Trabalho de MAC413 02/10/2002
Tópicos de Programação Orientada a Objetos

Alunos:	N^o. USP
Fernando Alberto de Sousa	3286224
Luiza Figueiredo Pagliari	3286440

Padrão Comportamental

Clone

1. Objetivo / Intenção

Prover uma maior eficiência na realização de tarefas de objetos muito requisitados.

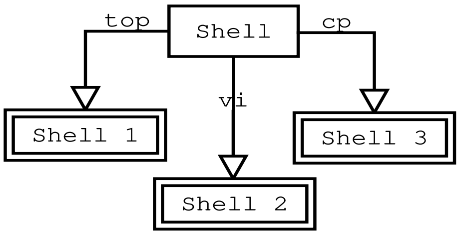
2. Também conhecido como

Fork

3. Motivação

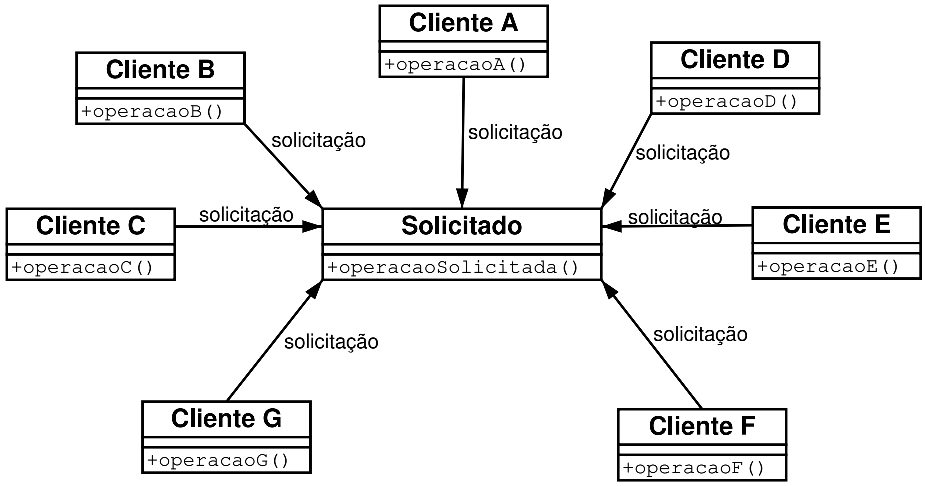
Considere a hierarquia de processos de um Sistema Operacional. Em diversas situações, o sistema ou o usuário podem decidir executar novas tarefas relacionadas a outras pré-existentes, como, por exemplo, programas executados a partir de um shell. Neste caso, para executar duas ou mais tarefas simultaneamente é necessário um shell para cada tarefa. Se o usuário de um sistema Linux desejasse executar o monitoramento da tabela de processos (comando "top"), copiar arquivos para disquete e editar um arquivo em um editor de texto (vi, por exemplo), ele teria que inicializar manualmente cada um dos três shells, ou ainda esperar que um dos processos termine para inicializar outro. A idéia de colocar dois ou três processos em background não é ruim, porém não mantém vínculos entre os processos e em alguns casos não possibilita o acompanhamento da execução (no caso do "top", a visualização da tabela de processos só é possível em foreground).

Seria interessante, portanto, prover um meio para que as requisições sejam atendidas por novos processos semelhantes entre si, de maneira a não perder o vínculo entre eles. Uma solução é a clonagem do shell original até serem gerados novos shells em número suficiente para que todas as tarefas sejam executadas.

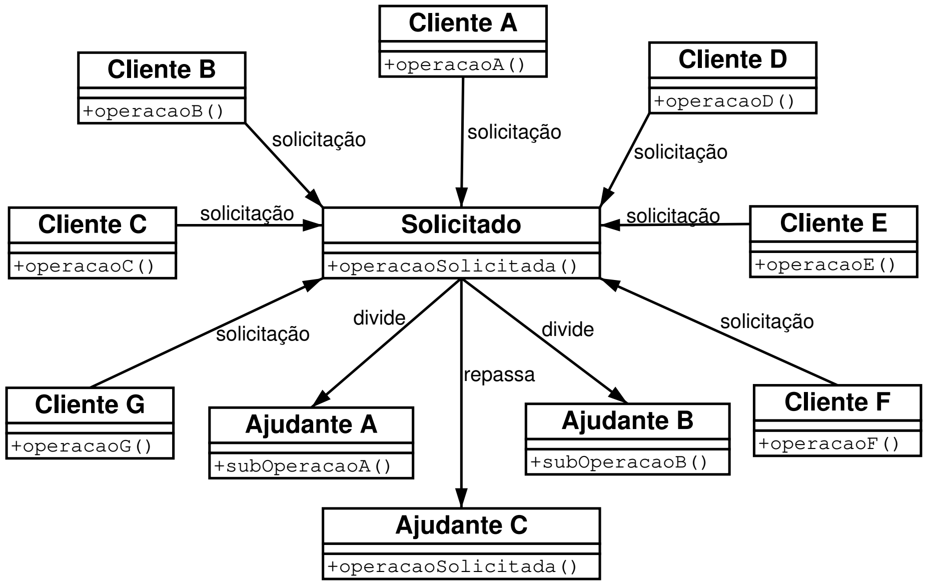


Poderíamos pensar também em um sistema distribuído com objetos recebendo múltiplos acessos para a realização de tarefas.

Representação de um objeto muito requisitado:



Alternativamente poderíamos ter a delegação das operações realizadas pelo objeto para outros objetos capazes de realizar as mesmas tarefas (ou subtarefas).



Para garantir a existência de objetos capazes de realizar tais operações, podemos aplicar a clonagem do objeto requisitado originalmente.

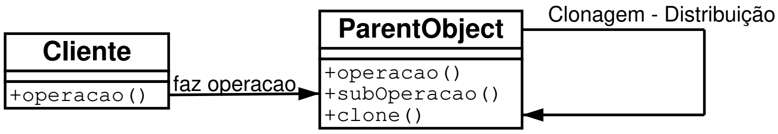
O processo de criação dos clones poderia ser realizado remotamente, assim teríamos um melhor aproveitamento de recursos.

4. Aplicabilidade

O padrão Clone pode ser usado quando:

- *um ou mais objetos do sistema receberão múltiplas requisições* - isto ocorre com grande freqüência em sistemas baseados em computação paralela e distribuída nos quais podem ocorrer chamadas remotas;
- *deseja-se obter uma maior eficiência através da divisão de tarefas* - em muitos casos as operações realizadas por um objeto podem ser divididas em subtarefas independentes, ou com um nível bem definido de dependência (informação necessária para a tarefa parmanece inalterada durante a realização da mesma, assim basta obter uma fonte estática de informação - informação inicial do objeto "pai"), assim, tais tarefas podem ser realizadas por objetos distintos;
- *o sistema deve aproveitar melhor os recursos disponíveis pelo ambiente de execução* - o ambiente de execução de um sistema pode prover recursos como os serviços de rede (sistema distribuído) que poderiam colaborar para a performance do mesmo.

5. Estrutura



6. Participantes

- **ParentObject**

  define o objeto alvo do cliente que receberá múltiplas requisições e que, para determinadas operações, delega solicitações de clientes para cópias de si mesmo (clones), ou divide a operação solicitada em suboperações que serão delegadas.

- **Cliente**

  faz requisições às instâncias de ParentObject, não tem conhecimento de qual instância realiza, de fato, a operação solicitada.

7. Colaborações

Os clientes se comunicam apenas com uma instância (o pai) de ParentObject, independemente de qual instância realizará realmente a operação desejada.

O ParentObject determina quais operações solicitadas pelos clientes serão delegadas para novas instâncias de si mesmo.

8. Conseqüências

O padrão *Clone* tem os seguintes benefícios e desvantagens:

1. ***Ele provê maior eficiência.***
   A divisão de tarefas proporcionada pela aplicação deste padrão pode conferir uma maior eficiência, uma vez que uma tarefa que seria originalmente realizada por um único objeto, poderá ser fracionada em subtarefas a serem distribuídas para um grupo de objetos realizadores, ou ainda, a tarefa poderá ser repassada para um objeto livre de requisições.

2. ***Ele possibilita um melhor aproveitamento dos recursos.***
   Uma vez que a realização de tarefas é fracionada, aumenta-se o número de objetos realizadores, assim, além de um possível melhor aproveitamento de memória, poderemos ter a utilização de serviços remotos para a criação e utilização dos novos objetos.

3. ***Ele evita sobrecarga de objetos.***
   Com a criação dos clones determinadas operações serão postergadas ou divididas para uma distribuição posterior, assim, o objeto originalmente requisitado pelo cliente não terá de realizar todo o trabalho sozinho, o que possibilita uma redução no tempo de resposta do mesmo e promove uma distribuição mais equilibrada de atividades.

4. ***Pode acarretar em sobrecarga de distribuição de tarefas.***
   Apesar da distribuição de tarefas colaborar para uma otimização do processo operacional, a distribuição em si pode se tornar uma tarefa muito custosa para um objeto com um número elevado de requisições. Desta forma, o objeto original poderia ser sobrecarregado de qualquer forma, uma vez que ele é o único responsável pela divisão e distribuição de suas operações.

5. ***Leva a um consumo considerável de memória***.
   Podemos ter um elevado número de novos objetos criados para a distribuição das tarefas, acarretando em um eventual considerável aumento no consumo de memória, podendo até ocasionar o inoperância do sistema.

6. ***É difícil estabelecer um bom critério de divisão de tarefas***.
   O objeto requisitado ParentObject deve ser muito criterioso para promover a criação de um clone, uma vez que, muitas vezes, as operações divididas e/ou delegadas podem não corresponder às operações mais solicitadas. Desta forma, os benefícios da aplicação do padrão não seriam relevantes e além disso, poderíamos promover um aumento desnecessário no consumo de memória e de outros recursos.

9. Implementação

Para a implementação do padrão Clone, o desenvolvedor deve considerar alguns aspectos importantes:

1. Critério de divisão de tarefas
   A escolha do critério para a criação de um novo clone para a realização de tarefas é o alicerce da aplicação do padrão, assim deve-se conhecer de antemão o conjunto de operações mais requisitadas do objeto.

2. Cópia por referência versus cópia por replicação
   Definir como o novo objeto será criado pode ser particularmente difícil quando as estruturas de objetos contêm referências circulares. A maioria das linguagens fornece algum suporte para clonagem de objetos, como Smalltalk que provê a mensagem *copy* para todas as subclasses de Object e C++, onde existe um contrutor *copy.* No entanto, ambas fornecem cópias por referência, ou seja, os apontadores são compartilhados entre a cópia e o objeto original. O problema em se fazer isto é que normalmente se deseja independência entre o clone e o original, o que só pode ser obtido através de uma cópia por replicação. A implementação de uma nova operação de clonagem possibilita a escolha do que deverá, ou não, ser compartilhado. Eventualmente os objetos a serem clonados possuem operações de *salvar* e *carregar*, as quais podem ser usadas para implementar a operação de clonagem, simplesmente salvando o objeto e carregando-o de volta imediatamente. A operação *salvar* salva um objeto em um buffer de memória, e a operação *carregar* cria uma cópia por reconstrução do objeto a partir do buffer.

3. Paralelismo de Operações
   Um outro ponto a ser observado é que, para dividirmos uma operação em suboperações paralelas, cada objeto executor deve pertencer a uma linha de execução específica (thread).

10. Usos Conhecidos

Sistemas Operacionais

O padrão Clone está presente na criação de processos em Sistemas Operacionais, onde, a partir de um único processo inicial, criam-se todos os processos de sistema e usuário – através do fork. Fork é o único meio de criar um processo novo. Ele cria uma duplicata exata do processo original, incluindo todos os descritores de arquivos, registradores, etc. e, depois, o processo original e a cópia (pai e filho) seguem caminhos diferentes. Todas as variáveis têm valores idênticos no momento do fork, mas como os dados do pai são copiados para criar o filho, posteriores mudanças em um deles não afeta o outro. A informação imutável é compartilhada entre pai e filho.

Na maioria dos casos, depois de um fork, o filho precisará executar um código diferente do pai, o que caracteriza uma delegação de tarefa.

Computação Paralela

Sistemas que necessitam de muitos cálculos, como a previsão do tempo, são freqüentemente implementados para rodar em paralelo, uma vez que podem ser subdivididos em frações do mesmo problema (divisão por área de superfície, por exemplo). Dessa forma, um processo inicial é clonado até ser obtida uma quantidade razoável de cópias para rodar o programa satisfatoriamente, aproveitando melhor os recursos e terminando o programa em tempo hábil.

11. Padrões Relacionados

Clone é útil para projetos que utilizem Prototype (121), onde a classe ParentObject seria um dos ConcretePrototype, tendo, inclusive, a operação de copiar a si mesma já implementada.

O processo de clonagem de ParentObject pode ser implementado utilizando AbstractFactory (95), de modo a isolar a criação da cópia de suas funcionalidades.