INF011 – Padrões de Projeto

15 – *Proxy*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Proxy

- Propósito:
  - Disponibilizar um substituto para outro objeto, controlando o acesso a ele
- Também conhecido como: *Surrogate*
- Motivação:
  - Adiar o custo de criação e inicialização de um objeto até o momento em que ele é realmente necessário
  - Ex: um editor de textos pode não carregar todas as imagens do texto no momento em que é aberto, somente aqueles atualmente visíveis na aplicação
  - Cria-se os objetos sob demanda

Proxy

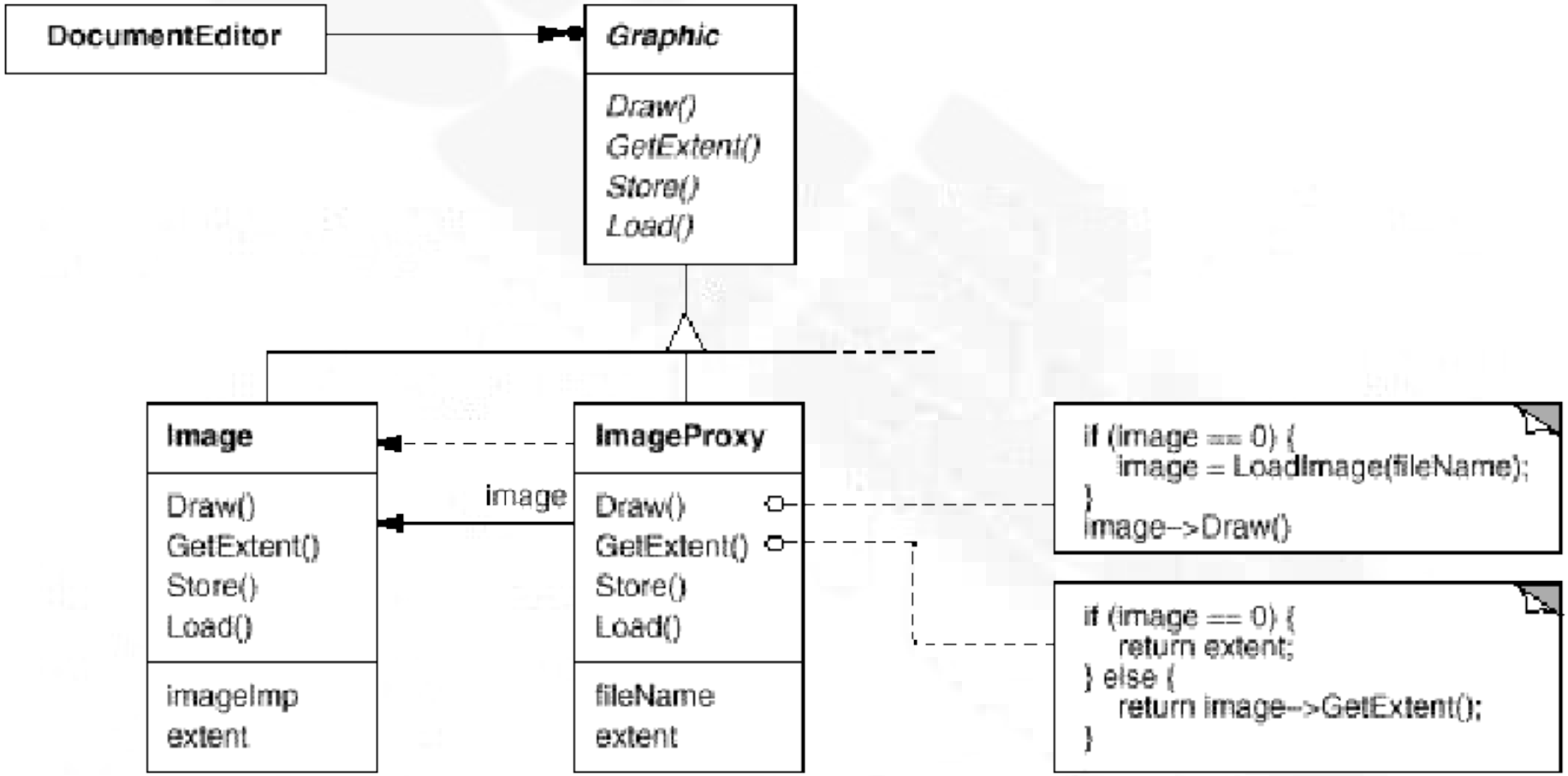
- Motivação:
  - Entretanto, o que colocar no lugar da imagem para indicar a sua presença no texto ?
  - Como pode-se abstrair a criação sob demanda da imagem, de modo que o código da implementação do editor (ex: renderização e formatação de texto) seja mantido simples ?
  - A solução é utilizar um objeto **procurador** (*proxy*) que atua no lugar da imagem real

Proxy

- Motivação:
  - O *proxy* cria a imagem real somente quando o editor solicita a sua exibição
  - O *proxy* armazena uma referência para a imagem e as suas dimensões, de modo que o algoritmo de formatação funcione mesmo sem conhecer a imagem real

Proxy

- Motivação:



Proxy

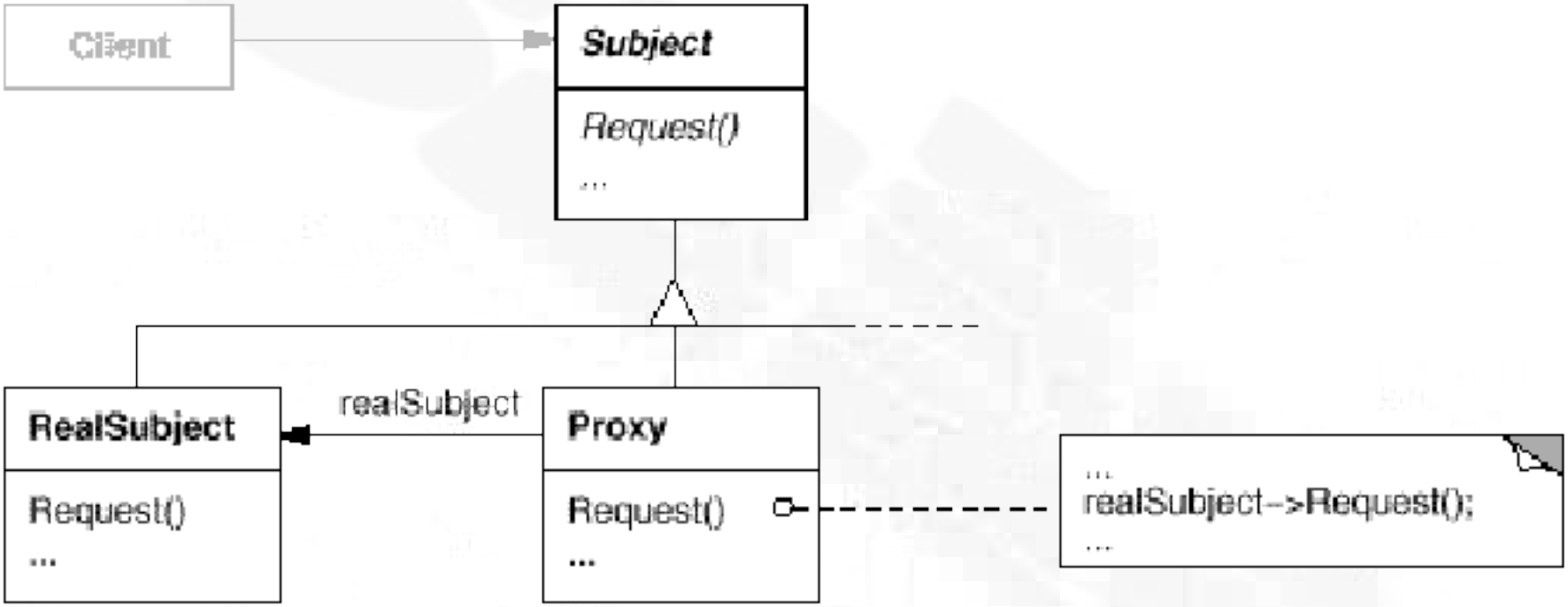
- Aplicabilidade:
  - Útil sempre que existe a necessidade de uma referência para objeto que seja mais sofisticada e versátil do que um ponteiro simples. Alguns exemplos:
    - *Remote Proxy* (*Ambassador*)*:* disponibiliza um representante local para um objeto que reside em um espaço de endereçamento diferente
    - *Virtual Proxy*: cria objetos custosos sob demanda
    - *Protection Proxy*: controla o acesso ao objeto original. Útil quando objetos precisam ter direitos de acesso diferentes

Proxy

- Aplicabilidade:
  - Útil sempre que existe a necessidade de uma referência para objeto que seja mais sofisticada e versátil do que um ponteiro simples. Alguns exemplos:
    - *Smart Reference* (*Smart Pointers*): alternativa aos ponteiros convencionais, realizando ações adicionais no momento em que o objeto é acessado. Por exemplo:
      - Contagem do número de referências para o objeto real
      - Carregamento de um objeto persistente na memória no primeiro momento em que ele é acessado
      - Verificação do bloqueio (*locking*) do objeto real antes do acesso para garantir que nenhum outro objeto o acessará simultaneamente

Proxy

- Estrutura:



Proxy

- Participantes:
  - *Proxy* (ImageProxy):
    - Mantém uma referência que permite o acesso ao objeto real. Pode se referir a *Subject* se as interfaces *Subject* e *RealSubject* são as mesmas
    - Disponibiliza uma interface idêntica à do *Subject*, de modo que seja um substituto do objeto real
    - Controla o acesso ao objeto real e pode ser responsável pela sua criação e remoção

Proxy

- Participantes:
  - *Proxy* (ImageProxy):
    - Outras responsabilidades dependem do tipo do *proxy*:
      - *Remote Proxies*: responsáveis pela codificação da requisição e seus argumentos e pelo envio destes para o *subject* real, residindo em um espaço de endereçamento diferente
      - *Virtual Proxies*: pode realizar *caching* de informações adicionais sobre o *subject* real de modo a adiar o acesso a ele
      - *Protection Proxies*: verifica se o componente que realiza a invocação possui as permissões de acesso requeridas
  - *Subject* (Graphics):
    - Define uma interface comum para *RealSubject* e *Proxy*, de modo que um *Proxy* possa ser utilizado naqueles lugar onde um *RealSubject* era esperado

Proxy

- Participantes:
  - *RealSubject* (Image):
    - Define o objeto real que é representado pelo *Proxy*

Proxy

- Colaborações:
  - O *Proxy* repassa as requisições para o *RealSubject* quando apropriado, dependendo do tipo de *proxy*

Proxy

- Consequências:
  - O *Proxy* introduz um nível a mais de indireção, com objetivos diversos:
    - O *Remote Proxy* esconde o fato que objetos residem em um espaço de endereçamento diferente
    - O *Virtual Proxy* pode realizar otimizações tais como criar o objeto sob demanda
    - Tanto o *Protection Proxy* quando o *Smart Reference* permitem a realização de tarefas de limpeza do ambiente quando o objeto é acessado
  - Outra otimização importante: *copy-on-write*

Proxy

- Implementação:
  - Sobrecarregando o *operator->* em C++:
    - Permite a utilização do *proxy* exatamente como um ponteiro:

```
class Image;
extern Image* LoadAnImageFile(const char*);
    // external function

class ImagePtr {
public:
    ImagePtr(const char* imageFile);
    virtual ~ImagePtr();

    virtual Image* operator->();
    virtual Image& operator*();
private:
    Image* LoadImage();
private:
    Image* _image;
    const char* _imageFile;
};
```

Proxy

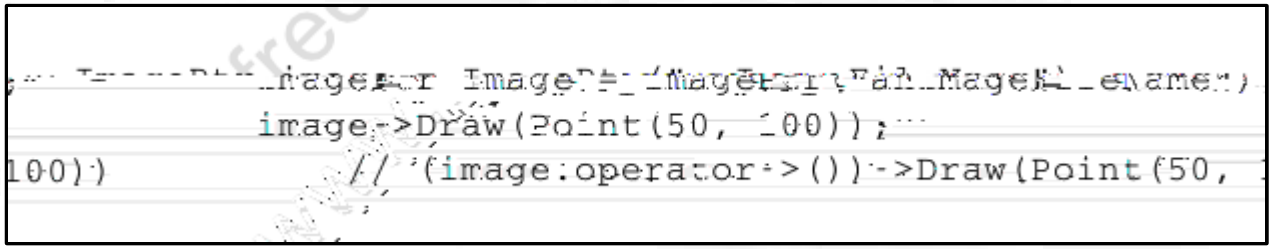
- Implementação:
  - Sobrecarregando o *operator*-> em C++:
    - Permite a utilização do *proxy* exatamente como um ponteiro:

```
ImagePtr::ImagePtr (const char* theImageFile) {
    _imageFile = theImageFile;
    _image = 0;
}

Image* ImagePtr::LoadImage () {
    if (_image == 0) {
        _image = LoadAnImageFile(_imageFile);
    }
    return _image;
}
```

```
Image* ImagePtr::operator-> () {
    return LoadImage();
}

Image& ImagePtr::operator* () {
    return *LoadImage();
}
```

Proxy

- Implementação:
  - Sobrecarregando o *operator*-> em C++:
    - Permite a utilização do *proxy* exatamente como um ponteiro:



Proxy

- Código exemplo:

```
class Graphic {
public:
    virtual ~Graphic();

    virtual void Draw(const Point& at) = 0;
    virtual void HandleMouse(Event& event) = 0;

    virtual const Point& GetExtent() = 0;

    virtual void Load(istream& from) = 0;
    virtual void Save(ostream& to) = 0;
protected:
    Graphic();
};
```

Proxy

- Código exemplo:

```
class Image : public Graphic {
public:
    Image(const char* file);  // loads image from a file
    virtual ~Image();
    virtual void Draw(const Point& at);
    virtual void HandleMouse(Event& event);

    virtual const Point& GetExtent();

    virtual void Load(istream& from);
    virtual void Save(ostream& to);
private:
    // ...
};
```

Proxy

- Código exemplo:

```
class ImageProxy : public Graphic {
public:
    ImageProxy(const char* imageFile);
    virtual ~ImageProxy();

    virtual void Draw(const Point& at);
    virtual void HandleMouse(Event& event);

    virtual const Point& GetExtent();

    virtual void Load(istream& from);
    virtual void Save(ostream& to);
protected:
    Image* GetImage();
private:
    Image* _image;
    Point _extent;
    char* _fileName;
};
```

e-ebooks-download.org

Proxy

- Código exemplo:

```
ImageProxy::ImageProxy (const char* fileName)  {
    _fileName = strdup(fileName);
    _extent = Point::Zero;  // don't know extent yet
    _image = 0;
}

Image* ImageProxy::GetImage() {
    if (_image == 0) {
        _image = new Image(_fileName);
    }
    return _image;
}
```

Proxy

- Código exemplo:

```
const Point& ImageProxy::GetExtent () {
    if (_extent == Point::Zero) {
        _extent = GetImage()->GetExtent();
    }
    return _extent;
}

void ImageProxy::Draw (const Point& at) {
    GetImage()->Draw(at);
}

void ImageProxy::HandleMouse (Event& event) {
    GetImage()->HandleMouse(event);
}
```

Proxy

- Usos conhecidos:
  - NEXSTEP: *remote proxies*
  - Qt4
  - *LibBoost*
  - STL (*Standard Template Library*)

Proxy

- Padrões relacionados:
  - O *Adapter* disponibiliza, para o objeto que ele adapta, uma interface diferente. O *Proxy* disponibiliza a mesma interface
  - Embora o *Decorator* tenha uma implementação similar o propósito é diferente. O *Decorator* adiciona uma ou mais responsabilidades a um objeto enquanto o *Proxy* controla o acesso ao objeto

INF011 – Padrões de Projeto

15 – *Proxy*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas