MAC5701 - Tópicos em Ciência da Computação

Um serviço de coleta de lixo em CORBA

Rosianni de Oliveira Cruz
Orientador: Alan Mitchell Durham
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

25 de junho de 2002

1 Objetivo

Este relatório é exigência da disciplina Tópicos em Ciência da Computação - MAC5701. Nesta disciplina foram estudados assuntos relacionados a coleta de lixo distribuido e CORBA (Commom Object Request Broker Architecture) com o propósito de integrar um serviço de coleta a esta arquitetura. O objetivo deste relatório é expor os tópicos estudados em MAC5701.

2 Introdução

O gerenciamento manual de memória exige que o programador realize um controle rigoroso sobre o código de uma aplicação para evitar desperdício ou desalocação indevida de memória. Geralmente essa recuperação indevida produz resultados inesperados e erros difíceis de serem detectados. Sistemas maiores e mais complexos possuem maior probabilidade a erros de programação relacionados a manipulação incorreta da memória.

Vários algoritmos de coleta de lixo foram propostos para simplificar a construção de sistemas [Wil94]. Estes algoritmos são responsáveis pela desalocação do espaços de memória não mais referenciados por uma aplicação sem a ação explícita do programador. Neste caso temos gerenciamento automático de memória.

Em sistemas onde a computação é distribuída entre diferentes máquinas que comunicam-se através de trocas de mensagens, a construção de coletores de lixo é uma tarefa mais complexa. O coletor deve considerar referências remotas à memória, falhas de máquinas e de comunicação.

CORBA é uma especificação que define um arcabouço para construção de sistemas distribuídos heterogêneos em uma arquitetura cliente/servidor orientada a objetos. O conjunto de abstrações e serviços concretos de CORBA, que facilitam a construção de sistemas distribuídos compostos por diferentes linguagens de programação, sistemas operacionais e plataformas de hardware, seria enriquecido com a adição de um serviço de coleta de lixo distribuído. Desta forma, o programador não se preocuparia com a difícil tarefa de gerenciar a desalocação de memória em sistemas distribuídos.

Um importante trabalho de levantamento e análise de técnicas de coleta de lixo distribuído foi apresentado em 1995 no "International Workshop om Memory Management" [PS95]. O mais recente trabalho com essa abordagem foi publicado em 1998 na "ACM Computing Surveys" [AR98].

Em 1997 foi apresentado um artigo que descreve um algoritmo em coleta de lixo distribuído para sistemas distribuídos construídos com CORBA [RRR97] . Em 1998, uma RFP(Request For Proposal) [rfp98] foi submetida a OMG(Object Management Group) solicitando um serviço padrão de coleta de lixo de instâncias ativas de objetos CORBA.

Este trabalho está oganizado da seguite forma: a Seção 1 discorre sucintamente sobre coleta de lixo distribuido, a Seção 2 faz uma introdução a CORBA, a Seção 3 explana a requisição de um serviço de coleta de lixo presentes em [rfp98], a Seção 4 apresenta o algoritmo proposto em [RRR97] e a última Seção discute nossa proposta de coleta de lixo em CORBA.

3 Coleta de Lixo Distribuído

Um sistema distribuído é composto por processos que executam em espaços de ende- reçamento distintos e comunicam-se apenas por troca de mensagens. Nesse contexto, não está sendo discutido sistemas distribuídos com memória compartilhada.

O principal objetivo do coletor lixo distribuído é detectar espaços de memórias que não são mais referenciados remotamente. Além disso, caso não sejam necessários localmente, esses espaços de memória poderão ser desalocados. Geralmente coletores de lixo local cooperam com o coletor de distribuído.

Nesta seção apresentaremos definições e termos necessários a compreensão de alguns algoritmos de coleta de lixo distribuído. Além disso, mostraremos uma visão geral de coleta da lixo local. No final, discorremos sobre alguns algorimos de coleta de lixo distribuído contido em [PS95] e [AR98] que são mencionados em [RRR97] e [rfp98].

3.1 Definições e Termos

Cada parte do sistema distribuído é chamado de *espaço*.

As estruturas alocadas em memória inspecionadas pelos coletores de lixo, chamaremos de *células* [1] . Uma célula está contida em somente um espaço. Uma célula é *pública* quando é referenciada por outra célula contida em um espaço diferente do seu. Uma célula é *local* quando é referenciada apenas dentro do espaço que a contém. Se duas células estão contidas no mesmo espaço e uma delas referência a outra, temos uma *referência local*, mas se as células estivessem em espaços distintos, teríamos uma *referência remota*. Em relação a uma célula pública, temos o espaço que a contém, *espaço proprietário*, e os espaços que a referenciam remotamente, *espaços clientes*. No espaço cliente existe um *representante* do objeto público. Este objeto público é chamado de *célula alvo* do representante.

Chamamos de *raiz*, as estruturas que certamente são utilizadas por um espaço, como por exemplo, as variáveis globais. Um sistema distribuído pode ser visualizado como um grafo de múltiplas raízes, onde vértices, arestas e raízes são respectivamente, células, referências e raiz de cada espaço desse

[1] O termo célula é utilizado em [AR98]. Em linguagens orientadas a objeto, uma célula seria um objeto

sistema. Uma célula que é alcançada por qualquer raiz do sistema distribuído é considerada *viva*, as outras são *lixo*. Os coletores de lixo detectam e removem lixo. Quando se percorre o grafo a partir de uma célula lixo e consegue-se voltar a essa célula, encontramos um *ciclo de lixo*. As células que compõem esse ciclo são lixo e podem estar no mesmo espaço ou em espaços distintos.

Durante a execução de um sistema distribuído, as operações de criação, duplicação, mi- gração e remoção de referências remotas alteram o grafo do sistema. Quem realiza essas operações em cada espaço do sistema distribuído é conhecido como *modificador*.

3.2 Visão Geral de Coleta de Lixo Local

A coleta de lixo local é realizada de forma independente em cada espaço do sistema distribuído. Independente porque apenas célula contidas no mesmo espaço do coletor de lixo local são investigadas. Para não se gerar inconsistências, uma especial atenção é atribuída as células públicas. Um coletor de lixo local pode agir de duas formas em relação a essas células: nunca coletar uma célula pública ou coletá-la se houver certeza que ela não é mais referenciada remotamente. Na primeira forma, as células públicas podem ser consideradas como parte da raiz do espaço proprietário. A segunda forma necessita da existência de um coletor de lixo distribuído.

As técnicas de coleta de lixo em sistema não distribuídos são utilizados pelo coletor de lixo local. Essas técnicas também serviram de base para a criação de muitos algoritmos de coleta distribuída. No coletor de lixo local, algumas adaptações são necessárias para a correta manipulação de células públicas ou a cooperação com coletores distribuídos. Por exemplo, quando o coletor local detecta que uma célula pública não é mais referenciada por nenhuma célula local, pode sinalizar ao coletor distribuído que o espaço não referencia mais aquela célula. As técnicas mais comuns de coleta de lixo em sistemas não distribuídos são:

- Contagem de referências: um contador é associado a cada célula contendo o número de referências a ela. A cada referência criada ou removida o contador é incrementado ou decrementado em uma unidade, respectivamente. Quando o contador é igual a zero, sua célula correspondente pode ser desalocada. Essa técnica não coleta lixos cíclicos.

- O Marcar-Varrer: algoritmo de duas etapas. Na primeira etapa, as células são percorridas a partir da raiz. Todas as células alcançadas são marcadas. Na segunda etapa, a memória é varrida e todas as células não marcadas são desalocadas.

- Cópia: a memória é dividida em duas partes: a parte ativa e a parte reservada. Na parte ativa, o coletor de lixo percorre as células a partir da raiz e copia todas as células alcançadas para a parte reservada. Não é necessario varrer a memória para descobrir as células lixo, pois toda a parte ativa pode ser desconsiderada. No final do rastreamento, o papel das partes é invertido, ou seja, a parte ativa torna-se reservada e a parte reservada torna-se ativa. Uma variação desta técnica divide a memória em vária partes de acordo com as gerações das células. A idéia do algoritmo é que células jovens possuem maior probabilidade de serem desalocadas. Por isso, a parte da memória com essas células é rastreada mais vezes.

Para um retrospecto detalhado das técnicas de coleta de lixo para sistemas não distribuí- dos, veja [Wil94].

3.3 Algoritmos de Coleta de Lixo Distribuídos

A seguir apresentaremos alguns algorimtos de coleta de lixo distribuído.

3.3.1 Contagem de Referência Distribuída

O algoritmo de contagem de referência distribuída associa a cada célula pública um con- tador. O contador armazena a quantidade de referências remotas a uma célula pública. As operações no contador são de incremento ou decremento de uma unidade. Este algoritmo não recupera lixos cíclicos. Isto porque os contadores de células pertencentes a um ciclo de lixo nunca chegarão a zero.

Descrição do algoritmo:

Quando uma referência remota a uma célula x é criada ou duplicada, o contador deve ser incrementado. Para isso, uma mensagem de incremento é enviada para o espaço proprietário de x. O responsável por enviar esta mensagem pode ser quem enviou a referência ou quem a recebeu. No último caso, referências em transito não são adicionadas ao contador.

Quando um cliente remove uma referência remota para uma célula x, o contador deve ser decrementado. Para isso, esse cliente envia uma mensagem de decremento para o espaço proprietário de x.

Quando o contador de uma célula for igual a zero, significa que não existe referências remotas para ela. Se essa célula também não for acessível localmente, poderá ser desalocada.

3.3.2 Listagem de Referência

O algoritmo de listagem referência armazena para cada célula pública uma lista de identicação de seus clientes. Em virtude dessas listas, o "overhead" de memória desse algoritmo é maior que no algoritmos de contagem de referência. Este algoritmo também não detecta lixos cíclicos.

Descrição do algoritmo:

Quando uma referência remota a uma célula x é criada ou duplicada, uma mensagem de inserção é enviada ao espaço proprietário de x. Se a lista de x não possui uma identificação do cliente que recebeu a referência, essa identificação é criada.

Quando um cliente remove uma referência remota para uma célula x, deve enviar um mensagem de exclusão para o espaço proprietário de x. Nesse caso, o identificador do cliente é removido da lista de x.

Quando a lista de uma célula pública está vazia, significa que não existe referências remotas para ela. Se essa célula também não for acessível localmente, poderá ser desalocada.

3.3.3 Marcar-e-Varrer Sequencial

No algoritmo sequencial de marcar-e-varrer o modificador é suspendido durante a coleta de lixo em todos os espaços. Nesse algoritmo tanto o modificador(M), como as fases de identificação(I) e reclamação(R) do coletor de lixo ocorrem em todos os espaços de forma sincronizada. Nesse algoritmo, um espaço controlador é responsável pela sincronização dos espaços. O controlador pode ser escolhido estaticamente ou dinamicamente. Se escolhido dinamicamente, o espaço que deseja iniciar a coleta de lixo pode ser o escolhido. Se houver mais de um espaço candidato a controlador, alguma politica de escolha deve ser adotada. Este algoritmo degrada muito o desempenho do sistema. Se apenas um espaço deseja a coleta de lixo, todos os espaços devem suspender o modificador até que a coleta de lixo distribuída seja concluída.

Descrição do algoritmo:

Quando um espaço deseja iniciar a coleta de lixo, avisa ao espaço controlador. O contro- lador envia uma mensagem de suspensão do modificador para todos os espaços do sistema.

Quando todos os espaços confirmam que o modificador foi suspendido, o controlador envia uma mensagem de coleta a todos os espaços para que estes realizem a fase de identificação de lixo. Nesta fase todas as células percorridas a partir do raiz são marcadas como vivas. Espaços que referenciam células remotas marcam o registro de saída e este envia uma mensagem de marcação para que célula alvo também seja marcada e continue com o processo de marcação.

Quando o processo de marcação chega a uma célula que já foi marcada, o processo não continua a partir dessa célula. No coletor de lixo distribuído, nosso interesse é que todas as células públicas alcançadas a partir da raiz sejam marcadas.

Quando todas os espaços confirmam que a fase de marcação foi finalizada, o controlador envia uma mensagem de varredura a todos os espaços para que estes realizem a fase de reclamação do lixo. Nesse caso, todas as células públicas que não foram marcadas são recuperadas.

Quando todos os espaços confirmam que a fase de reclamação foi concluída, o controlador envia uma mensagem a todos os espaços para que o modificador continue sua execução.

3.3.4 Marcar-e-Varrer Concorrente

O algoritmo de marcar-e-varrer concorrente permite que o modificador e o coletor de lixo trabalhem concorrentemente em cada espaço. Apenas é necessário a sincronização de todos os espaços nas fases de identificação e reclamação. Não existe dois espaços do sistema distribuído, onde um realiza a fase de marcação e o outro realize a fase de varredura. Apenas algumas operações são mutuamente exclusivas entre o modificador local e o coletor.

Nesse algoritmo serão consideradas três cores: branco, preto e verde. As cores preta e verde identificam células que não são lixo e a cor branca identificam possíveis células lixo. Os reprensentantes são marcados de branco ou de preto. Nesse caso, preto indica que a célula alvo já foi percorrida e branco que a célula alvo ainda deve ser percorrida.

O algoritmo é dividido em quatro partes: fase de iniciação, fase de marcação, detecção de terminação e fase de varredura.

Durante a fase de marcação, nenhuma célula preta pode referenciar remotamente uma célula branca, exceto nos casos em que uma mensagem de marcação está em trânsito para uma célula branca. Para isto o modificador pode cooperar da seguinte forma: quando uma célula for modificada para apontar para um item de saída branco, uma mensagem de marcação deve ser enviado para sua célula alvo, em seguida o item de saida deve ser marcado de preto e por último a célula pode ser modificada para apontar para o item de saida. Essas três operações ocorrem de forma atômica.

Descrição do algoritmo em cada espaço:

Na fase de iniciação, todas as células e os representantes são marcados como branco e a raiz é marcada como verde. Durante esta fase o coletor e o modificador local são mutuamente exclusivos.

Durante a fase de marcação cada representante apontado por uma célula verde x deve enviar uma mensagem de marcação para sua célula alvo e em seguida, o representante deve ser marcado de preto. Depois disso, x é marcado de preto. Essas três operações ocorrem de maneira mutuamente exclusiva entre o coletor e modificador. Nesta fase, quem recebe uma mensagem de marcação deve ser marcado de verde.

Na detecção de terminação é necessário alguma forma de identificar que não há mais células verdes em nenhum espaço. Nesse caso, todos os espaços devem concordar com o término da fase de marcação.

Na fase de varredura todas as células brancas são coletadas.

4 Introdução a CORBA

A OMG (Object Management Group) escreveu e mantém a especificação CORBA. Esta especificação define um arcabouço para a construção de sistemas distribuidos heterogêneos orientados a objeto. Para permitir que objetos sejam implementados em diferentes linguagens de programação foi definido uma linguagem padrão de definição de interfaces, a IDL. A interface definida em IDL é mapeada para várias linguagens de programação, como Java, Smalltalk e C++ para que sejam realmente implementadas. Intâncias da implementação de objetos CORBA em uma linguagem de programação são chamadas de serventes (objetos públicos). Objetos CORBA são ditos entidades virtuais e serventes são ditos encarnações de objetos CORBA.

Em um sistema CORBA, processos clientes realizam requisições a objetos CORBA implemetados em processos servidores. Neste contexto, os termos cliente e servidor são definidos em torno de uma requisição, ou seja, quem faz a requisição é chamado de cliente e quem a recebe é chamado de servidor. O objeto CORBA que para o qual se destina uma requisição é chamado de objeto alvo. Processos clientes e servidores podem executar na mesma máquina ou não.

A comunicação com objetos CORBA é realizado através do ORB(Object Request Broker). O ORB posibilita a transparência de localização do objeto CORBA, pois na visão do cliente a chamada ao objeto alvo é sempre local. Para que diferentes implemetações do ORB se relacionem, a OMG especifica um protocolo padrão de comunicação entre ORB's, o GIOP ("General InterORB Protocol"). Além disso, para que a comunicação entre ORB's seja possível o formato da referência para objetos CORBA deve ser padrão, para isso a OMG definiu um formato de referencia, a IOR (Interoperable Object Reference). Cada IOR identifica um único objeto CORBA. O cliente deve assegurar uma IOR para o objeto CORBA, caso deseje invocar operações desse objeto.

Quando um objeto A invoca uma operação x em um objeto CORBA B deve existir no lado do cliente um representante de B. Este representante é instanciado em tempo de execução quando o cliente recebe a IOR para B. A operação x é invocada localmente ao representante de B, e este delega a operação para B através do ORB. Chegando ao ORB destino, há uma interação com o adaptador de objetos para se encontrar o servente que encarnou o objeto alvo. O adaptador de objetos é reponsável por despachar a requisição para o servente apropriado. Atualmente, o adaptador de objetos mais utilizado é o POA (Portable Object Adapter).

O ciclo de vida de objetos CORBA e serventes são distintos. Objetos CORBA podem ser transientes ou persistentes. Objetos CORBA transientes existe durante o tempo de vida da aplicação que implementa seu servente. Já objetos CORBA persistentes possui tempo de vida indeterminado e guardam seu estado geralmente em uma base de dados. Durante a vida de um objeto CORBA várias instâncias de servente podem encarná-lo, mas em qualquer ponto do tempo não pode existir mais que uma instância encarnando-o. O tempo de vida dos serventes depende das políticas adotadas pelo POA reponsável pelo sevente.

5 Requisitos para um serviço de coleta de lixo em CORBA

Um serviço de coleta de lixo para um ambiente distribuído como CORBA deve realizar um conjunto de tarefas para que a coleta seja efetuada com sucesso. A OMG propôs um documento que especifica quais requisitos mínimos tal serviço deve possuir para a coleta de serventes que não são mais utilizados pelo sistema [rfp98].

Um dos principais objetivos desse documento é padronizar a política de coleta de lixo para a desativação de serventes. Atualmente, cada ORB possui sua própria política de desativação sendo que muitos deles sequer realizam a desativação desses objetos. Além disso, a [rfp98] não deseja políticas baseadas em contagem de referência por serem ineficientes para situações em que ocorre *ciclos de lixo*.

Um serviço de coleta de lixo deve preocupar-se com o eventual particionamento da rede ou a falha de espaços. Além disso, é importante que a política de coleta de lixo funcione corretamente não importando o número de espaços.

Diante de todas a dificuldades existentes para se implementar coleta de lixo em ambientes distribuídos, o documento em questão especifica quais são os requisitos que uma proposta de serviço de coleta de lixo para CORBA deve atender, são eles:

1. O coletor deve possuir mecanismos de desativação de todas as serventes inalcancáveis, inclusive dos ciclos de lixo.

2. Os serventes devem escolher explicitamente o serviço de coleta de lixo como sua política de desativação. Dessa forma, fica a cargo do programador especificar quais serventes são coletáveis dentro do seu sistema.

3. O serviço de coleta de lixo pode precisar de implementação adicional nos serventes que o utilizam, caso a implementação possua *referências fracas*, *escondida* ou *mascaradas* ou caso utilizem outros recursos além da memória que devem ser liberados antes da sua desativação. Uma referência fraca é uma referência invisível para o coletor de lixo, pois se um objeto é referenciado apenas por referências fracas ele é considerado lixo pelo coletor; já uma referência escondida é uma referência guardada em um lugar que o coletor não examina, como por exemplo um arquivo; e uma referência mascarada é uma referência cujo formato foi alterado e apenas a aplicação consegue identificá-la.

4. Caso o serviço de coleta de lixo permita referências fracas, deve descrever: como uma referência forte é feita fraca; caso seja possivel uma referência fraca torna-se forte, como isso será feito; se clientes que possuam referências fracas para um objeto serão avisados quando esse objeto for coletado; se quando um objeto é coletado todas as referências fracas para ele deixarão de existir. Caso contrário, o serviço de coleta de lixo deve descrever alternativas para serem utilizadas por usuários típicos de referências fracas.

5. A proposta de serviço de coleta de lixo deve descrever em que ordem os servente serão desativados.

6. O serviço de coleta de lixo distribuído pode utilizar um coletor de lixo local, mas não deve requerer a existência do mesmo.

Cada espaço do sistema possuirá um objeto coletor de lixo. Para o correto funcionamento do serviço de coleta de lixo algumas interfaces 1 relacionadas ao objeto coletor podem ser necessárias: interface 1: usada e implementada pelos objetos coletores de lixo; interface 2: usada pelo pseudo-objeto ORB e implementada pelo objeto coletor; interface 3: usada objeto coletor e implementada pelo pseudo-objeto ORB; interface 4: usada pelo objeto coletor e implementada pelo gerenciador de memória local.

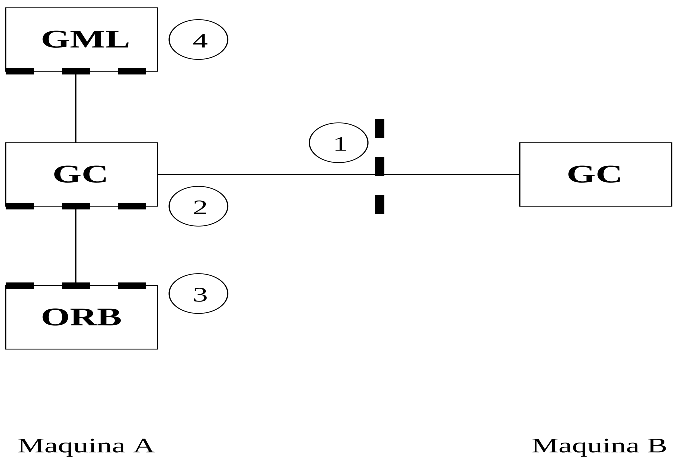

Figura 1: Interfaces para o serviço de coleta de lixo distribuído

Para atender os requisitos desse documento pode ser necessário alterar a especificação CORBA existente, além das especificações do mapeamento da linguagem.

6 Um algoritmo de Coleta de Lixo distribuído em CORBA

O artigo [RRR97] descreve um algoritmo de coleta de lixo cíclico e sua implementação em CORBA. Nesta seção iremos relatar os principais tópicos desse artigo.

6.1 Visão Geral

A técnica de coleta de lixo distribuído utilizada pelo algoritmo é o *rastreamento-de-volta*. Nos algoritmos de marcar-e-varrer, um objeto é definido como vivo se partindo-se das raizes do sistema consegue-se alcançar esse objeto. Já nos algoritmos de rastremanto de volta, para se descobrir se um objeto é vivo, parte-se do objeto e se alguma raiz do sistema for alcançada este objeto é vivo. Outra diferença é que o algoritmo proposto pelo artigo não necessita de sincronização global do sistema.

O coletor de lixo distribuído proposto em [RRR97] utiliza um coletor de lixo local para descobrir as raizes e as *referencias de volta* locais no sistema. Uma referência de volta é um tipo especial de referência utilizada pelo algoritmo para percorrer o objeto e tentar chegar as raízes. Esta referência indica quem aponta um objeto. O coletor de lixo local utilizado nesse artigo é coletor conservativo de Boehm [HJBS91] adaptado.

6.2 O algoritmo

O algoritmo de rastreamento-de-volta consiste em encontrar um objeto suspeito a lixo. Depois recursivamente rastrear a partir do suspeito as referências de volta: um conjunto C inicialmente contém apenas o objeto suspeito a lixo; todos os objetos que apontam para o suspeito são adicionados ao conjunto C; todos os objetos contidos em C que não tiveram suas referências de volta percorrida são verificados e os objetos apontados por essas referências são incluidas em C, e assim sucessivamente, até que não haja mais elementos novos a serem incluidos em C ou uma raiz for encontrada. Se no final do algoritmo não for encontrada nenhuma raiz, todos os elementos contidos em C são lixo.

6.3 Dificuldades de implementação

Para a implementação do algoritmo de rastreamento-de-volta é necessário que alguns problemas sejam solucionados. A seguir serão mostradas os problemas e a solução adotada pelo artigo.

1. Descoberta do suspeito a lixo: Deve-se obter uma boa heurística para a descoberta do suspeito a lixo para a eficiência do algoritmo. Caso contrário, todo o tempo de execução do algorítmo será desperdiçado, pois se o objeto escolhido não for lixo nenhuma memória será liberada.

   Solução: Considerar como suspeito os serventes que não são alcançados localmente

2. Ponteiros unidirecionais: Geralmente o único tipo de referência existente em um sistemas é a *referencia_para*, ou seja, a referência que indica para qual objeto um outro objeto aponta. As referências que indicam quais são os objetos que referenciam um objeto A qualquer não são conhecidas, este tipo de referência chamaremos de *referencia_deVolta*.

   Solução: cada servente e seu representante possuem uma lista de referencias_deVolta. Cada elemento dessa lista possui o seguinte formato: ("identificação do servente ou representante ou raiz local", "servidor no qual está localizado"). Os elementos dessa lista são obtidos descobrindo-se as referencias_deVolta locais e remotas dos serventes e as referencias_deVolta locais de seus representantes. A forma encontrada para descobrir as referências_deVolta locais foi modificar o coletor de lixo local de Boehm. Já as referências_deVolta remotas são facilmente obtidas através da lista de representantes que cada servente deve possuir.

3. Vários rastreamento-de-volta passando pelo mesmo objeto: Se forem executados varios algoritimos de rastreamento-de-volta concorrentemente, um mesmo objeto pode ser rastreado mais de uma vez, portanto deve-se decidir aonde guardar o estado do objeto em cada rastreamento.

   Solução: Cada rastreamento-de-volta será representado por uma fila F. Cada elemento da fila possui o mesmo formato da lista de referências_deVolta. F será mantida no servidor que iniciou o rastreamento de volta.

4. Objetos já rastreados alterados depois pelo modificador pode gerar inconsistência. Observe: Suponha que um sistema S1 qualquer possua somente os objetos *A,B,C,D* sendo que:

   - o objeto *A* faz parte da raiz.

   - o objeto *A* referencia *B* e *C*, e o objeto *C* referencia *D*.

   Suponha que *D* seja suspeito a lixo, portanto seguindo o algoritmo de rastreamento-de-volta os objetos que referenciam *D* serão percorridos, no caso *C*. Em seguida, objetos que referenciam *C* seriam percorridos, mas antes que isto ocorra, o objeto *D* é referenciado por *B* e em seguida *A* deixar de referenciar *C*. Quando o algoritmo percorrer *C*, verificará que ele não é referenciado por ninguém. Neste caso, incorretamente *D* será considerado.

   Solução: Cada servente possuirá um campo chamado versão. Sempre que um objeto for modificado, sua versão será atualizada. Aos elementos da lista de referências_deVolta e na fila F será acrescido mais um campo indicando a versão do elemento. Caso, no final do algoritmo de rastreamento-de-volta, a raiz não seja encontrada todos os elementos de F serão coletados se suas versões forem iguais as versões dos serventes ou representantes associados aos elemetos em questão.

6.4 Implementação

O artigo [RRR97] utilizou um ORB proprietário, o Orbix, para descrever a implementação do algoritmo em CORBA.

O coletor de lixo distribuído foi descrito na implementação como uma biblioteca adicionada ao Orbix. A responsabilidade desta biblioteca é implementar um objeto coletor de lixo em cada servidor. Esses objetos coletores se comunicam para colaborarem na descoberta de serventes que podem ser coletados. Observe que cada servidor pode se ligar a biblioteca ou não. Um servidor ligado a biblioteca pode *importar* ou *exportar serventes coletáveis*, ou seja, pode possuir serventes que se forem lixo poderão ser coletados pelo algoritmo de rastreamento-de-volta ou pode possuir representantes para serventes localizados em servidores ligados a biblioteca. Servidores não ligados a biblioteca podem apenas ter representantes de serventes coletáveis.

No Orbix existe uma tabela de objetos em cada servidor. Esta tabela contém referências para todos os serventes e representantes de serventes contidos no servidor. No entanto, o Orbix não mantém uma lista dos clientes que referenciam remotamente um servente, chamada de *lista de representantes*. Neste caso, para que as referencias_deVota remotas de um servente sejam descobertas as operações “duplicate” e “release” do Orbix devem ser modificadas. Essas operações são chamadas explicitamente pelo programador na duplicação ou remoção de ponteiros para serventes e seus representantes.

Para a descoberta das referencias_deVolta locais e dos suspeitos a lixo, o coletor de lixo conservativo de Boehm foi modificado. O coletor de lixo conservativo de Boehm usa a técnica de marcar-e-varrer para a descoberta de lixo e considera como ponteiro tudo na memória que pareça com um. A alteração proposta é rastrear de forma especial as raízes. Como as estruturas de dados do Orbix fazem parte da raiz e a tabela de objetos do Orbix referencia todos os serventes e representantes de serventes em um servidor, o coletor de Boehm sempre consideraria esses objetos alcançáveis e eles nunca seriam coletados. Portanto, essa tabela não deveria incialmente fazer parte das raizes. No fim da fase de marcação, os candidatos a suspeitos a lixo são os serventes que não foram marcados, pois isto indica que eles não são alcançados localmente. Os serventes e representantes de serventes que foram marcados, podem adicionar às suas respectivas listas de referências_deVolta a indicação da raiz. Já para cada servente e representante de servente x que não foi marcado, percorre-se todos os elementos alcançados a partir de x; os serventes e representantes de serventes encontrados devem adicionar x a suas listas de referencias_deVolta. Além disso, os serventes não marcados devem adicionar os elementos de suas listas de representantes em suas listas de referencias_deVolta. Depois deve-se marcar os objetos rastrados a patir dos serventes e representantes de serventes não marcados para que não haja coleta indevida de objetos referenciados remotamente.

Para distinguir objetos coletáveis de não coletáveis o artigo sugere que a propriedade coletável seja ligada ao nome do objeto, adicionando-se um especial sufixo ao nome do objeto.

O objeto coletor de lixo distribuído possui duas “threads”: “Timer” e “Back-tracing”. A “thread Timer” executa o coletor de lixo local periodicamente para calcular as listas de referências_deVolta. Já a “thread Back-tracing” realiza o algoritimo de rastreamento de volta para diferentes serventes suspeitos a lixo. Em cada servidor há somente um “thread Timer” e uma ou mais “thread Back-tracing”.

7 Proposta

Incialmente nossa proposta de dissertação de mestrado era adicionar um serviço de coleta de lixo a CORBA com base no artigo [RRR97] e na RFP [rfp98]. A política de coleta de lixo escolhida para ser implementada neste serviço utilizaria o algoritmo descrito no artigo, pois ele atende a algumas das exigências da [rfp98]. Entre essas exigências temos a escalabilidade do algoritmo, pois o mesmo não necessita de sincronização global, e a possiblidade de coletar ciclos de lixo.

Como a implementação do serviço solicitado pela [rfp98] pode provocar alterações nas especificações CORBA, optamos por uma implementação do ORB de código aberto. Para este propósito o JacORB foi escolhido. O JacORB é implementado como um conjunto de bibliotecas escritas em Java. Toda a funcionalida de CORBA implementada pelo JacORB é provida pelos *“proxys”* e *“esqueletos”*, os quais por herança utilizam a biblioteca de classes do JacORB. “Proxys” e “esqueletos” são gerados a partir da compilação da especificação IDL de objetos CORBA.

No artigo [RRR97], todos os serventes contidos em um processo ligado a biblioteca de coleta de lixo seriam coletáveis. A [rfp98] sugere que cada servente escolha se deseja ser coletável ou não. Em nossa implementação surgiram duas possiblidades além da indicada pelo artigo:

- Segundo [rfp98], cada servidor do sistema possuirá um objeto CORBA reponsável por parte da coleta de lixo distribuída que chamaremos de objeto coletor. Este objeto guardará a referência de todos os serventes no servidor que são coletáveis, além da lista de referências_deVolta e da versão dos serventes e dos representantes de serventes contidos no servidor. Portanto, um servente para ser coletável deve registrar-se junto ao objeto coletor. Os objetos coletores devem se comunicar para colaborarem no algoritmo de coleta de lixo. Esta comunicação só será possivel se as IOR's dos objetos coletores forem conhecidas. Sendo assim, esses objetos devem registrar-se no serviço de coleta de lixo. O serviço de coleta seria uma aplicação como o serviço de nomes [HV99], cuja localização seria bem conhecida.

- Acrescentar na interface escrita em IDL uma propriedade que indica se um objeto é coletável.

Uma das principais dificuldades da implementação do algoritmo proposto por [RRR97] é a descoberta das raízes de um sistema. No caso do JacORB, é necessario interagir com o coletor de lixo da máquina virtual Java. Entretanto, na linguagem Java não existe nenhuma interface que permita a descoberta das raízes. Portanto, temos como proposta para trabalhos futuros desenvolver uma interface que permita ao serviço de coleta de lixo distribuído requisitar ao sistema quem são suas raízes. Atualmente nossas pesquisas estão voltadas para a definição dessa interface e de usa implementação na máquina virtual Java. Para isso, estamos estudando o artigo [DJ01] que propõe uma interface genérica entre um sistema em execução e o gerenciador de memória. Outra fonte de estudo é o funcionamento de uma máquina virtual Java para que a interface seja implementada.

Referências

[AR98] Saleh E. Abdullahi and Graem A. Ringwood. Garbage collecting the Internet: a survey of distributed garbage collection. ACM Computing Surveys, 30(3):330–373, setembro 1998.

[DJ01] Alan M. Durham and Ralph E. Johnson. Modular garbage collectors: a generic interface for memory management. Abril 2001.

[HJBS91] Alan J. Demers Hans-J. Boehm and Scott Shenker. Mostly parallel garbage collection. In Proceedings of Conference on Programming Languages Design and Implementatio, ACM SIGPLAN Notices, Toronto, Ontario, junho 1991. ACM Press.

[HV99] Michi Henning and Steve Vinoski. Advanced CORBA Programming with C++. Addison-Wesley, 1999.

[PS95] David Plainfossé and Marc Shapiro. A survey of distributed garbage collection techniques. In IWMM'95 Proceedings of International Workshop on Memory Management, volume 986 of Lecture Notes in Computer Science, Kinross, Scotland, setembro 1995. Springer-Verlag.

[rfp98] Corba garbage collector service request for proposal. Technical report, Framingham Corporate Center, Agosto 1998.

[RRR97] Gustavo Rodriguez-Rivera and Vince Russo. Cyclic distributed garbage collection whithout global syncronization in corba. In OOPSLA97 Workshop on Garbage Collection and Memory Management, outubro 1997.

[Wil94] Paul R. Wilson. Uniprocessor garbage collection techniques. Technical report, University of Texas, January 1994. Expanded version of the IWMM92 paper.