4

O Pronome CREATOR

Neste capítulo, é explorado o pronome CREATOR. Na Seção 4.1, a relação entre o objeto corrente e seu criador é explicada. Na Seção 4.2, são mostradas as transformações de código necessárias para que o pronome CREATOR seja corretamente executado. Na Seção 4.3, é apresentado o nível Meta necessário para a obtenção destas transformações. Finalmente, na Seção 4.4, a implementação do algoritmo paralelo de divisão e conquista, descrito na Subseção 2.4.2, utilizando o pronome CREATOR, é comparada com a implementação tradicional.

4.1

O objeto criador do objeto corrente

No início da execução de um sistema orientado a objetos, apenas seu objeto principal, no caso de linguagens OO que baseiem sua execução na existência deste, ou os objetos globais, no caso das outras linguagens, estão criados. Durante sua execução, no entanto, objetos são criados e destruídos em conseqüência de ações de outros objetos, definindo uma relação de criador-criatura entre os objetos que compõem o sistema. Podendo ser definida como “o criador é um objeto que, enquanto atendendo a uma solicitação de serviço, cria outro objeto, sua criatura”, esta relação é estabelecida quando um objeto é criado.

A visibilidade da relação é dada do criador para a criatura, já que aquele para criar esta necessita endereçá-la. O pronome CREATOR pode ser usado para representar o objeto criador no contexto da criatura e, com isto, permitir que a relação seja percorrida também no sentido inverso.

A Figura 4.1 mostra o cenário no nível base de duas classes que utilizam o pronome CREATOR. A classe *A*, em algum de seus métodos, envia a mensagem *m* através do pronome e cria um objeto modelado pela classe *B*. A

PUC-Rio - Certificação Digital Nº 9816130/CA

classe *B*, por sua vez, possui um atributo modelado pela classe *A* e envia a mensagem *n* a seu criador. As duas classes declaram as mensagens enviadas via pronome através do especificador <<message>> introduzido na linguagem.

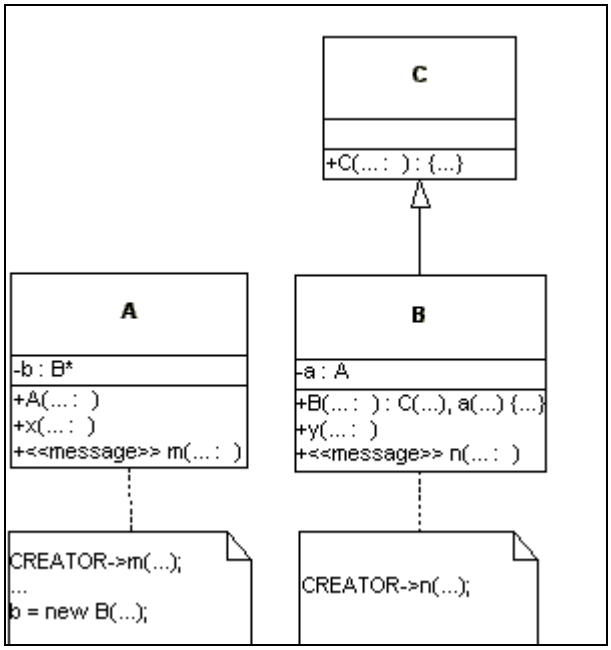

Figura 4.1 – Cenário de utilização do pronome CREATOR em nível base

4.2

Transformações de Código

O cenário anterior deve ser transformado de modo que:

1. as classes *A, B* e *C* possuam uma superclasse que implemente de forma vazia um tratador para *m()* e para *n()*; e
2. o objeto criador do objeto corrente seja conhecido no momento em que uma mensagem a este seja enviada.

A Figura 4.2 mostra o cenário transformado.

Para a primeira transformação, pode ser adotada a mesma estratégia usada para o pronome SENDER.

Para a segunda transformação, como no caso da relação emissor-receptor descrita no capítulo anterior, é necessário mapear quando a relação é definida e sua duração. A relação criador-criatura é definida quando um objeto é criado e dura por todo o ciclo de vida deste. Para que o objeto criador seja conhecido durante o ciclo de vida de sua criatura, precisa ser guardado como um de seus

PUC-Rio - Certificação Digital Nº 9816130/CA

atributos, sendo este preenchido em tempo de construção do objeto. Isso pode ser feito nos construtores das classes, sendo o objeto criador passado a estes como um parâmetro extra.

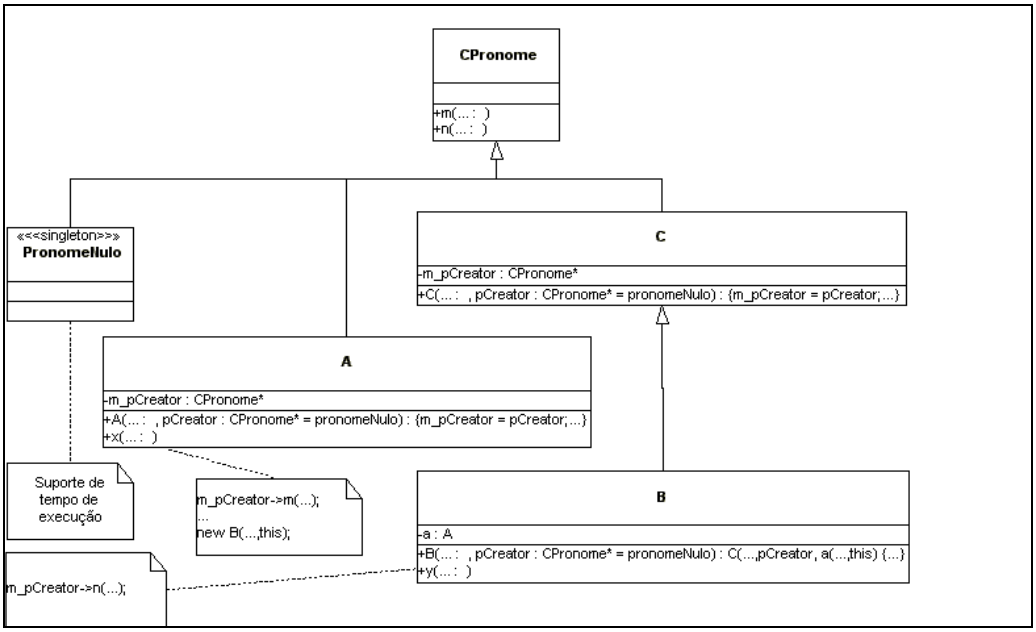

Figura 4.2 – Cenário de utilização do pronome CREATOR transformado

Em toda classe raiz de uma árvore hierárquica é introduzido, portanto, um novo atributo (*m_pCreator*), que armazenará o objeto criado. A restrição da classe sem superclasse especificada vem do fato de, se introduzido na raiz, o atributo será de todas as classes que compõem sua árvore hierárquica. Se, por outro lado, o atributo for inserido em todas as classes, uma classe terá desnecessariamente *n* atributos *m_pCreator*, sendo *n-1* o número de superclasses existentes em sua árvore hierárquica.

Objetos podem ser criados explicitamente, quando declarados como referência, ou implicitamente, no caso contrário. O primeiro caso, é exemplificado no cenário anterior pelo atributo *b* da classe *A*. Como este é uma referência, é criado explicitamente através do comando *new*. O segundo caso, é exemplificado pelo atributo *a* da classe *B*. Como este não é uma referência, é criado quando seu pai (objeto da classe *B*) é criado.

Em todo construtor, um parâmetro é inserido para que o objeto criador lhe seja passado. Nas classes raízes de árvores hierárquicas, *A* e *C* no exemplo, o parâmetro é atribuído ao novo atributo. Nas outras classes, *B* no exemplo, o parâmetro é repassado, nos inicializadores dos construtores, para o construtor da superclasse.

PUC-Rio - Certificação Digital Nº 9816130/CA

Os construtores dos atributos regulares (criados quando o objeto pai é criado), atributo *a* da classe *B* no exemplo, são executados nos construtores das classes que os declara e o objeto que está sendo criado é passado como argumento extra.

Criações explícitas também são alteradas, sendo o objeto corrente da execução, que está criando explicitamente o outro, passado como argumento extra.

Atributos estáticos não possuem criador, dado que são criados em tempo de inicialização do sistema ou quando o primeiro objeto modelado pela classe é criado, dependendo da implementação da linguagem. Nestes casos, o valor *default* do atributo é utilizado e o novo atributo fica se referenciando ao *singleton pronomeNulo* que é fornecido como suporte de tempo de execução. Este é uma herança da classe CPronome, tendo portanto a implementação *default* para todos os tratadores de mensagens enviadas através de pronomes. Isso evita erros de execução caso um objeto estático envie erroneamente alguma mensagem através do pronome.

4.3

O Nível Meta

Para obter estas transformações, a todas as classes do nível base é associada a metaclasse *MCCreator*, a menos da classe *CPronome* que é associada a uma metaclasse especial denominada *MCRaizPronome*. O nível meta e sua associação com o nível base descrito anteriormente são detalhados na Figura 4.3.

Os nomes dos métodos que aparecem na Figura 4.3 são os definidos em OpenC++, podendo no entanto ser mapeados para qualquer outra linguagem aberta desde que suas características de reflexão sejam preservadas. Os métodos *Initialize(), RegisterNewAcessSpecifier(), RegisterNewClosureStatment(), TranslateClass(), BaseClasses(), ChangeBaseClasses(), TranslateUserStatment() e TranslateArguments()* da metaclass *Class* e os métodos *IsFunction()* e *GetUserAccessSpecifier()* da metaclass *Member* já foram descritos no Capítulo 3. Alguns, no entanto, estão sendo apresentados nesta transformação:

- void Class::AppendMember(Member& added_member, int specifier=Public): adiciona um novo membro a classe;

PUC-Rio - Certificação Digital Nº 9816130/CA

- Ptree* Class::TranslateInitializer(Environment* env, Ptree* var_name, Ptree* expr): transforma um inicializador de variável expr;

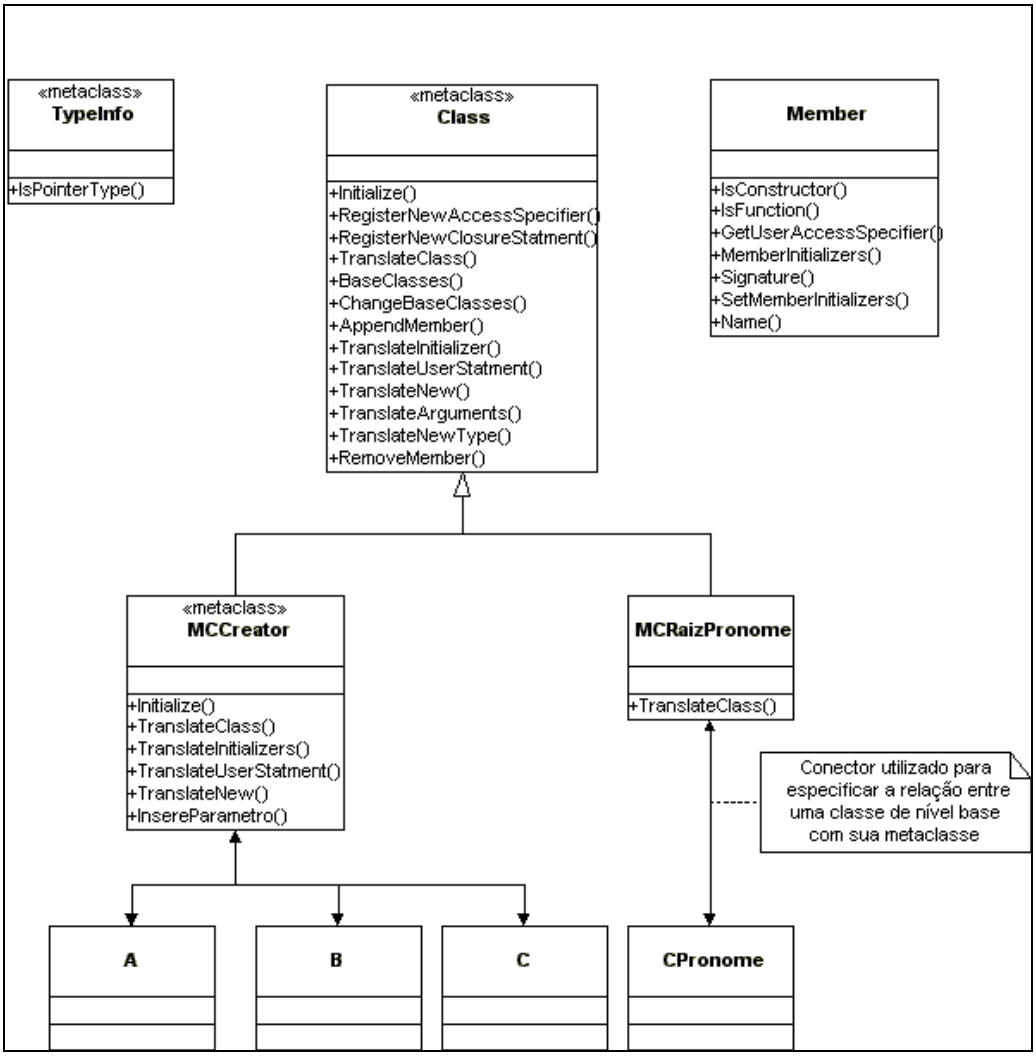

Figura 4.3 – Nível Meta para implementação do pronome CREATOR

- Ptree* TranslateNew(Environment* env, Ptree* header, Ptree* new_op, Ptree* placement, Ptree* type_name, Ptree* arglist): transforma uma expressão new;
- Ptree* Class::TranslateNewType(Environment* env, Ptree* type_name): transforma o nome de tipo incluído em uma expressão new;
- void Class::RemoveMember(Member& m): remove o membro especificado por m;
- bool Member::IsConstructor(): retorna true se o membro é um construtor;
- Ptree* Member::MemberInitializers(): retorna os inicializadores do membro se este for um construtor;

PUC-Rio - Certificação Digital Nº 9816130/CA

- void Member::Signature(TypeInfo& t): retorna, em t, o tipo do membro;
- void Member::SetMemberInitializers(Ptree* init): altera os inicializadores do membro, se este for construtor, para init;
- Ptree* Member::Name(): retorna o nome do membro;
- bool TypeInfo::IsPonterType(): retorna true se o tipo for uma referência.

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

- RegisterNewAcessSpecifier("message")
- RegisterNewClosureStatment("CREATOR")

As inserções da nova herança e do novo atributo nas classes raízes das hierarquias de herança, do parâmetro extra na declaração dos construtores e o armazenamento das mensagens enviadas através do pronome são feitos na redefinição do método *TranslateClass()* pela classe *MCCreator*.

Para o armazenamento das mensagens enviadas através do pronome, foi utilizado o objeto *Singleton* [G+95] *RepositorioMensagens*, descrito no capítulo anterior. Este, armazena as mensagens declaradas com o especificador <<message>> e as disponibiliza para ser introduzidas na classe *CPronome*.

As inserções do argumento extra nos inicializadores das superclasses e nas chamadas dos construtores dos atributos regulares são feitas pelo método *TranslateInitializers()*. A inserção do objeto corrente da execução como argumento extra na chamada de um construtor de uma classe é feita pelo método *TranslateNew()*. Por fim, a transformação do envio efetivo da mensagem através do pronome CREATOR é feita redefinindo-se o método *TranslateUserStatment()*. Este substitui no comando original a palavra-chave CREATOR pelo atributo *m_pCreator* que, pela transformação efetivada pelo método *TranslateClass()*, está disponível para toda a classe.

Ao término da transformação da última classe associada a *MCCreator*, todas as mensagens enviadas através do pronome CREATOR estão armazenadas.

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

PUC-Rio - Certificação Digital Nº 9816130/CA

4.4

O Algoritmo de Divisão e Conquista Utilizando CREATOR

O algoritmo paralelo de divisão e conquista, descrito na Subseção 2.4.2, pode ser implementado utilizando-se o pronome CREATOR. Esta implementação é esquematizada na Figura 4.4.

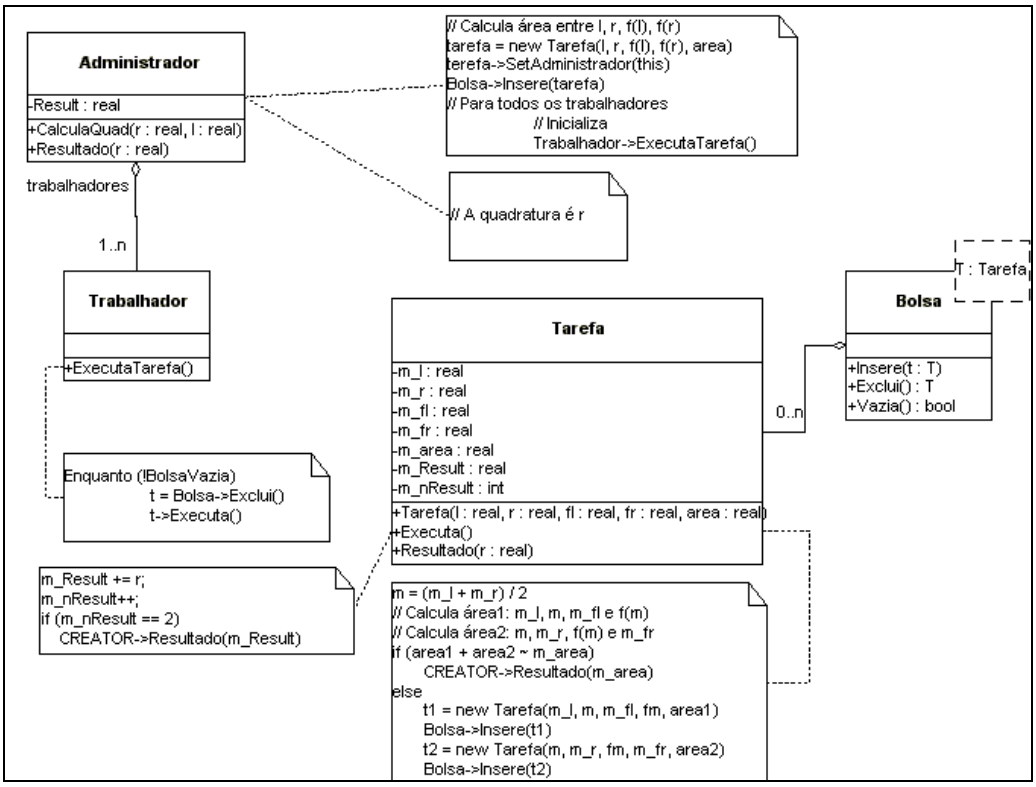

Figura 4.4 – Esquema da implementação do algoritmo paralelo de divisão e conquista utilizando CREATOR

Nesta solução, algumas facilidades são introduzidas e o problema da determinação do término, identificado na Subseção 2.4.2, é resolvido.

Ao contrário da implementação tradicional, agora as tarefas retornam o resultado não para o objeto administrador, mas sim ao objeto que o gerou. Desta forma, o atributo da classe *Tarefa* que guardava, na implementação apresentada anteriormente, a referência ao objeto administrador se torna desnecessário.

Por outro lado, cada tarefa que gera outras duas tarefas espera receber dois resultados e precisa ter um método para recebê-los. Este acumula o resultado em um novo atributo (*m_Result*) e controla o número de chamadas em outro novo atributo (*m_nResult*). Dois resultados são acumulados e a soma destes retornada ao objeto que criou a tarefa.

PUC-Rio - Certificação Digital Nº 9816130/CA

Como o objeto administrador gera uma tarefa, espera o retorno de um resultado e o algoritmo termina quando este é recebido. A classe Administrador não precisa mais, portanto, acumular resultados, tornando desnecessário o atributo onde isso era feito. Além disso, o código do método Resultado é simplificado para simplesmente retornar o resultado.

Além da solução do problema de determinação de término, uma referência ao objeto administrador não precisa mais ser exportado para as tarefas, diminuindo o acoplamento do sistema e a necessidade de mecanismos de proteção para a referência.

A desvantagem da utilização do pronome que é o aumento no tempo de execução do sistema devido à necessidade de manutenção da relação criador-criatura pode ser melhor avaliada observando-se a Tabela 4.1 e o gráfico mostrado na Figura 4.5. Nestes, os tempos de execução das implementações do algoritmo paralelo de divisão e conquista tradicional e utilizando o pronome CREATOR são comparados. Estes tempos foram conseguidos executando-se o algoritmo nas duas implementações e variando-se o número de objetos (não tarefas) criados sem participação direta na implementação do algoritmo. Isto é, a implementação do algoritmo permanece idêntica por toda a avaliação, sendo incluídos outros objetos dinâmicos. Imaginando-se a implementação do algoritmo como um módulo componente aplicação, a variação é feita incluindo-se novos objetos dinâmicos em outros módulos que não este.

A variação escolhida foi devido ao fato de ser exatamente neste tipo de construção que a implementação do pronome CREATOR adiciona código a ser executado que não tem influência direta no algoritmo e, com isso, aumenta seu tempo de execução sem incorporação de benefício.

Da mesma forma que para o pronome SENDER, cada configuração do exemplo foi executada 1000 vezes, sendo os tempos mostrados na Tabela 4.1 a média destas execuções. Os desvios padrões das amostras consideradas são apresentados na Tabela 4.2. O ambiente de execução destes experimentos foi o mesmo do utilizado para o pronome SENDER. Isto é, um computador Pentium4, com 256 MB de RAM, utilizando-se Linux Mandrake 8.1.

PUC-Rio - Certificação Digital Nº 9816130/CA

Número de objetos criados	Implementação em [JaJ92] (μs)	Implementação com CREATOR(μs)	Diferença (μs)	Aumento Percentual
0	155270	156509	1239	0,7979
1	153442	154773	1331	0,8674
10	157974	159748	1774	1,1229
100	157239	158579	1340	0,8522
1000	166069	168639	2570	1,5475
10000	218520	221313	2793	1,2781
100000	786076	788002	1926	0,2450
1000000	6147375	6327535	180160	2,9306
10000000	61463750	63674573	2201823	3,5817
100000000	624727492	637959997	13232505	2,1181
1000000000	6367665906	6522014253	154348347	2,4239
10000000000	65796960059	68341556771	2544596712	3,8673

Tabela 4.1 – Tempos de execução do algoritmo paralelo de divisão e conquista

PUC-Rio - Certificação Digital Nº 9816130/CA

Número de mensagens enviadas	Desvios Padrões Implementação Padrão	Desvios Padrões Implementação com CREATOR
0	0,2459	0,2472
1	0,2173	0,2959
10	0,2079	0,3361
100	0,3165	0,3275
1000	0,3801	0,2851
10000	0,2950	0,2511
100000	0,2749	0,2287
1000000	0,3039	0,2480
10000000	0,3174	0,2693
100000000	0,3210	0,3069
1000000000	0,2947	0,3193
10000000000	0,2517	0,2941

Tabela 4.2 – Desvios Padrões das amostras de tempo utilizadas para análise da execução do algoritmo paralelo de divisão e conquista

Os dados apresentados na Tabela 4.1 podem ser melhor visualizados na Figura 4.5.

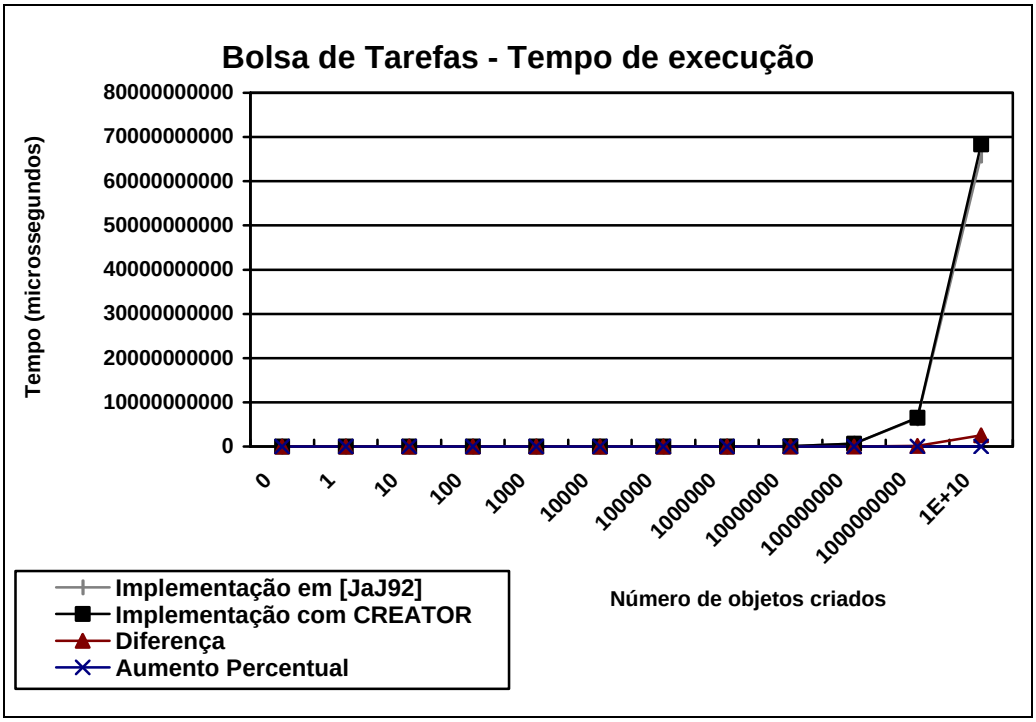

Figura 4.5 – Gráfico de comparação entre os tempos de execução do algoritmo paralelo de divisão e conquista

Na Tabela 4.1 e no gráfico mostrado na Figura 4.5 pode-se verificar que o tempo de execução na implementação do padrão utilizando-se o pronome CREATOR é sempre um pouco maior, o que já era esperado. O aumento percentual máximo obtido, no entanto, foi de 4%, evidenciando a não degradação do sistema com o uso do pronome.

PUC-Rio - Certificação Digital Nº 9816130/CA