3

O Pronome SENDER

Neste capítulo, é explorado o pronome SENDER. Na Seção 3.1, a relação entre o objeto corrente e o objeto emissor da mensagem que está sendo tratada é explicada. Na Seção 3.2, são mostradas as transformações de código necessárias para que o pronome SENDER seja corretamente executado. Na Seção 3.3, é apresentado o nível Meta necessário para a obtenção destas transformações. Finalmente, na Seção 3.4, a implementação do padrão *Distributed Callback* [MM97], descrito na Subseção 2.4.1, utilizando o pronome SENDER é comparada com a implementação tradicional.

3.1

O Objeto Emissor da Mensagem Corrente

O fluxo de execução de um programa orientado a objetos pode ser dividido de acordo com os vários objetos em cujos contextos algum código foi executado. Da mesma forma que numa corrida em equipe, onde um competidor passa o bastão para seu companheiro, um objeto envia uma mensagem a outro passando para este a responsabilidade sobre a execução do sistema. Estes envios e recebimentos de mensagens, alinhavam a execução do sistema, definindo uma relação de emissor-receptor entre os objetos que o compõem.

Nesta relação, a visibilidade é dada do emissor para o receptor já que é aquele que endereça a mensagem. O pronome SENDER pode ser usado para representar o objeto emissor no contexto do receptor e, com isto, permitir que a relação seja percorrida também no sentido inverso. Retorno de resultados, que antes precisavam ser feitos de forma síncrona, ou através de referências cruzadas, podem agora ser feitos através deste pronome.

3.2

Transformações de Código

A Figura 3.1 mostra o cenário no nível base de duas classes que utilizam o pronome SENDER. A classe *A*, em algum de seus métodos, envia a mensagem *m* através do pronome. Da mesma forma, a classe *B* envia a mensagem *n*.

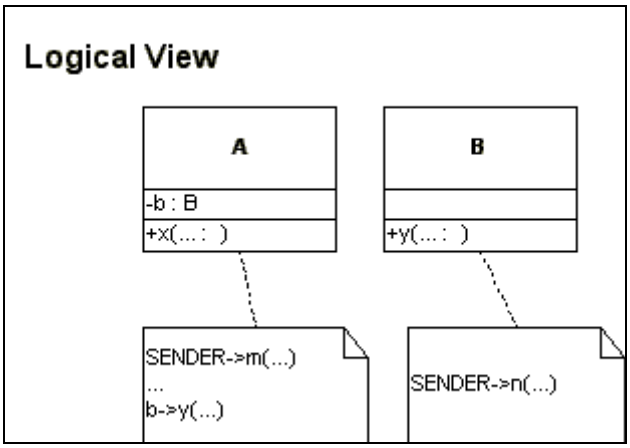

Figura 3.1 - Cenário no nível base de utilização do pronome SENDER

Este cenário deve ser transformado de modo que:

1. o objeto emissor seja conhecido no momento em que uma mensagem a este seja enviada para que o comando possa ser transformado no envio direto ao objeto emissor; e
2. as mensagens enviadas através do pronome (*m* e *n*), caso não sejam tratadas, sejam descartadas sem que nenhum erro de execução seja gerado.

Este último comportamento é desejável devido ao caráter anônimo dos pronomes e, para obtê-lo, todas as classes de nível base, depois de transformadas, passam a herdar de uma classe comum onde todas as mensagens enviadas via pronomes são implementadas de forma vazia.

A determinação de quais mensagens devem ser implementadas de forma vazia na superclasse comum exige que o código de todas as classes do sistema seja analisado a procura de mensagens enviadas através de pronomes. A fim de evitar este passo dispendioso na transformação, pode-se introduzir uma seção declarativa na classe, onde são declaradas todas as mensagens enviadas por seus objetos via pronomes. A simples inspeção desta seção garante a geração

correta da superclasse. Esta seção declarativa pode ser obtida com a introdução de um novo especificador na linguagem, por exemplo *message*. Assim, as classes *A* e *B* da Figura 3.1, deveriam declarar que enviam, respectimavemente, *m* e *n* com este especificador:

- Classe *A*: <<message>> m(...)
- Classe *B*: <<message>> n(...)

As mensagens declaradas como enviadas através de pronomes são, portanto, captadas e adicionadas a uma classe *CPronome*, provida como suporte de tempo de execução, com implementação vazia. Além disso, é introduzida nas classes *A* e *B* uma herança da classe *CPronome*. Esta inserção é feita, na realidade, nas classes raízes das árvores hierárquicas, sendo feita nas duas classes do exemplo por estas não possuírem superclasses especificadas.

Para a primeira transformação, é necessário mapear quando a relação é definida e sua duração, isto é, de que ponto a que ponto do código o objeto emissor pode ser representado por um mesmo objeto. A relação emissor-receptor é definida quando uma mensagem é enviada e dura enquanto esta está sendo tratada. Para que o objeto emissor seja conhecido no decorrer deste código, ele é adicionado aos parâmetros da mensagem. Como não se sabe previamente se alguma mensagem via SENDER será enviada no decorrer do tratamento de uma mensagem, o objeto emissor é, na realidade, adicionado aos parâmetros de todas as mensagens. Assim, um envio através do pronome SENDER pode ser transformado em um envio para o objeto recebido através deste parâmetro.

A Figura 3.2 mostra o cenário transformado. Neste, a classe *CPronome* aparece como raiz da árvore hierárquica e implementa de forma vazia os tratadores para as mensagens enviadas através do pronome SENDER. Aos tratadores de mensagens (*x* da classe *A* e *y* da classe *B*) foi introduzido um novo parâmetro (*pSender*) para receber o objeto emissor da mensagem. Os envios de mensagens através do pronome SENDER (*m* enviada no contexto da classe *A* e *n* enviada no contexto *B*) foram alterados para que sejam endereçadas ao objeto recebido no novo parâmetro (*pSender*). Além disso, em todos os envios, o objeto corrente da execução é introduzido como argumento extra, a fim de que seja conhecido no contexto do objeto receptor.

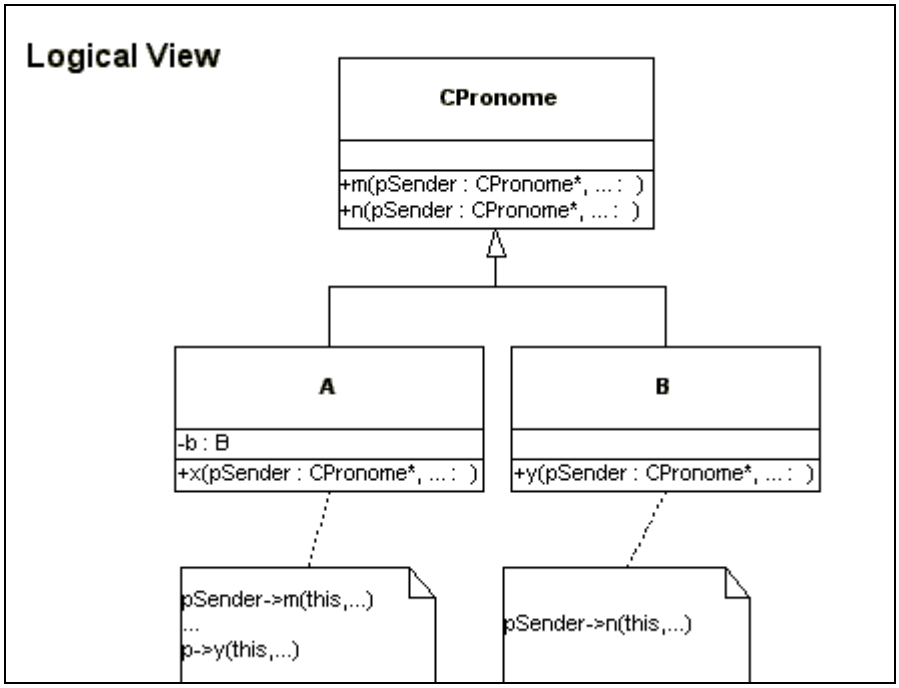

Figura 3.2 – Cenário de utilização do pronome SENDER transformado

3.3

O Nível Meta

Para se obter estas transformações, a todas as classes do nível base é associada a metaclasse *MCSender*, a menos da classe *CPronome* que é associada a uma metaclasse especial denominada *MCRaizPronome*. O nível meta e sua associação com o nível base mostrado anteriormente são detalhados na Figura 3.3.

Os nomes dos métodos que aparecem na Figura 3.3 são os definidos em OpenC++, podendo no entanto ser mapeados para qualquer outra linguagem aberta desde que as características de reflexão sejam preservadas :

- *void Class::Initialize()*: inicializador da classe, é executado apenas uma vez para cada metaclasse exatamente antes do compilador iniciar;
- *void Class::RegisterNewAcessSpecifier(char* especificador)*: registra o especificador;
- *void Class::RegisterNewClosureStatment(char* palavra-chave)*: registra *palavra-chave* como um identificador de um novo comando que não possui desvio de fluxo de execução;

- *void Class::TranslateClass(Environment* env)*: transforma a declaração da classe;
- *Ptree* Class::BaseClasses()*: retorna as superclasses da classe;

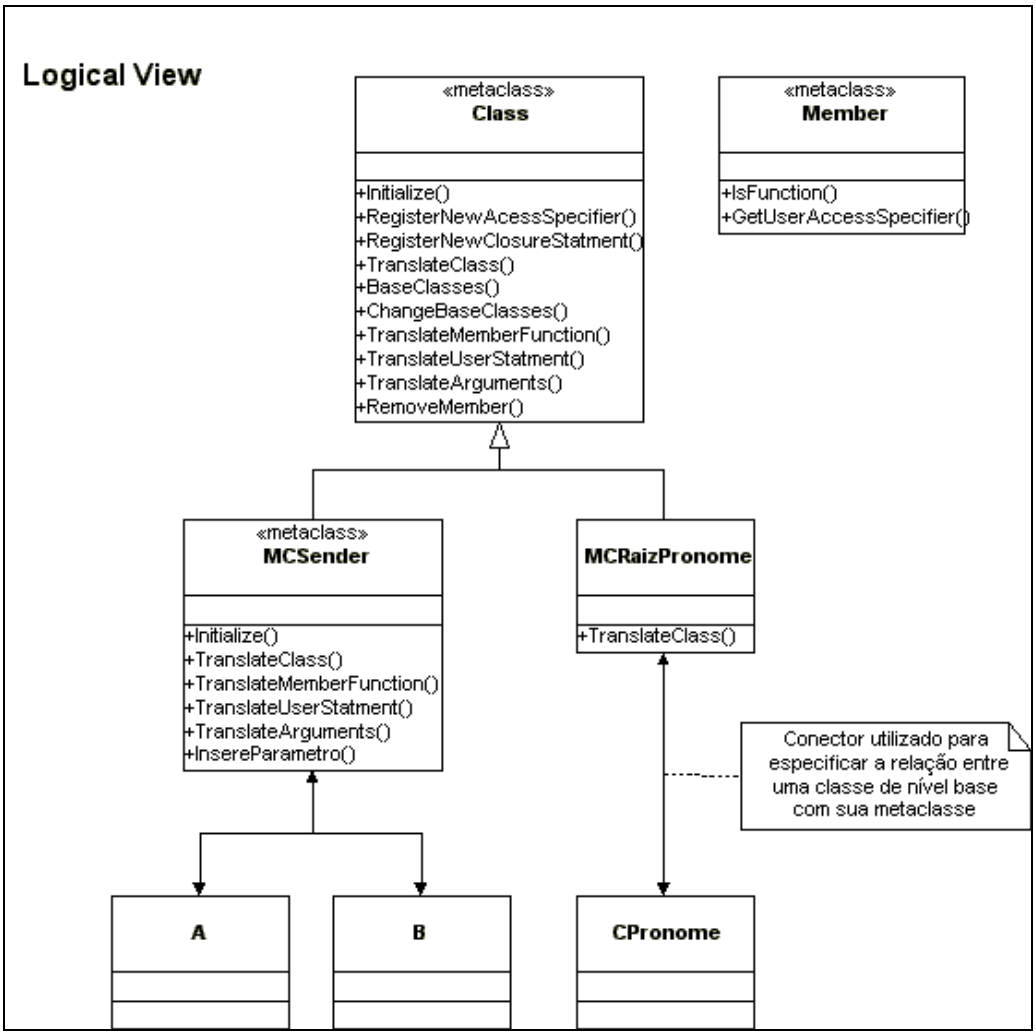

Figura 3.3 – Nível Meta para implementação do pronome SENDER

- *void Class::ChangeBaseClasses(Ptree* base-classes)*: altera as superclasses da classe para *base-classes*;
- *void Class::TranslateMemberFunction(Environment* env, Member& m)*: transforma a implementação de uma função membro especificada por *m*;
- *Ptree* Class::TranslateUserStatment(Environment* env, Ptree* object, Ptree* op, Ptree* palavra-chave, Ptree* rest)*: transforma um comando registrado na inicialização;
- *Ptree* Class::TranslateArguments(Envorinment* env, Ptree* args)*: transforma uma lista de argumentos;

- *void Class::RemoveMember(Member& m)*: remove o membro especificado por *m*;
- *bool Member::IsFunction()*: retorna *true* se o membro for uma função;
- *Ptree* Member::GetUserAccessSpecifier()*: retorna um especificador registrado, se este for aplicado ao membro.

Para que o novo especificador (*message*) e o novo comando (envio de mensagem para o SENDER) seja identificado, a metaclasse *MCSender* redefine o método *Initialize()* e faz os dois registros:

- RegisterNewAcessSpecifier("message")
- RegisterNewClosureStatment("SENDER")

A inserção da nova herança, caso a classe seja raiz de uma árvore hierárquica, e o armazenamento das mensagens enviadas através do pronome são feitos na redefinição do método *TranslateClass()* pela classe *MCSender*.

Para o armazenamento das mensagens enviadas através do pronome, é utilizado um objeto *Singleton* [G+95], denominado *RepositorioMensagens* (apresentado na Figura 3.4), introduzido como suporte de tempo de execução. Este objeto armazena as mensagens declaradas com o especificador <<message>> e as disponibilizam para que sejam introduzidas na classe *CPronome*.

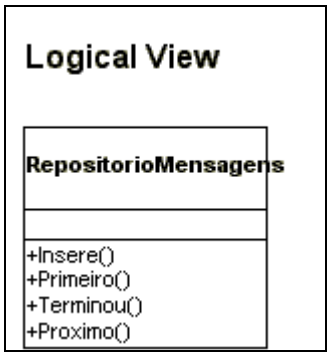

Figura 3.4 – Repositório de mensagens disponibilizado como suporte de tempo de execução

A inserção do parâmetro extra nos protótipos dos métodos é feita na redefinição do método *TranslateMemberFunction()*. Já a inserção do objeto corrente como argumento extra em um envio de mensagem é feita redefinindo-se o método *TranslateArguments()*.

Por fim, a transformação do envio efetivo da mensagem através do pronome SENDER é feita redefinindo-se o método *TranslateUserStatment()*. Este substitui no comando original a palavra-chave SENDER pelo parâmetro *pSender* que, pela transformação efetivada pelo método *TranslateMemberFunction()*, está disponível em todos os métodos.

Ao término da transformação da última classe associada a *MCSender*, todas as mensagens enviadas através do pronome SENDER estão armazenadas e já com o parâmetro extra inserido.

A metaclasse *MCRaizPronome* redefine o método *TranslateClass()* e, neste, insere na classe *CPronome* uma função membro para cada mensagem armazenada no repositório, sempre com implementações vazias.

3.4

Distributed Callback utilizando SENDER

O padrão *Distributed Callback* [MM97], descrito na Subseção 2.4.1, pode ser implementado utilizando-se o pronome SENDER. Esta implementação é esquematizada na Figura 3.5.

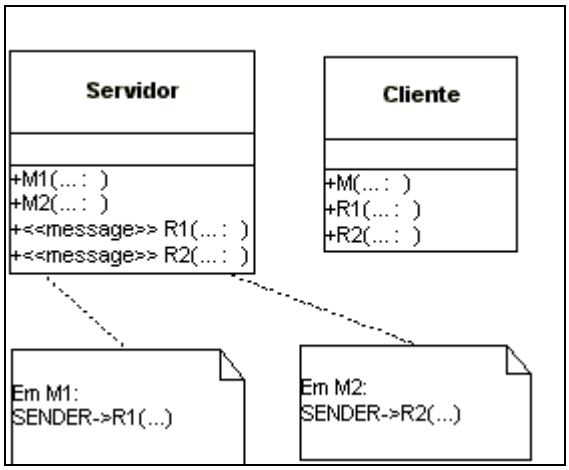

Figura 3.5 – Esquema da implementação do padrão *Distributed Callback* utilizando SENDER

Nesta solução não há risco de propagação da referência, já que o cliente não precisa exportar para o servidor uma referência de si para que o retorno lhe seja enviado. Além disso, as classes tornam-se totalmente desacopladas,

aumentando a generalidade do servidor e permitindo, assim, que um mesmo servidor atenda a diferentes tipos de clientes.

A desvantagem da utilização do pronome, que seria o aumento no tempo de execução do sistema devido à necessidade de manutenção da relação emissor-receptor, pode ser melhor avaliada observando-se a Tabela 3.1 e o gráfico mostrado na Figura 3.6. Nestes, os tempos de execução da implementação do padrão *Distributed Callback* utilizando o pronome SENDER e da implementação sugerida no próprio padrão são comparados. Estes tempos foram conseguidos executando-se o padrão nas duas implementações e variando-se o número de mensagens enviadas sem participação direta na implementação do padrão. A variação escolhida foi devido ao fato de ser exatamente neste tipo de construção que a implementação do pronome SENDER adiciona código a ser executado "desnecessariamente", aumentando, com isso, o tempo de execução do sistema. Além disso, para se obter uma melhor confiabilidade nos resultados obtidos, cada configuração do exemplo foi executada 50 vezes, sendo os tempos mostrados na Tabela 3.1 as médias destas execuções. Estes experimentos computacionais foram realizados em um computador Pentium4, com 256 MB de RAM, utilizando-se Linux Mandrake 8.1.

Número de mensagens enviadas	Implementação Padrão (μs)	Implementação com SENDER(μs)	Diferença (μs)	Aumento Percentual
0	1080	1149	69	6,3888
1	1116	1191	75	6,7204
10	1105	1228	123	11,131
100	1198	1260	62	5,1752
1000	1138	1251	113	9,9297
10000	1363	1488	125	9,1709
100000	3794	4167	373	9,8313
1000000	28812	31473	2661	9,2357
10000000	287120	313993	26873	9,3595
100000000	2870200	3154227	284027	9,8957
1000000000	28692222	31684740	2992518	10,4297
10000000000	286822565	317889885	31067320	10,8315

Tabela 3.1 – Tempos de execução do padrão *Distributed Callback*

Número de mensagens enviadas	Desvios Padrões Implementação Padrão	Desvios Padrões Implementação com SENDER
0	0,2678	0,2367
1	0,2891	0,3189
10	0,2801	0,4321
100	0,2168	0,1254
1000	0,2791	0,2609
10000	0,2846	0,2738
100000	0,2709	0,2386
1000000	0,4291	0,2951
10000000	0,3289	0,2418
100000000	0,3319	0,3178
1000000000	0,3167	0,3065
10000000000	0,2795	0,3290

Tabela 3.2 – Desvios Padrões das amostras de tempo utilizadas para análise da execução do padrão *Distributed Callback*

Os dados apresentados na Tabela 3.1 podem ser melhor visualizados na Figura 3.6.

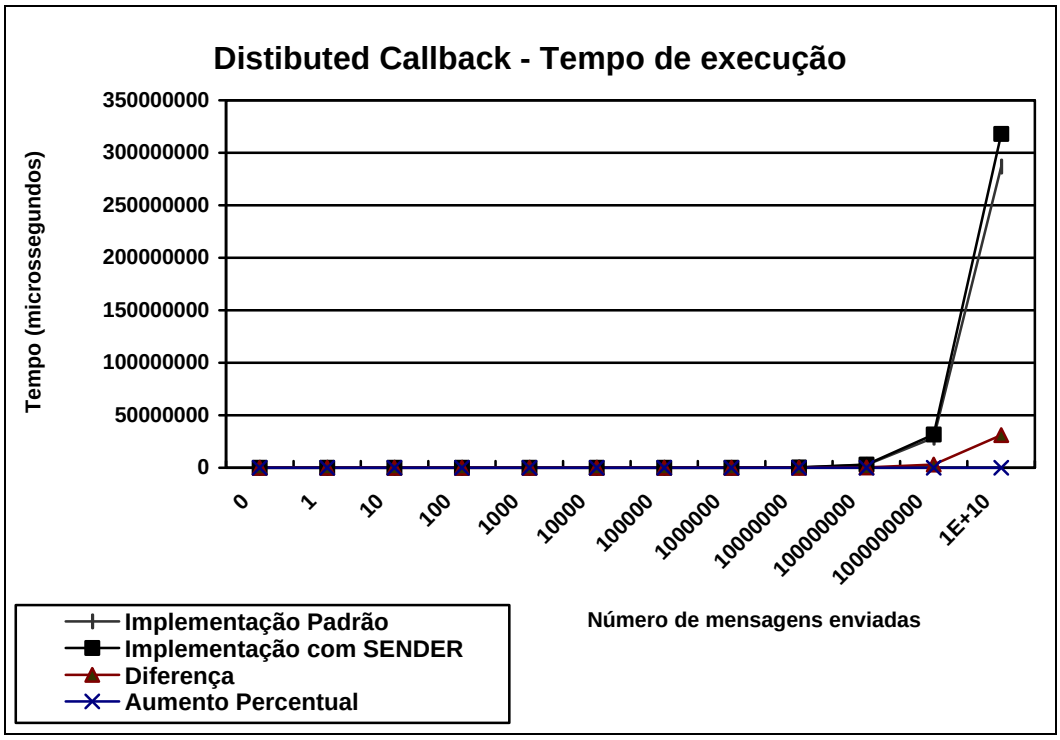

Figura 3.6 – Gráfico de comparação entre os tempos de execução do padrão *Distributed Callback*

Na Tabela 3.1 e no gráfico mostrado na Figura 3.6 pode-se verificar que o tempo de execução na implementação do padrão utilizando-se o pronome SENDER é sempre um pouco maior, o que já era esperado. O aumento percentual máximo obtido, no entanto, foi de 10%, evidenciando a não degradação do sistema com o uso do pronome.