memória. Esta abordagem é baseada na estimativa de potenciais problemas considerando o tipo de aplicação que prevalece no domínio e suas características. Para adaptar o gerenciamento ao ambiente, o gerente deve utilizar uma biblioteca de agentes, selecionando e combinando-os de acordo, ou criando novos agentes e adicionando-os à biblioteca existente.

Uma extensão deste trabalho é o esquema de gerenciamento onde todos os componentes podem ser móveis, se movendo explicitamente como definido em seu código ou se movendo transparentemente por uma decisão externa do middleware ou de um cliente fazendo uma requisição. Maiores detalhes são apresentados mais adiante no texto.

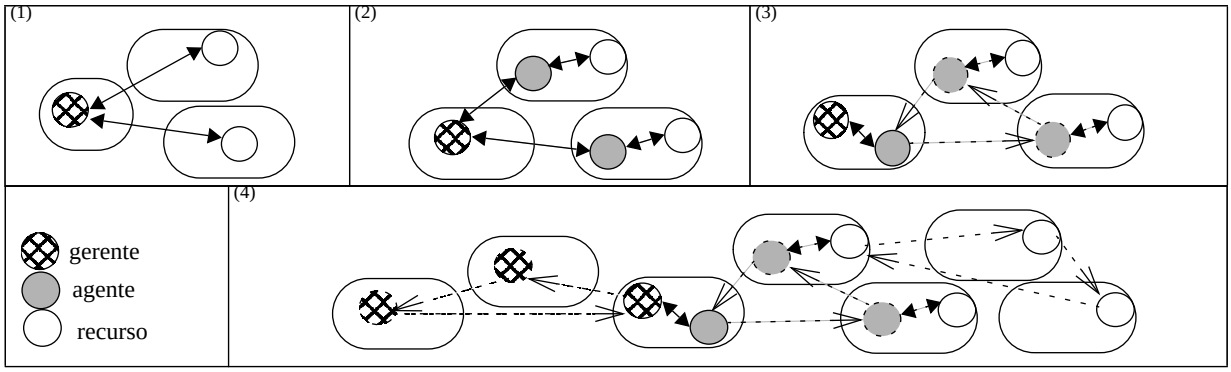


Figura 1. Esquemas de Gerenciamento.

2.2. Paradigma de Agentes Móveis

Na Figura 2, representamos o mapeamento topológico do plano de agentes para o plano correspondente de processamento distribuído e o plano físico subseqüente.

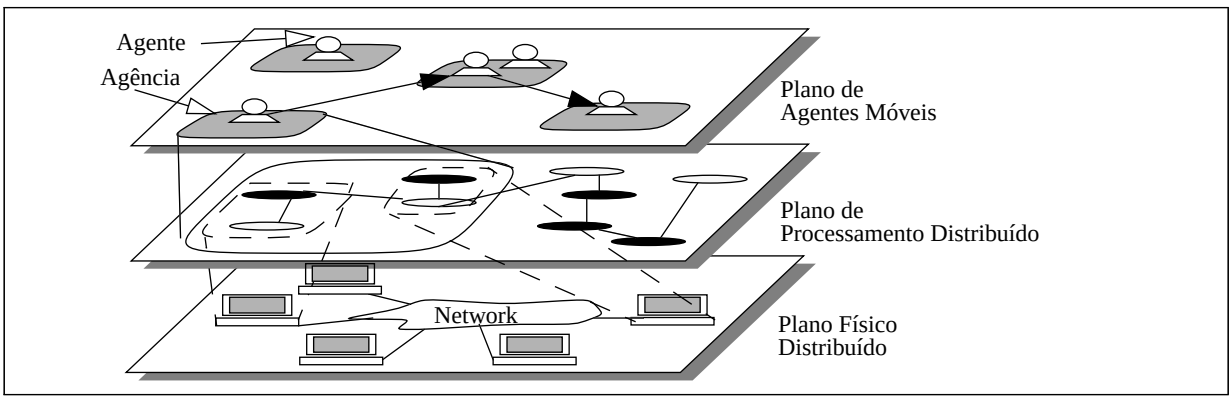


Figura 2. Ambiente de Agentes Distribuídos.

A figura apresenta uma agência agrupando dois nós correspondentes a duas máquinas diferentes no plano físico. A abordagem assíncrona de agentes móveis necessita menor organização de mensagens e sincronização entre processamentos. Espera-se abordar problemas de gerenciamento de forma mais objetiva no plano de agentes móveis do que em nível dos outros dois planos.

Agentes móveis permitem uma arquitetura dinâmica para enviar serviços de gerenciamento,

com aplicações leves e flexíveis, que podem ser criadas, desdobradas, estendidas, ou suprimidas em tempo real [11,12,13,4,w1,w2,w3,w4]. Permite a criação e distribuição de serviços necessários para novos dispositivos orientados a serviços. Serviços podem ser incorporados diretamente nos agentes para executar tarefas de gerenciamento autonomamente e de intervenção sem ajuda humana.

2.3. Ajustes

Nesta seção descrevemos sucintamente os mecanismos de ajuste para o balanceamento dos recursos gerenciados por meio de um *serviço de disponibilidade* [11,12]. Serviços indisponíveis, podem ser (re)distribuídos atendendo uma demanda de balanceamento de carga e/ou *caching* inverso.

Na Figura 3 representamos um agente genérico A enviando uma requisição a um agente móvel B. O agente B pode ser encontrado em uma das agências representadas. A requisição segue o procedimento de localização com um multicast ao nível final a fim de selecionar um hospedeiro que atenda à solicitação. Os mesmos passos são seguidos numa rechamada do agente B ao A. O mesmo procedimento também ocorre quando o agente A deseja migrar transparentemente e envia uma requisição para uma nova instância ativa, isto é, um agente A envia uma requisição a um novo agente A. O primeiro hospedeiro a atender prossegue com a execução do agente.

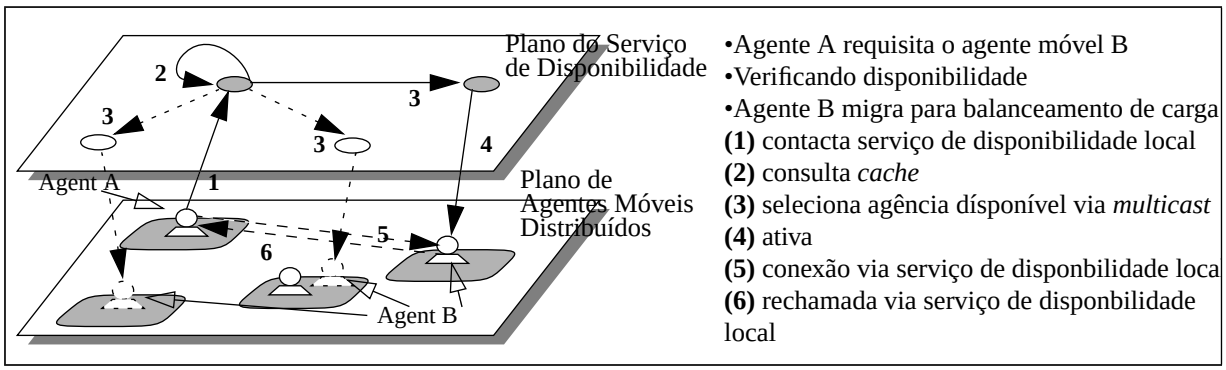


Figura 3. Migração Transparente.

Definimos mobilidade explícita como a demanda por mobilidade que está codificada explicitamente no agente. Também definimos mobilidade implícita ou transparente como sendo conseqüência de uma ação ou razão externa. Por exemplo, um movimento explícito do agente B pode resultar implicitamente em um movimento do agente A se o segundo tiver que executar associado ao primeiro.

Propomos que um movimento implícito ocorra no momento de uma rechamada do primeiro agente em resposta a uma requisição do segundo, como na Figura 3. Com esta abordagem baseada na rechamada esperamos adiar a migração transparente até o agente e os recursos associados responderem. Assim esperamos minimizar migrações desnecessárias ou um efeito *pingue-pongue*.

Procurando e Alocando Recursos. Um agente cliente requisita um outro agente através do serviço de disponibilidade local o qual obtém o domínio deste outro agente, seja de um cache local, de um serviço de Nomes ou Negociador. A partir do domínio obtido o cliente procura conectar-se ao agente servidor, selecionando uma agência aleatoriamente. A conexão à agência passa pelo gerente da agência que consulta o serviço de Nomes local pela IOR correspondente (Referência Interoperável de Objetos). Se somente uma cópia ativa for permitida, o gerente

tenta se conectar retornando a IOR. Se falhar, tenta uma ativação local e retorna a IOR. Se falhar retorna o controle ao lado cliente.

Balanceamento de Recursos . Dois cenários diferentes de serviços e agentes são possíveis. Um cenário é baseado nos serviços que existem somente durante o tempo de vida de uma aplicação, isto é, são serviços transitórios. Este cenário é dominado pelo tempo de inicialização e distribuição dos serviços individualmente. Um outro cenário consiste de serviços e agentes que participam em mais de uma aplicação e que persistem além do tempo de vida individual das aplicações, isto é, serviços persistentes. Persistente não significa estático.

Durante a migração a computação de um serviço está suspensa. Assim, assumimos o tempo de computação (θ) de um serviço como a adição (eq.1) de seu tempo ativo (λ) e seu tempo inativo (δ):

$$\theta \geq \lambda + \delta \qquad \text{(EQ 1)}$$

e tomamos como figura de mérito (ε) expressa na eq.2:

$$\varepsilon \leq \lambda / (\lambda + \delta) \qquad \text{(EQ 2)}$$

As eq.1 e 2 são desigualdades devido à possibilidade de uma falha de uma máquina ou de uma conexão. De acordo com o expresso anteriormente, num cenário do tipo 1 um agente deve evitar vários níveis de requisições e evitar arrastar outros agentes.

Se considerássemos os agentes executando ao longo de uma trajetória predefinida ou aleatória, eq.1 seria expressa como um somatório de pares de tempo ativo e inativo, das etapas computacionais individuais (eq.3).

$$\theta_t = \sum_{i=1}^{n} \theta_i \therefore \theta_t \geq \sum_{i=1}^{n} (\lambda_i + \delta_i) \qquad \text{(EQ 3)}$$

Do acima tomamos uma outra figura de mérito (υ):

$$\upsilon = \sqrt{\sum_{i=1}^{n} \theta_i^2 / \theta_t^2} \therefore \upsilon \leq \frac{1}{n} \qquad \text{(EQ 4)}$$

O ajuste da carga de processamento e de comunicação é baseado no tempo de computação de cada conexão cliente/servidor, que chamamos de um *doublet*. Um *doublet* é a conexão de um agente cliente com um agente servidor através de uma interface. A interface estabelece as fronteiras da célula de migração do agente. As cargas de processamento e de comunicação não são tratadas separadamente mas como um todo.

2.4. Trabalhos Relacionados

Apresentamos sucintamente algumas iniciativas que unem gerenciamento distribuído e código móvel, em particular as que estão sendo realizadas em Java.

A arquitetura do *Java Dynamic Management Kit* (JDMK) [w6] consiste de um conjunto de agentes estáticos que recebem *beans* dinamicamente distribuídos pela rede, sendo (des)encaixados no agente, a fim de adicionar, modificar, ou suprimir serviços, tal como componentes de *hardware* que são (des)encaixados em um bastidor. Há uma biblioteca com um núcleo de serviços de gerenciamento implementados como componentes JavaBeans enquanto serviços novos de gerenciamento podem ser criados. Os agentes JDMK podem colaborar diretamente na resolução de problemas, ao invés de tradicionalmente repassá-los a um gerente num nível mais elevado. A inteligência integrada nos dispositivos pretende: reduzir

o tráfego de gerenciamento na rede, permitir que problemas de baixo nível sejam tratados localmente sem geração de alarmes, permitir uma resposta mais rápida aos eventos e reduzir os custos de administração. Na arquitetura JDMK os beans são a parte móvel do código visto que os agentes são estáticos. Os benefícios são: integração rápida, compatibilidade, reuso de código, economia de tempo e escalabilidade dinâmica.

O *Object Management Group (OMG)* trabalha atualmente na especificação de uma estrutura para suporte à mobilidade de agentes através da especificação do *MASIF - Mobile Agent System Interoperability Facilities Specification* - [4]. O Grasshopper e MAGNA [13, w4] são ambientes de agentes móveis inteligentes segundo o padrão OMG/MASIF. Permitem a criação de aplicações distribuídas com agentes se beneficiando da comunicação e acesso a dados localmente, a velocidade mais alta. Grasshopper implementa uma facilidade de agentes móveis que poderia ser usada como a estrutura para o desenvolvimento de um sistema de gerenciamento baseado no paradigma de agentes móveis.

Este trabalho usa um *middleware* CORBA e agentes móveis CORBA para gerenciar o *middleware* e as aplicações associadas. Em adição, apresenta um *serviço de disponibilidade* para viabilizar ajustes no ambiente.

3. Arquitetura

Nesta seção apresentamos nossa arquitetura MomentA baseada no modelo OMG/CORBA para o middleware e para as aplicações gerenciadas. Esta arquitetura baseia-se nos esquemas de gerenciamento com componentes móveis mencionados na seção 1. Começamos com uma console estática e agentes móveis distribuídos que se conectam aos recursos controlados. Antevemos, uma console móvel e agentes móveis que se conectam a recursos controlados também móveis.

A proposta de gerenciamento é orientada basicamente a monitoração da atividade dos objetos no ambiente gerenciado. Os componentes básicos são: um *intermediador de requisições de objetos* (ORB), um *conjunto de agentes de gerenciamento* da atividade dos objetos, um *serviço de disponibilidade* para o suporte de ajustes no domínio e no ambiente controlados, e um *suporte para mobilidade*.

3.1. Infra-estrutura de Gerenciamento

A infra-estrutura de gerenciamento é composta basicamente de agentes que monitoram a atividade dos objetos. A *monitoração* é *evento-orientada* e baseada em *contabilidade* e *desempenho*. Os *eventos* básicos são as requisições enviadas e recebidas pelos objetos [10,8].

A abordagem é detectar um problema através de agentes com funcionalidades diferentes e o detalhamento sucessivo do problema. Os agentes coletam informação de um conjunto de *sensores* que interagem com *pontas-de-prova* que interceptam as requisições. Os sensores são usados para monitorar hospedeiros no ambiente com um detalhamento até a granularidade de objetos ativos.

Os sensores implementados consideram as métricas apresentadas na Tabela 1. No texto nos referimos a sensores como filtros. O suporte a mobilidade de agentes usa uma infra-estrutura de desenvolvimento própria. Nossa infra-estrutura de mobilidade executa agentes móveis como objetos CORBA que se movem de uma agência para outra e se (des)registram no ambiente CORBA na chegada (partida).

Sensores	Métricas
F0	- número de requisições enviadas (recebidas) - número de respostas - *throughput* total
F1	- tempo total de residência - tempo total de resposta
F2	- taxa de dados enviada por requisição/ resposta - taxa de dados recebida por requisição/ resposta
F3	- tempo de *marshalling* - tempo de *unmarshalling*
F4	- método (operação) mais requisitado
F5	- ocupação de memória e cpu por requisição

Tabela 1. Sensores Modelados.

3.2. Agentes de Gerenciamento

Dois tipos de informação de gerenciamento são considerados: de visão geral com pouca informação coletada em um domínio grande, e de visão mais detalhada em um domínio menor. De acordo com esta abordagem, propomos um conjunto de agentes de dois tipos principais: agentes *em_largura* e agentes *em_profundidade*. Alguns agentes sugeridos são apresentados na Tabela 2.

O agente deve ser configurado pelo gerente para refletir as características do ambiente gerenciado, isto é, o tipo de usuário. Por exemplo, um agente *em_largura* pode coletar o *throughput* total e a ocupação de cpu e memória em um grupo de hospedeiros, usando *agws* do tipo 1 e *ags* do tipo 7. O gerente analisa esta informação e decide sobre a emissão de um agente *em_profundidade* ao hospedeiro(s) em aparente situação crítica, a fim de monitorar em detalhes, a exemplo, por processo: o *throughput* e o número de requisições (*agd* tipo 1), o tempo médio de resposta (*agd* tipo 2), e assim por diante. Toda informação coletada é repassada a um gerente que decide sobre etapas adicionais, isto é, emitir outros agentes (*agw* e *agd*), intervir diretamente na realocação de processos, ou permitir a realocação pela infra-estrutura de ajuste. Para cada agente há um outro agente correspondente de ativação de filtros, *ags* do tipo 6. Há sempre ao menos duas etapas a serem seguidas: um agente para ativar os filtros desejados, e os agentes a coletar e trabalhar a informação filtrada.

tipo	agentes em_largura (*agw*)	filtro	agentes em_profundidade(*agd*)	filtro
1	requisições e *throughput*	F0	requisições, *throughput* e dados	F0+F2
2	tempos de residência e resposta	F1	tempos	F1+F3
3	requisições e tempos de residência/resposta	F0+F1	objetos	F0+F2+F4
4			cpu e memória	F1+F5
5			geral	F0+F1+F2+F3
	agentes especiais (ags)			
6	ativação de filtros	nenhum		
7	ocupação de cpu e memória (%)	nenhum		

Tabela 2. Alguns agentes de gerenciamento utilizando os filtros propostos na Tabela 1.

Um diagrama do sistema de gerenciamento baseado em agentes está representado na Figura 4. Constam as seqüências de gerenciamento: (1,2) enviando um agente *em_largura*; (3) relatando ao gerente (móvel); (4,5) enviando um agente *em_profundidade*; e (6) relatando ao gerente.

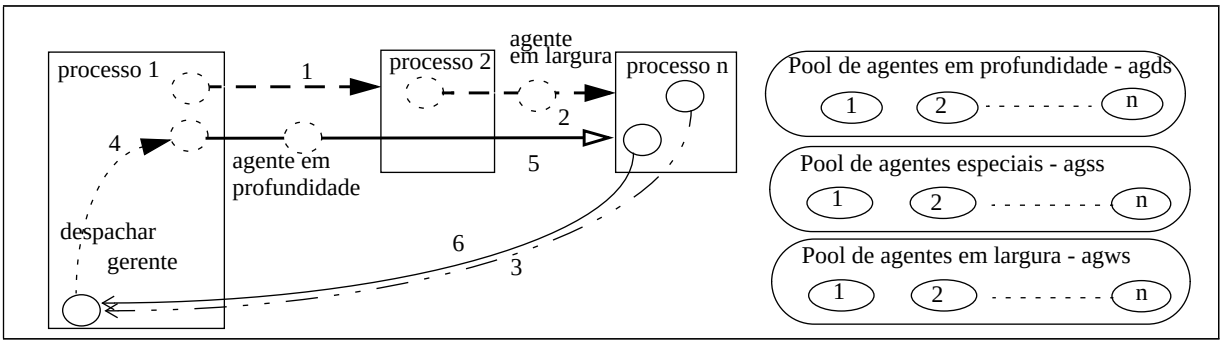


Figura 4. O macro modelo da estrutura de gerenciamento.

3.3. Infra-estrutura de Ajuste

Nesta seção discutimos o uso de um *serviço de disponibilidade* no contexto de gerenciamento de configuração a fim de permitir ajustes no ambiente gerenciado [11,12]. O serviço de disponibilidade intervém fazendo uma contínua migração da computação, de acordo com a demanda e a disponibilidade. Esta intervenção co-existe com os agentes de gerenciamento. Cabe ao gerente permitir este ajuste contínuo ou simplesmente bloqueá-lo. Uma abordagem simples é o bloqueio completo deste ajuste pelo gerente enquanto um refinamento do gerenciamento seria um bloqueio seletivo em determinados hospedeiros ou processo (*cluster*) de objetos.

Até o momento descrevemos como o serviço de disponibilidade proposto participa dos ajustes. Toda requisição com o propósito de ajuste passa pelo serviço de disponibilidade local existente em cada hospedeiro. O serviço de disponibilidade é responsável por encontrar um componente disponível de acordo com o conjunto de restrições passadas na requisição. Se for permitido ao componente se mover no momento da requisição, move-se no sentido dos recursos de prontidão e tem-se uma sintaxe como:

requisição (nomeagente, operação, restrições, QoS).

Esta infra-estrutura é uma facilidade adicionada ao ORB e baseada em outros serviços CORBA. Esta facilidade é adicionada para ajustar o ambiente controlado baseado na migração dos componentes e tem o macro modelo apresentado na Figura 5. Neste modelo, dois serviços foram adicionados às facilidades CORBA para permitir migração transparente: o *serviço de disponibilidade* com uma *interface de transparência* e o *serviço de mobilidade*.

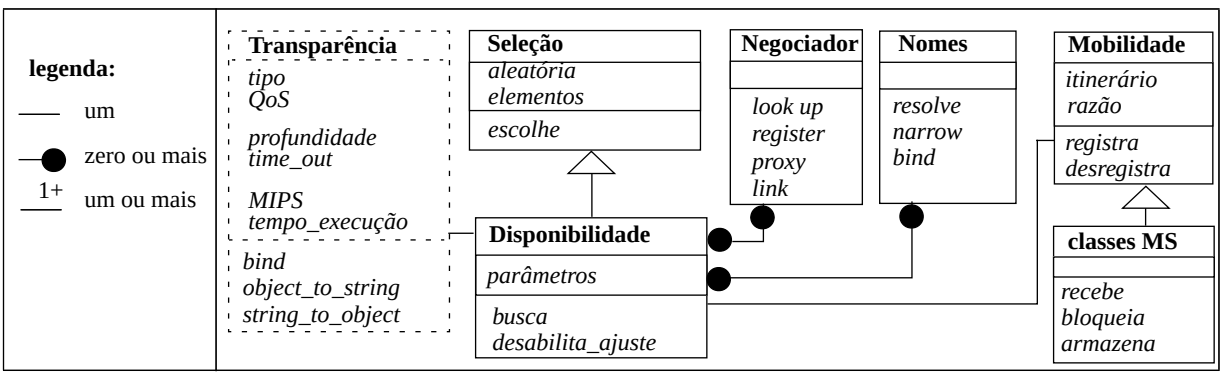


Figura 5. O macro modelo objeto de *disponibilidade*.

Disponibilidade: Esta é a implementação da classe *disponibilidade* no servidor de disponibilidade. Os atributos *parâmetros* são os recursos a serem buscados, como ilustrado na

interface de transparência. O método *busca* procura os recursos e conecta a eles. A estratégia é: 1.busca o serviço de Nomes; 2.busca os recursos do intra domínio; 3.busca os recursos do extra domínio; 4. seleciona; e 5. conecta a instância ativa. O método *desabilita_ajuste* simplesmente comuta a mobilidade para implícita através do atributo *razão,* na classe *mobilidade*.

Transparência: Esta é a interface local disponível a um agente. Os *parâmetros* são gerais. O atributo *tipo* é a razão para a transparência ou tipo de aplicação, por exemplo: desempenho, tolerância a falha e assim por diante. O atributo *QoS* é usado conjuntamente com o *tipo* e define a escala de qualidade desejada na seleção. O atributo *profundidade* limita a profundidade nas conexões remotas e junto com *time_out* limita a pendência das conexões. O atributo MIPS é usado conjuntamente com *tempo_execução* para obter o desempenho desejado de cpu. O método *bind* é usado para resolver o nome de um objeto ou agente remoto. O método *object_to_string* transforma uma referência de objeto em *string* a fim de armazená-la na *cache*, serviço de Nomes ou Negociador. *string_to_object* converte de volta a referência de objeto.

Mobilidade: O atributo itinerário representa a nova localização de um agente, podendo ser uma agência ou domínio de agências. O atributo adicional *razão* indica o tipo da estratégia de migração, isto é, explícito ou implícito. Para a migração implícita, o itinerário é aleatório. O método *(des)registrar* (des)registra a referência do agente em cada agência do domínio. A classe mobilidade aparece como super classe no modelo do suporte à mobilidade apresentado na Figura 6, onde as classes do MS são executadas com os métodos correspondentes. O método *lock_agent* bloqueia o agente a novas requisições. O método *save_agent* armazena o estado. O método *receive_agent* recebe o agente, ativa-o e recupera o estado.

3.4. Serviço de Mobilidade

É uma facilidade CORBA adicionada ao ORB onde um agente móvel tem a estrutura de um objeto CORBA e visto como tal por qualquer outro agente. O que distingue um agente CORBA de um simples objeto CORBA é a existência de um construtor especial no agente que inicializa a sua parte autônoma do código na instanciação. Toda nova ativação após a instanciação simplesmente recupera o estado do agente e prossegue a execução.

Agentes de CORBA se registram em toda agência destino do itinerário como um objeto CORBA, e a fim de mover-se, solicitam desativação na origem e reativação no destino. Distinguimos um objeto móvel CORBA de um agente móvel CORBA estendendo a descrição anterior, isto é, um objeto se move devido a uma demanda externa enquanto um agente pode mover-se devido às demandas externas e internas.

Definimos previamente mobilidade explícita como a demanda interna de mobilidade que está codificada no agente. Externamente um agente associado poderá se mover em conseqüência do movimento explícito do primeiro agente. Esta mobilidade implícita ocorre ao nível de uma rechamada do primeiro agente em resposta a um requisição do segundo, como representado na Figura 3.

O diagrama de classes de mobilidade está representado na Figura 6. Estas classes são usadas integralmente no suporte à mobilidade explícita dos agentes de gerenciamento. Para a mobilidade implícita com a infra-estrutura de ajuste, usamos o *serviço de disponibilidad*e proposto em conjunto com o serviço de mobilidade. A classe *MS_1* é usada para a primeira ativação de um agente enquanto *MS_2* garante todas as ativações subseqüentes em outros hospedeiros. *MSc* interfaceia o controle da *Aplicação Gerente*. *Callbackimpl* implementa *Callback_abs* e permite o gerente ser rechamado pelos agentes. A classe *BaseAgent* é uma classe abstrata e estendida para implementar o agente com *Agimpl*. A classe *Mthread* é responsável pela execução do agente. A classe *MSclient* permite lançar os agentes

alternativamente através de linha de comando. Maiores detalhes deste suporte de mobilidade encontram-se em [14].

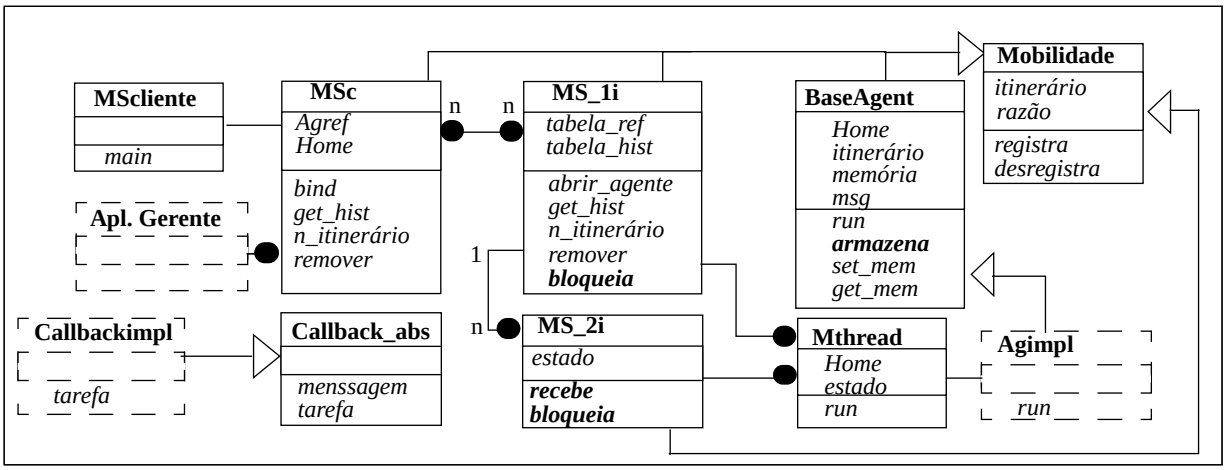


Figura 6. Diagrama de classes do suporte à mobilidade.

4. Aspectos de Implementação

Nesta seção detalhamos alguns aspectos de implementação a respeito do porte de nossa arquitetura orientada a agentes de gerenciamento para um ambiente de computação distribuída baseado em CORBA. A implementação considera Java, JavaORB e as classes org.omg.CORBA disponíveis em *OW3.0* [w5], *javaIDL - JDK1.2* [w8] e *Vbj3.2* [15]. Em adição, temos o serviço de Nomes disponível em javaIDL (org.omg.Naming) e uma versão de Negociador [w7,16]. Trabalhamos basicamente em Unix (SunOS 5.5) e considerando MSWindows mais tarde.

4.1. Infra-estrutura de Gerenciamento

O gerenciamento de monitoração está baseado em contabilidade e desempenho. A infra-estrutura de gerenciamento é composta basicamente de um conjunto de agentes que monitoram a atividade dos objetos através das requisições enviadas e recebidas.

A Figura 7 esboça a estrutura de filtragem de eventos disponível na OrbixWeb2x [w5] e como é introduzida nos processos controlados para posterior leitura por um agente de gerenciamento.

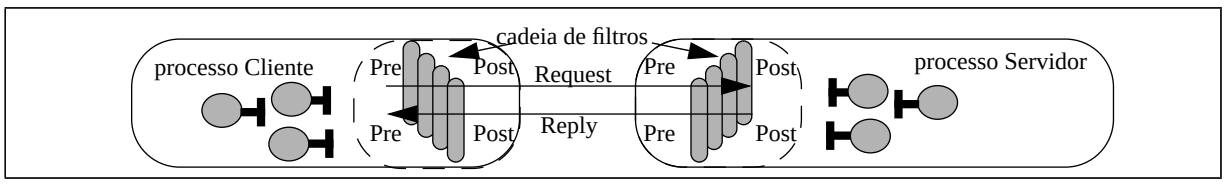
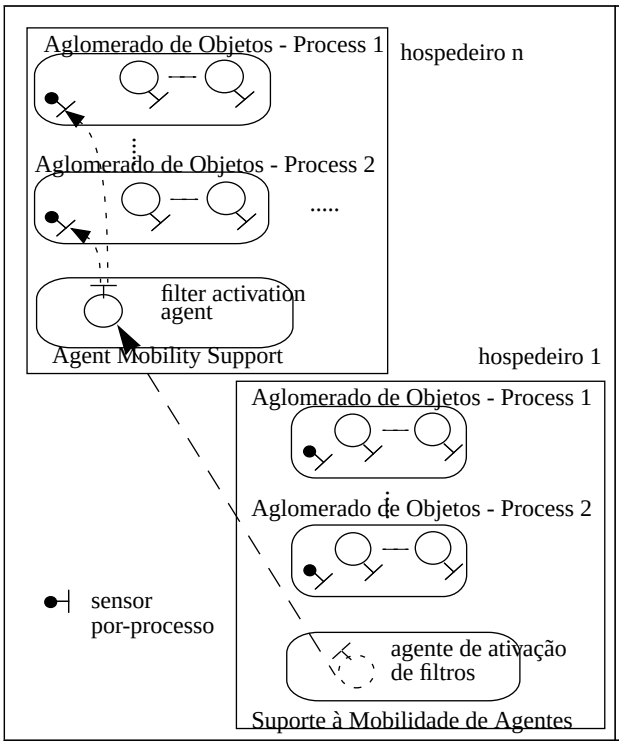


Figura 7. Pontos de monitoração em um processo e o cascateamento de filtros.

Os pontos de inserção desta estrutura são: *saída-requisição-pré(pós)-marshall* e *entrada-resposta-pré(pós)-marshall* no lado cliente, e *entrada-requisição-pré(pós)-marshall* e *saída-resposta-pré(pós)-marshall* no lado do servidor. Em cada ponto é possível cascatear diferentes processamentos do evento. Combinamos estes pontos de inserção com o cascateamento de processamento de eventos para construir nossos filtros.

Do lado esquerdo, na Figura 8, estão respresentados os processos aglutinando objetos, em hospedeiros diferentes. Cada processo ou objeto pode ter ativado tipos diferentes de sensores atra-

vés de um agente móvel. Os sensores escrevem dados em uma estrutura da informação de gerenciamento existente em cada processo. A interface de ativação de filtros em um processo é usada também para leitura da estrutura de informação de gerenciamento. Os agentes móveis são representados movendo-se de um hospedeiro para outro através do suporte a mobilidade de agentes. No lado direito, está a interface IDL com a estrutura de informação de gerenciamento (*dataFilter*) e os métodos que (des)ativam o(s) filtro(s) e lêem a estrutura. A estrutura de informação de gerenciamento suporta todos os filtros da Tabela 1.




```
typedef sequence<boolean> seqbool;
  interface EncapsFilterProcess {
struct dataFilter {
long number_of_request_received;
        long   number_of_request_sent;
        long   number_of_response_received;
        long   number_of_response_sent;
        double residence_time;
        double response_time;
        double rate_of_data_received;
        double rate_of_data_sent;
        double marshalling_time;
        double unmarshalling_time;
        long   memory_occupancy;
        long   cpu_occupancy;
        double throughput;
        char operation_name;
        char marker_name;
        char server_name;
        short  activation_mode;
        char principal;
        double startup_time;
        double final_time; };
    void Activate_Deactivate_Proc(in seqbool filtrosproc);
    dataFilter AcessData( ); };
```

Figura 8. Cenário de ativação de sensores e a interface IDL dos filtros.

Agentes de Gerenciamento: Implementamos todos os filtros da Tabela 1 e um esqueleto de agente que permite implementar os agentes da Tabela 2. Para melhor descrever a implementação de agentes de acordo com a arquitetura descrita na seção anterior, apresentamos um estudo de caso simples e os agentes associados. Para o detalhamento de problemas definimos agentes *em_largura* e *em_profundidade* e um abordagem por-hospedeiro e por-processo. Um processo é um aglomerado de objetos como representado nas Figura 7 e 8.

No cenário de implementação esperamos detectar um conjunto de hospedeiros críticos e entregar um relatório ao gerente. Cabe ao gerente informar os níveis críticos determinados pelo tipo predominante de aplicação e usuário. Baseado nos relatórios o gerente decide-se por ações subseqüentes dos agentes de gerenciamento. O agente *ags* - tipo 6 ativa os filtros desejados nos processos dos hospedeiros. Os agentes *ags* - tipo 7 são similares e uma instância diferente é enviada a cada hospedeiro controlado. Devem permanecer lá por um período verificando a ocupação de cpu e memória para serem lidos mais tarde por um agente *em_largura*, *agw.* Um agente *agw* (tipo 1, 2, 3) circula no ambiente gerenciado, avalia os hospedeiros que estão acima de um ponto crítico e emite um relatório ao gerente. Destes hospedeiros críticos resulta um conjunto que pode ser o itinerário de um agente *em_profundidade* subseqüente, *agd* (tipo 1, 2, 3, 4, 5). Os agentes *agw* e *agd* usam combinações diferentes de filtros.

Na Figuras 9 e 10 ilustramos *ags* - tipo 6 e tipo 7 respectivamente ilustrando o código de uma

implementação de agente. A estrutura de decisão interna de agentes é baseada em níveis, usando valores *fixos* predefinidos ou *médios gerados* a partir de mínimos e máximos coletados numa passada anterior do agente. A decisão pode ser baseada na combinação de sensores diferentes. Cabe ao gerente enviar outros agentes e/ou realocar processos pela via direta ou através da infra-estrutura de ajuste.

```
package AgAct;
import ...;
public class AgAct extends BaseAgent
{
...
public AgAct( ) { super( ); ... ; }
public void run( ) {
/* Filter activation */
_EncapsFilterProcessRef ref1;
_IT_daemonRef daemon;
serverDetailsSeq servers =
new serverDetailsSeq();
seqbool filprocess1 = new seqbool( );
filprocess1.length = 6;
filprocess1.buffer[0] = true;
filprocess1.buffer[1] = false;
filprocess1.buffer[2] = false;
filprocess1.buffer[3] = false;
filprocess1.buffer[4] = false;
filprocess1.buffer[5] = false;
String host = itinerary.buffer[0];
try {
daemon =
IT_daemon._bind("", host);
daemon.listActiveServers(servers);
} catch (... ) {... ; return; }
for(int i=0;i<servers.length;i++) {
try {
srv = servers.buffer[i].server;
ref1 =
EncapsFilterProcess._bind(":" + servers.buffer[i].server, host);
}
} catch (...) { ... ; return; }
}
tinitial = System.currentTimeMillis( );
time = Long.parseLong((String)memory.elementAt(0));
}
```

Figura 9. Agente de ativação de filtros, *ags* tipo 6.

```
package AgCPM;
import ...;
public class AgCM extends BaseAgent
implements _CollectCMOperations {
...
public AgCM( ) { super( ); ... ; }
public void run( ) {
_EncapsFilterProcessRef ref1;
_IT_daemonRef daemon;
String host = itinerary.buffer[0];
tinitial = System.currentTimeMillis( );
time = Long.parseLong((String)memory.elementAt(0));
while ( ) {
try {
/* Read cpu occupancy */
child1 = exec("sar -u 1 10");
in1 = child1.getInputStream( );
...
in1.close( );
// Wait for subprocess to exit
try { child1.waitFor( );
} catch (...) { ... ; }
/* Read Memory occupancy */
child2 = exec("/n/utils/bin/top 0");
in2 = child2.getInputStream( );
...
in2.close( );
try { child2.waitFor( );
} catch ( ... ) { ... ; }

} catch ( ... ) { ... ; }
... ;
}
memory.addElement(host);
memory.addElement(data); } }
```

Figura 10. Agente de medida de ocupação de cpu e memória, *ags* tipo 7.

4.2. Infra-estrutura de Ajuste

Os blocos básicos do serviço de disponibilidade são: um Negociador, um serviço de Nomes, um avaliador de recursos dinâmicos, uma interface de transparência e um gerente de ativação. As etapas seguintes ocorrem durante uma requisição a um agente. Envolve a interação do serviço de disponibilidade local com o Negociador, o serviço de Nomes e os agentes, ou agências, requisitadas através do serviço de disponibilidade remoto.

Localização na Cache: O agente requisitado existe na cache local: 1) Requisita um agente; 2) Encontra uma tabela de localização para o agente na cache local. 3) Faz um multicast para uma instância ativa; 4) Termina com uma conexão bem sucedida; 5) Retorna a execução ao objeto de aplicação.
Exceção [1]: As agências não sabem sobre o agente requisitado: 1) Registra o agente no conjunto de agências definidas na tabela da localização; 2) Reinicia a localização na cache.
Exceção [2]: Nenhuma agência pode atender a requisição em tempo: 1) Começa a localização no serviço de Nomes.
Exceção [3]: Nenhuma tabela de localização para o agente foi encontrada na cache local: 1) começa a localização no serviço de Nomes (ou salta para a localização no Negociador).
Localização no Serviço de Nomes: Há uma referência do agente no serviço de Nomes. O ser-

viço de Nomes é atualizado em cada (des)ativação e reativação.
Exceção [1]: Nenhuma localização para o agente foi encontrada no serviço de Nomes: 1) Começa a localização no Negociador.
Localização no Negociador: O agente requisitado existe no Negociador: 1) Importa a informação do Negociador; 2) Copia a tabela de localização na cache local; 3) Reinicia a localização na cache.
Exceção [1]: O Negociador sabe sobre o agente mas não sabe sobre a tabela de localização: 1) Solicita informação sobre agências com as características sugeridas; 2) Importa a lista de agências que combinam as características anteriores; 3) Seleciona o número sugerido de agências; 4) Registra o agente no novo conjunto de agências; 5) Reinicia a localização no Negociador.
Exceção [2]: O Negociador não sabe sobre o agente requisitado: 1) Retorna a execução ao objeto de aplicação para novas instruções.

5. Resultados

Nesta seção apresentamos alguns resultados no gerenciamento de processos CORBA baseados na seção de implementação. Usamos um programa de distribuição de carga para simular hospedeiros críticos. A carga distribuída é composta de processos CORBA, onde os processos são aglomerados de objetos. Usamos um domínio de teste com três hospedeiros SunOS5.5 e esperamos que os agentes de gerenciamento detectem os problemas simulados. Consideramos todos os processos gerenciados com uma única *thread* de execução, isto é, uma única requisição é atendida de cada vez. Consideramos todas as requisições com resposta, i.e., nenhuma de sentido único.

Experimentação. Enviamos um agente *ags6* para ativar o filtro desejado (F0) e diferentes agentes *ags7* para cada hospedeiro. Estes permanecem nestes hospedeiros e medem a ocupação de cpu e memória. O código para estes agentes está ilustrado na Figura 9 e 10. Após 4 minutos enviamos um agente *agw1* para navegar pelos hospedeiros, coletando dados e relatando ao gerente quais hospedeiros devem ser inspecionados por um outro agente *agd1*. A avaliação do *agw1* é baseada nos padrões da Tabela 3. Durante a primeira passagem os valores são prefixados. Um recurso adicional seria o *agw1* atualizar os níveis para a média dos valores mínimo e máximo coletados na primeira passagem e fazer uma segunda (ou mais) passagem.

ocupação de cpu	**ocupação de memória**	***throughput***	**enviar agente em_profundidade (?)**
acima	acima	acima	sim
acima	acima	abaixo	sim
acima	abaixo	acima	não
acima	abaixo	abaixo	sim
abaixo	acima	acima	não
abaixo	acima	abaixo	sim
abaixo	abaixo	acima	não
abaixo	abaixo	abaixo	não

Tabela 3. Tabela de níveis de decisão para ocupação de cpu, memória e *throughput*.

Na Tabela 4 apresentamos o relatório retornado por *agw1* e por *agd1*. Do relatório gerado o gerente pode decidir enviar um agente adicional ao hospedeiro 3. Mas pode decidir enviar um agente adicional ao hospedeiro 1, por exemplo, um relatório por-processo baseado num *agd1*. O relatório de *agd1* não sendo conclusivo resulta em um *agd2* e/ou *3*, e assim por diante.

Agente: agw1	Mínimo	Máximo	Hospedeiro1	Hospedeiro2	Hospedeiro3
ocupação de cpu (%)	40	50	40	42	50
ocupação de memória(%)	80	94	80	92	94
throughput	0.0132	0.0198	0.0137	0.0198	0.0132
requisições			1000+1000+160	1000+1000+160	1000+1000+160
enviar outro agente			não	não	sim
Agente: agd1			**Hospedeiro1**	**Processo 1**	**Processo 2**
ocupação de cpu (%)	54			54	
ocupação de memória(%)	83			83	
throughput		0.0003833	0.0132		0.0003833
requisições		2000	80		2000
enviar outro agente	sim / não			sim / não	

Observações. Apresentamos a seguir uma comparação de tráfego de mensagems na rede gerado por cada um dos três primeiros esquemas de gerenciamento da Figura 1. Incluímos como mensagem qualquer tipo de acesso remoto. Consideramos a *struct* de cada processo para armazenar a saída dos filtros, como nossa estrutura de informação de gerenciamento. O tráfego (T) de mensagens gerado na rede seria portanto:

Esquema 1	T1 = (estrutura de informação de gerenciamento) * (# hospedeiros) * (# processos)
Esquema 2	T2 = (estado do agente) * (# agentes estáticos) * (# hospedeiros)
Esquema 3	T3 = (código do agente + estado do agente) * (# agentes móveis) * (# hospedeiros +1)

Em princípio, o tráfego de mensagens gerado por agentes estáticos e agentes móveis é comparável se, no primeiro, o tempo de *polling* for comparável ao tempo de coleta do agente móvel, por hospedeiro. A vantagem dos agentes móveis é permitir uma comparação no gerenciamento de hospedeiros sem custo extra de mensagens. Com agentes estáticos, esta comparação só é possível a um custo mais elevado de mensagens. Com agentes móveis temos como limitação uma latência maior.

6. Conclusão

As contribuições da arquitetura MomentA são: gerenciamento de monitoração com agentes móveis para serviços em ambiente CORBA; gerenciamento de configuração com um serviço de disponibilidade; suporte à mobilidade de agentes de gerenciamento; e suporte à migração transparente de componentes.

Das plataformas de agentes citadas, algumas começam a oferecer uma integração com CORBA através da implementação das interfaces CfMAF propostas no MASIF [4]. O JDMK [w6] cuja implementação é voltada para gerenciamento se baseia somente em código móvel. Nenhum dos trabalhos explora a migração transparente de componentes para viabilizar os ajustes no gerenciamento de configuração.

Os agentes móveis suportados pelo serviço de mobilidade são objetos CORBA, isto é, têm referência de objeto. Quando um objeto CORBA se move de um hospedeiro para outro, sua referência muda. O suporte à mobilidade guarda a nova referência, retornando-a quando solicitado. Agentes CORBA se comunicam como objetos CORBA, facilitando as interações.

O gerenciamento é flexível no sentido de agentes em_largura e em_profundidade refletirem um procedimento de gerenciamento a partir do detalhamento do problema. Domínios, hospedeiros, processos e objetos podem ser a trajetória da análise. O gerenciamento de serviços usa os filtros fornecidos pela OrbixWeb [w5] como sensores. Estes filtros demonstram

ser apropriados para coletar dados e executar contadores e temporizadores. O gerente escolhe os agentes a partir de um conjunto e os envia aos hospedeiros desejados. O agente incorpora parte do processamento a fim de minimizar o tráfego de mensagens e não necessita estar previamente lá podendo ser enviado de maneira assíncrona.

Destacamos que, com agentes móveis, se torna possível tratar gerenciamento de maneira assíncrona e simplificar a descrição dos problemas de gerenciamento. Em adição, é possível basear o gerenciamento na comparação entre hospedeiros sem custo adicional de mensagens que seria necessário num esquema com agentes estáticos. Desdobramentos possíveis deste trabalho são aspectos de gerenciamento de configuração e o gerenciamento de hospedeiros móveis baseado no paradigma de agentes móveis.

Agradecimentos. Este trabalho conta com apoio CAPES, CNPq, FAPESP e Pronex/FINEP.

Referências:

1.*The Common Object Request Broker: Architecture and Specification.* Object Managemnt Group, 1998.
2.*CORBA Services*. Object Managemnt Group, 1997.
3.*CORBA Component Model: Initial Submission.* OMG Document orbos/97-11-07, 1997.
4.*Mobile Agent System Interoperability Facilities Specification.* Object Managemnt Group, OMG Document: orbos/97-10-05, Novembro 10, 1997.
5.*ORB Portability IDL/Java Language Mapping*. OMG TC Document orbos/98-01-06, 1998.
6.*Fault Tolerant CORBA.* OMG TC Document orbos/98-03-05, 1998.
7.G. S. Goldszmidt, Distributed Management by Delegation. *Ph.D. Thesis,* Graduate School of Arts and Sciences, Columbia University, USA, 1996.
8.*Standardized Performance Instrumentation and Interface Specification for Monitoring DCE based Application.* OSF DCE RFC 33.0, 1995.
9.A. Queiroz and E. R. M. Madeira, Management of CORBA objects monitoring for the Multiware platform. *Proceedings of the ICODP'97*, Toronto, Canada, Maio, pp. 122-133, 1997.
10.P. Ropelatto et al., Distributed Objects Management in Corba Environment using Mobile Agents. *Proceedings of the International Conference on Networks and Communication Systems - NCS'98*, Iasted, Pittsburgh, USA, Maio, pp. 13-16, 1998.
11.B.Schulze e E.R.M.Madeira, Migration Transparency in Agent Systems, proceedings of the International Symposium on Autonomous Agents and Decentralized Systems, Tóquio, Japão, pg. 320-323, Março de 1999.
12.B.Schulze e E.R.M.Madeira, Contracting and Moving Agents in Distributed Applications Based on a Service-Oriented Architecture *LNCS #1219*, Springer-Verlag, pg. 74-85, 1997.
13.Krause S. et al., MAGNA - A DPE-based Platform for Mobile Agents in Electronic Service Markets. *Proceedings of the 3rd Int. Symposium on Autonomous Decentralized Systems - ISADS'97,* Berlin, Germany, pp. 93- 102, 1997.
14.F.J.S.Vasconcellos e E.R.M.Madeira, Projeto e Desenvolvimento de um Suporte a Agentes Móveis baseado em CORBA, *Anais do 16o SBRC*, Rio de Janeiro, RJ, Maio de 1998, pg. 501-520.
15.Visibroker for Java Progra*mmer's Guide, Release 3.0.* Visigenic Software, Inc., 1997.
16.*ODP Trading Functions.* ISO/IEC JTC1/SC 21, June 20, ftp.dstc.edu.au/pub/arch/RM-ODP, 1995.

Referências da WEB:

w1.Expertsoft Corporation, Tutorial on Distributed Objects, Power Broker CORBAplus, www.omg.org/news/begin.htm.

w2.*Odyssey.* General Magic, www.genmagic.com.
w3.*Aglets Workbench*. IBM Corporation, www.trl.ibm.co.jp/aglets.
w4.*Grasshopper: An Intelligent Mobile Agent Platform.* IKV++ GmbH, www.ikv.de/products/grasshopper/grasshopper.html, 1998.
w5.*OrbixNames Programming Guide*, Iona Technologies, Dublin, Ireland., www-usa.iona.com/support/kb/OrbixNames/articles/604.482.html.
w6.*Java Dynamic Management Kit - JDML.* Sun MicroSystems, www.sun.com/software/java-dynamic/ds-jdmk.html, 1998.
w7.*TOI: OMG Trading Object Service*, IKV++ GmbH, www.ikv.de/products/toi.html.
w8.*JavaTM Development Kit - JDK 1.2.* Sun Microsystems, Inc, java.sun.com/products/jdk/1.2/index.html, 1998.