Relatório

Projecto de Reutilização de Software

SimpleFacebook

Fábio Miguel Ferreira Pedrosa (2006124898 - fmfp@student.dei.uc.pt)

Marco António Machado Simões (2006125287 - msimoes@student.dei.uc.pt)

Especificação do Conceito

Neste projecto decidimos não optar pela especificação dada pelo professor e elaborar uma outra aplicação onde aplicar as diversas design patterns.

A aplicação que decidimos elaborar baseia-se no *Facebook (www.facebook.com)*, uma plataforma online do tipo rede social, com algumas simplificações. Demos-lhe o nome de *SimpleFacebook*. Utilizar uma plataforma famosa pela sua complexidade e existente no mercado actual deverá oferecer mais casos práticos onde as patterns podem oferecer flexibilidade e capacidade de reutilização, bem como uma referência da qualidade que se pretende obter.

Essencialmente trata-se de uma aplicação em que os utilizadores estão ligados remotamente a um servidor centralizado onde conseguem ver, gerir e interagir com a sua rede social. A interface gráfica oferece a possibilidade de gerir a lista de amigos (ver, adicionar, remover e confirmar amizades), adicionar conteúdo à sua página pessoal (chamada de mural ou *wall*), visualizar a mural dos outros utilizadores, fazer pesquisas de conteúdo, entre outras funcionalidades.

Cada utilizador pode colocar na sua mural um conjunto de items de vários tipos (texto, foto ou vídeo) e são automaticamente colocados para visualização na sua mural pessoal. Qualquer utilizador da aplicação pode visualizar a mural de outro utilizador, bem como visualizar na sua página principal uma mural composta por todos os seus próprios items e os items dos seus amigos directos. Cada item aceita comentários de qualquer utilizador.

Estrutura

O SimpleFacebook têm uma arquitectura Cliente-Servidor em que o cliente pode funcionar utilizando diferentes protocolos de comunicação (suporta http e java objects actualmente) e interface gráfica (suporta html e swing actualmente). Este suporte foi criado propositadamente de forma a explorar a ideia de criar uma linguagem abstracta de definição de interface gráfica, permitindo vários suportes de visualização (suporta html e swing actualmente). A aplicação começa por iniciar dois diferentes servidores, um que escuta por ligações na porta 80 para clientes que utilizam o protocolo HTTP (browsers web), e porta 81 para clientes swing (foi criado um cliente standalone exemplar).

Cliente HTTP / HTML

Para aceder ao SimpleFacebook através do protocolo HTTP basta apontar um browser para o endereço do servidor, e navegar utilizando a interface criada pelo layout HTML. HTML é uma linguagem markup standard para os browsers web. É descrita em forma textual, num formato XML (hierárquico).
Geralmente segue uma estrutura como esta:

```
<html>
    <head>
        <title>Título da página</title>
    </head>
    <body>
```

```
            <p>Isto é um parágrafo</p>
            <img src=”caminho para uma imagem” />
        </body>
</html>
```

Cliente Java Objects / Swing

Para aceder ao SimpleFacebook utilizando objectos java foi criado um cliente standalone (SwingApp) que conecta com o servidor da aplicação e faz pedidos enviando um objecto java serializado (ClientRequest) ao qual o servidor responde com um objecto de resposta (SwingResponse) constituído por um objecto gráfico Swing que foi construído no servidor e precisa apenas de ser mostrado ao utilizador. Swing é uma API gráfica baseada no AWT do java, que permite criar interfaces gráficas no desktop nos vários sistemas operativos suportados pelo java. Um layout exemplar pode ser criado com código como este:

```
JPanel painel = new JPanel(new BorderLayout());
painel.add(new JLabel(“Este é um texto no painel”));
painel.add(new JLabel(new ImageIcon(new URL(“http://www.caminho_de_uma_imagem”)));
JFrame frame = new JFrame(“Janela da interface);
frame.setContentPanel(painel);
frame.setVisible(true);
```

Servidor

Para criar a resposta aos pedidos em ambos tipos de protocolos e API’s gráficas foi elaborado uma linguagem abstracta para definir a interface gráfica com elementos básicos (página, botão, parágrafo, etc) que pode depois ser convertida para a interface pretendida. Esta abstração é possível utilizando a pattern Builder com a estrutura desenvolvida utilizando o pattern Composite. No caso particular da interface HTML foi necessário incluir a pattern Visitor devido á necessidade de criar a interface em dois passos (um passo para criar a hierarquia, outro para converter para o código HTML) ao contrário da interface Swing que é criada num só passo. Mais detalhe será dado na descrição do pattern na próxima secção.

Arquitectura do Servidor

Para permitir uma maior reutilização das classes utilizadas no servidor de ambos protocolos, foi utilizada uma pattern arquitectural chamada **Model-View-Controller (MVC).** Esta pattern isola em três camadas a lógica de tratamento dos pedidos ao servidor em *Modelo* (dados da aplicação), *Vista* (obtém os dados dos modelos e apresenta) e *Controlador* (recebe os pedidos e contrói a resposta com a vista indicada).

O fluxo de tratamento dos pedidos é o seguinte:

- O servidor de um dos protocolos recebe um pedido utilizando o respectivo protocolo e passa a uma thread para tratar o pedido;
- É criado um *Builder* para a interface que é utilizada para este protocolo (swing para ligações de java objects, html para ligações http);
- Utilizando a pattern *Command* é invocado o devido *Controller* ao qual é passado os parâmetros do pedido, juntamente com o builder criado;
- O controller verifica a lógica do pedido garantindo que está logado se necessário, carrega os dados necessários para a vista e faz as devidas alterações nos modelos quando necessário, e conclui por passar á vista indicada o builder;

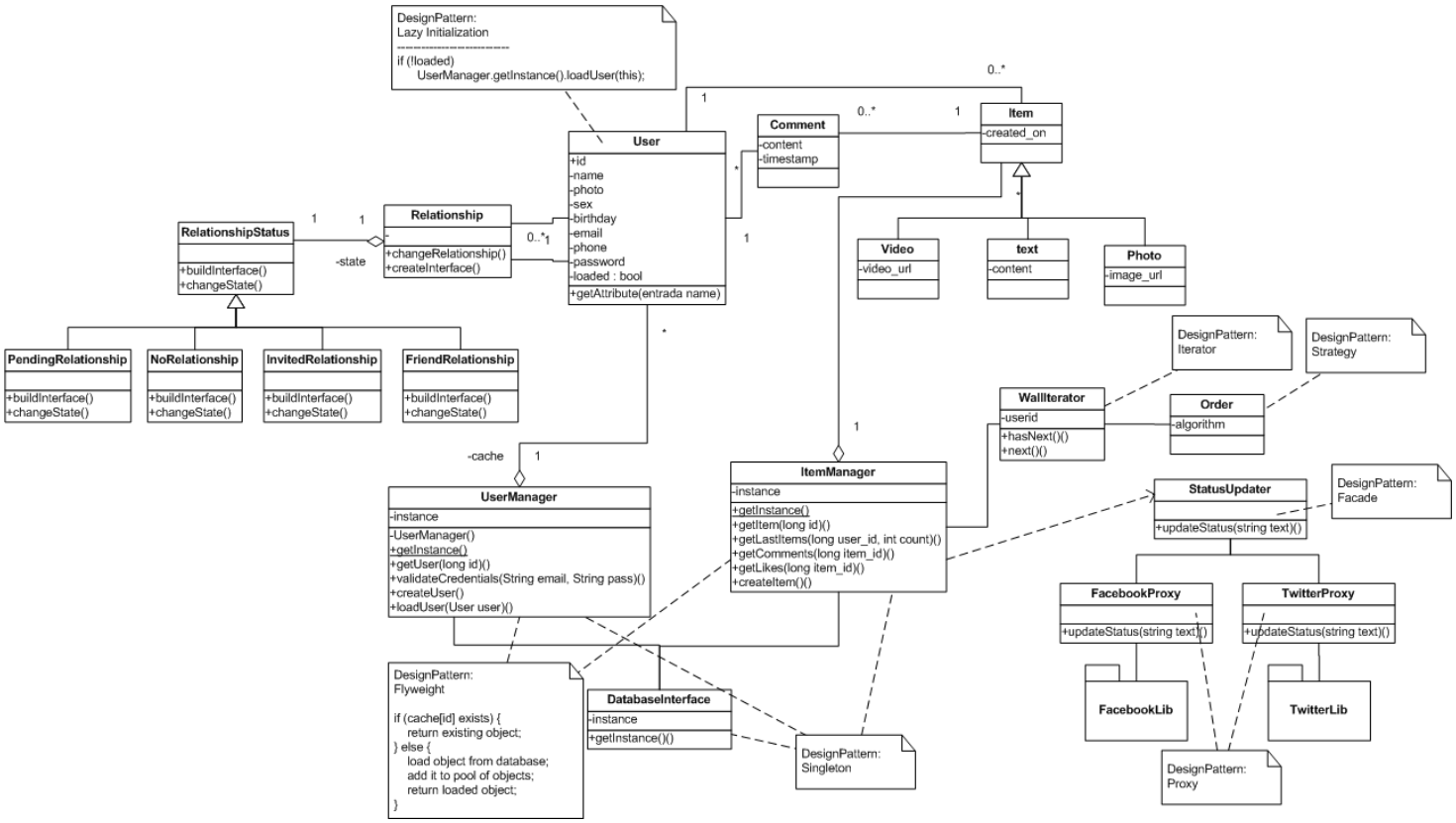
- A vista contrói utilizando o builder recebido, bem como os dados necessários, uma descrição da interface que se pretende mostrar ao utilizador;
- O controlo retorna à thread de tratamento do pedido e o resultado do builder é utilizado para responder ao cliente.

Com este fluxo é permitido adaptar cada uma das camadas de uma forma simples e isolada.

Design do Sistema

Overview Geral

O diagrama abaixo apresenta a estrutura geral do SimpleFacebook:

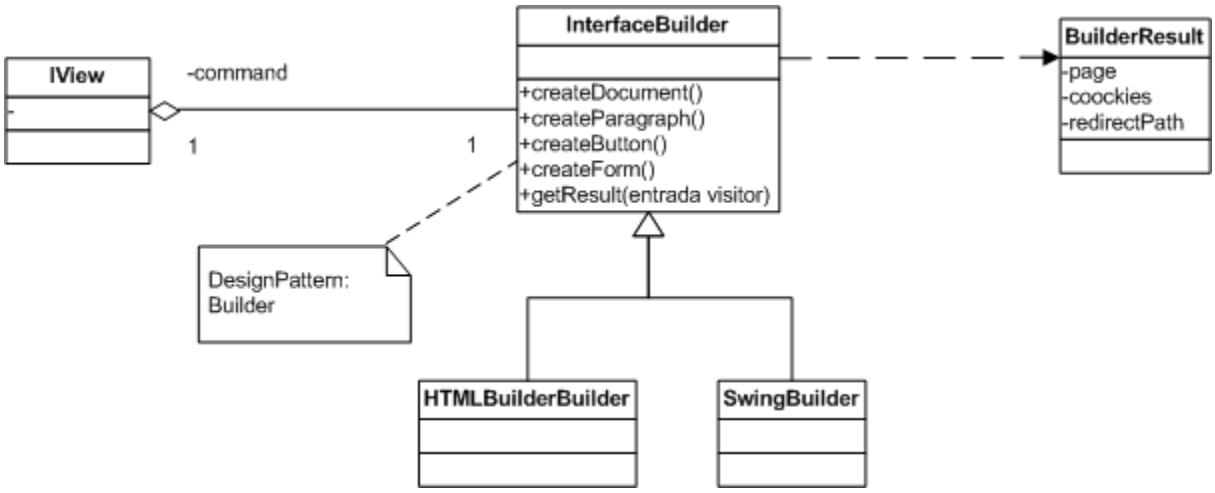


Design Patterns

Criacionais

Builder

A pattern builder permite abstrair os passos de contrução de objectos de forma a que várias implementações destes passos possam construir diferente representações de objectos. Esta situação verifica-se com perfeição na criação de diferentes interfaces gráficas utilizando os mesmos passos de construção.

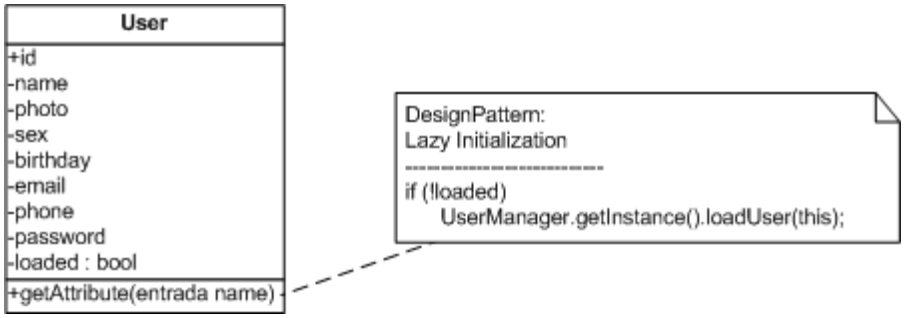


A classe InterfaceBuilder define as diferentes primitivas de construção de uma interface gráfica (createButton, createPage, createParagraph, etc) para o qual existem diferentes sub-classes implementando de forma diferentes as várias primitivas. No *SwingBuilder* o *createPage* é implementado criando uma painel Swing (JPanel) mas no *HTMLBuilder* o *createPage* é implementado criando uma hierarquia de objectos subclasse de Element que permitem criar a hierarquia da interface (*Composite*) e mais tarde ser traduzida para código HTML (*Visitor*).

Lazy Initialization

Existe uma grande dependência entre os vários modelos de dados no SimpleFacebook, especialmente nas classes User e Item. Por exemplo para apresentar a página de um utilizador é preciso carregar os dados desse utilizador, bem como a lista dos seus amigos. Como é preciso algumas informações destes amigos para serem apresentados, é necessário carregar alguma informação básica destes amigos (foto e nome por exemplo), mas não é uma tarefa leve carregar todos os dados desse utilizador para este objectivo. A forma que se encontrou para minimizar os carregamentos de informação foi implementar a pattern de *Lazy Initialization* nestas classes muito interligadas.

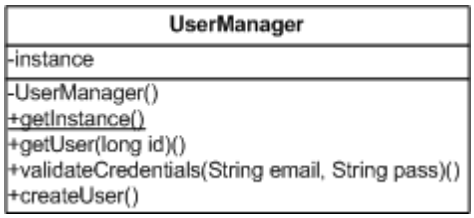
Utilizando esta pattern, a restante informação do utilizador bem como a lista de amigos é apenas requesitada à base de dados no momento em que é realmente necessária para ser apresentada. A mesma pattern aplica-se na classe *Item* que apenas carrega os seus comentários quando realmente surgir essa necessidade. A imagem abaixo mostra o diagrama para a classe *User*:



Esta pattern não está documentada no livro utilizado como referência principal da cadeira, mas pode ser encontrado em: *Fowler, Martin (2002). Patterns of Enterprise Application Architecture. Addison-Wesley. ISBN*

978-0321127426.

Singleton



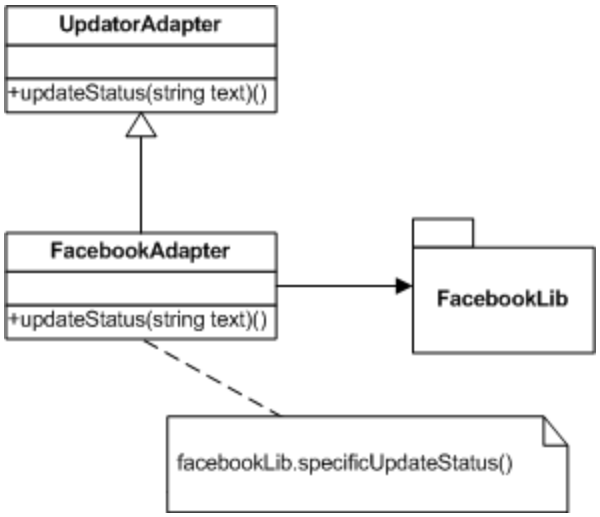
As classes *ItemManager* e *UserManager* sao classes de apoio á aplicação e não necessitam de várias instancias, apenas de uma que pode ser acedida de forma global. Ambas fornecem acesso ao modelo de dados, para os items e utilizadores (respectivamente) e a criação de várias instâncias não seriam necessárias por não existir um estado interno a estas que as fizesse diferir.

Estruturais

Adapter

Uma das funcionalidades do SimpleFacebook é permitir ao utilizadores partilhar com os seus amigos conteúdo de vários tipos (texto, foto ou vídeo).

Quando o utilizador partilha texto, é possível se o utilizador o permitir, enviar esta publicação para outros serviços online semelhantes (Twitter e Facebook por exemplo). Para cada um destes serviços foi criado uma *Adapter* que traduz a interface de utilização da biblioteca java do serviço para uma interface compatível com a lógica da aplicação (TwitterAdapter e FacebookAdapter respectivamente). Ambos adaptadores são subclasse da interface *UpdateAdapter* que generaliza a interface para serviços deste género.

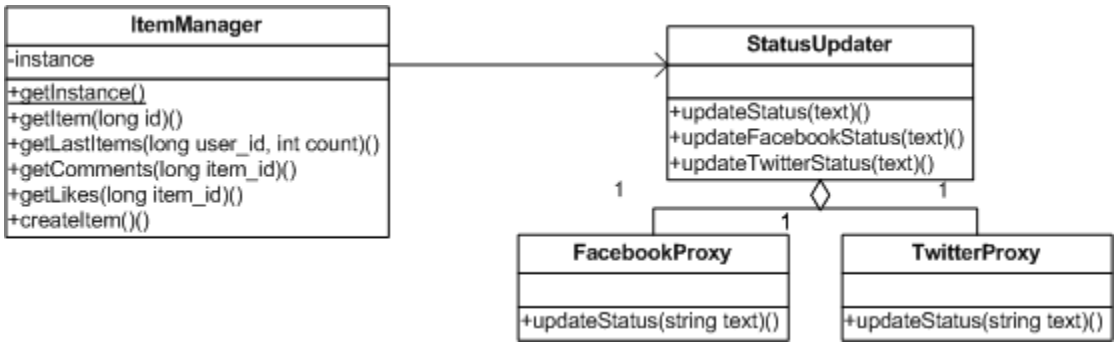


Facade

Com a inclusão de vários adaptadores de interface com bibliotecas de serviços online, era importante criar um ponto único de acesso a estes e isolar a quantidade de adaptadores deste tipo à funcionalidade de publicar

nos serviços.

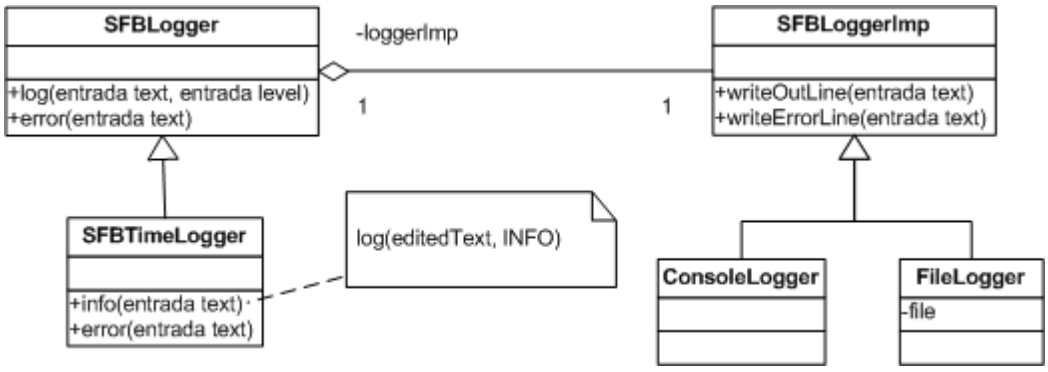
A classe *StatusUpdater* faz este ponto-único através da disponibilização de métodos que abstraiem os vários adaptadores que existem, bem como as características de cada API (autenticação, pedidos assíncronos, etc).



Bridge

Criámos um sistema de Logging no qual quisemos permitir abstrair quer o Logger quer a sua implementação. Usámos uma bridge para tal. A class SFBLogger define os métodos base para qualquer logger: fazer log com um determinado nivel e fazer log de um erro. Qualquer class que herde desta pode usar estes métodos e extendê-los para novos métodos. Criámos, por exemplo, o SDBTimeLogger que adiciona a data e a hora a todos os logs. Assim, a função info por exemplo adiciona ao texto de entrada a hora e chama o metodo do pai “log” com o novo texto e o nível de info.

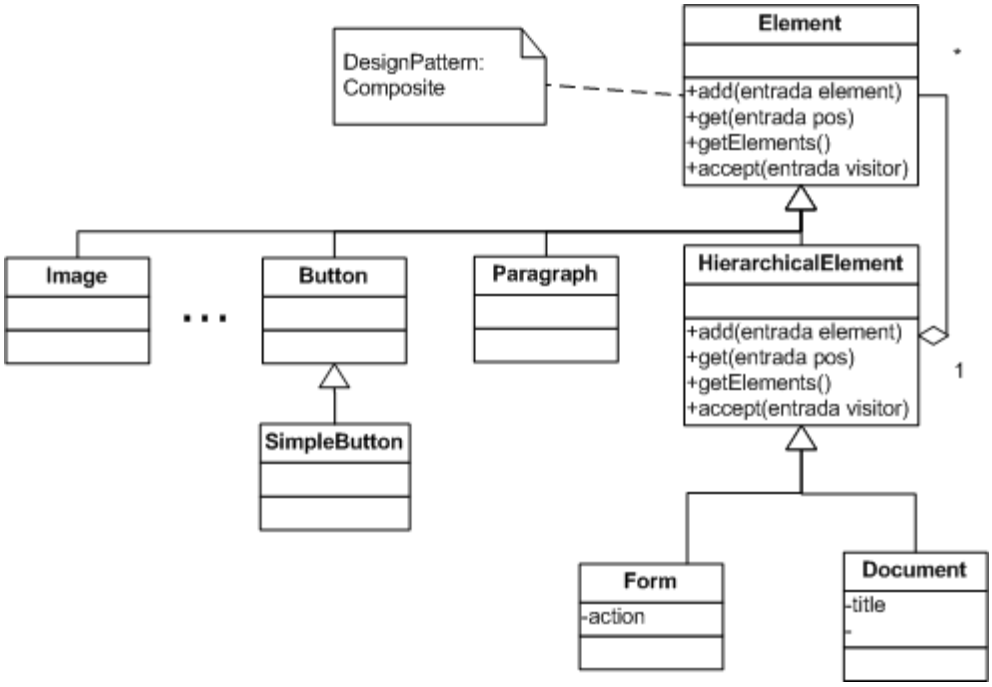
Para abstrairmos a implementação também o SFBLogger tem uma instância de SFBLoggerImp, que pode usa para escrever os logs e os erros. Para exemplo criámos uma implementação esta class para a consola e uma para ficheiro.



Composite

Quando criado a implmentação do builder gráfico para codificar a interface gráfica em HTML verificou-se a necessidade de decompor esta codificação em dois passos: criação da hierarquia em memória, tradução da hierarquia para html (ver Visitor).

Para manter a hierarquia da interface em memória criou-se várias classes do tipo Element para cada tipo de primitiva gráfica (Button, Table, etc), mas muitos destes são também eles portadores de vários elementos (hierarquia) e portanto implementou-se a pattern *Composite*.

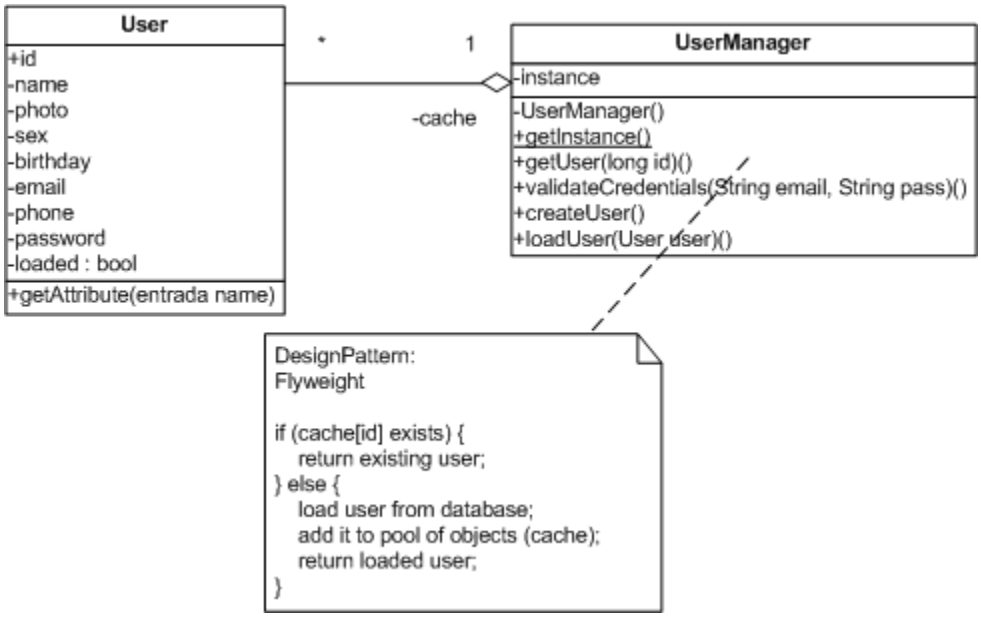


Todos os elementos são subclasse da classe Element que define os métodos presentes em elementos não hierárquicos e elementos hierárquicos. No caso do elemento ser hierárquico (composição), extende da classe HierarquicalElement que implementa os métodos deste tipo de elementos, ao contrário da classe Element que apenas declara estes métodos mas não os implementa. Desta forma, as classes que fazem uso desta hierarquia podem ignorar a diferença entre as composições de objectos e objectos individuais.

Flyweight

Os objectos utilizados no SimpleFacebook são comuns a muitas situações e portanto era importante encontrar forma de minimizar o espaço utilizado em memória. Apoiado na decisão de existir um Manager (instância única através de Singleton) para gerir os Utilizadores e Items (UserManager e ItemManager respectivamente) foi incorporada a pattern *Flywight* que permite reutilizar as mesmas instâncias de utilizadores e items evitando muitas das vezes acessos às bases de dados.

Mantendo apenas uma única instância de cada um destes objectos torna-se não só menor a carga de memória, mas também simplifica os problemas de sincronização que iriam ocorrer quando se pretende alterar valores destes objectos (passa a ser apenas necessário actualizar o objecto, e persistir as alterações para a BD).

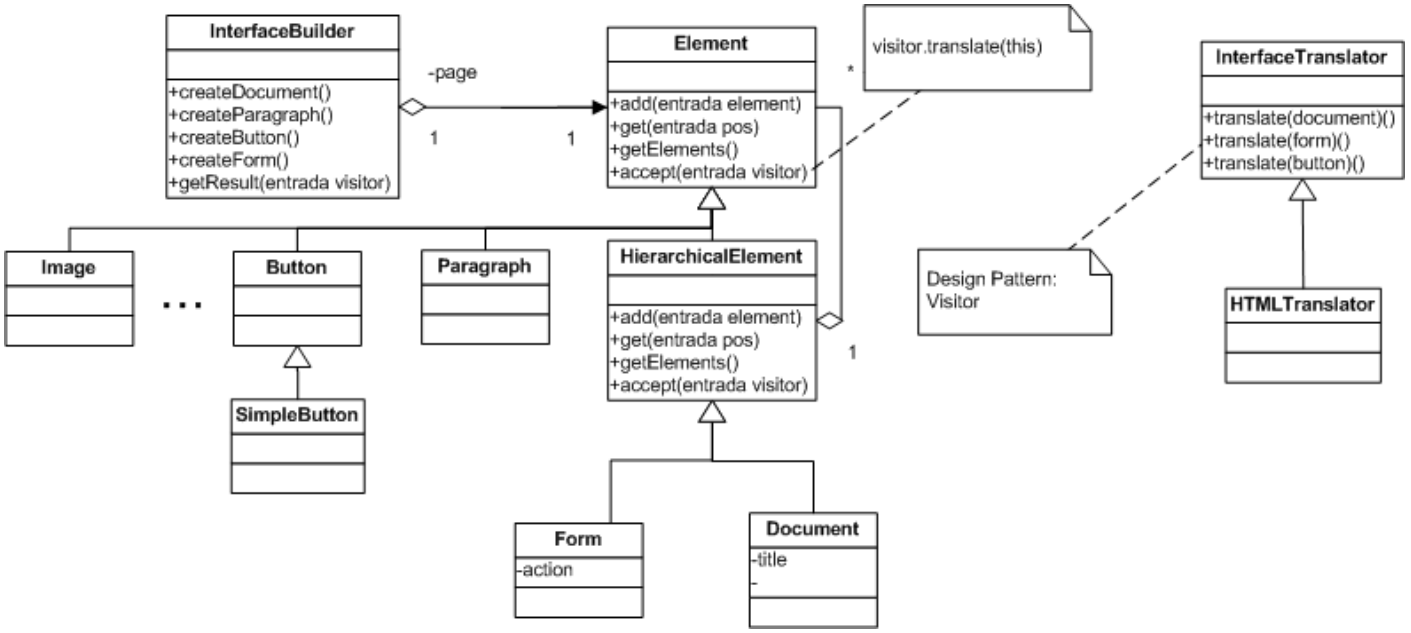


Comportamentais

Visitor

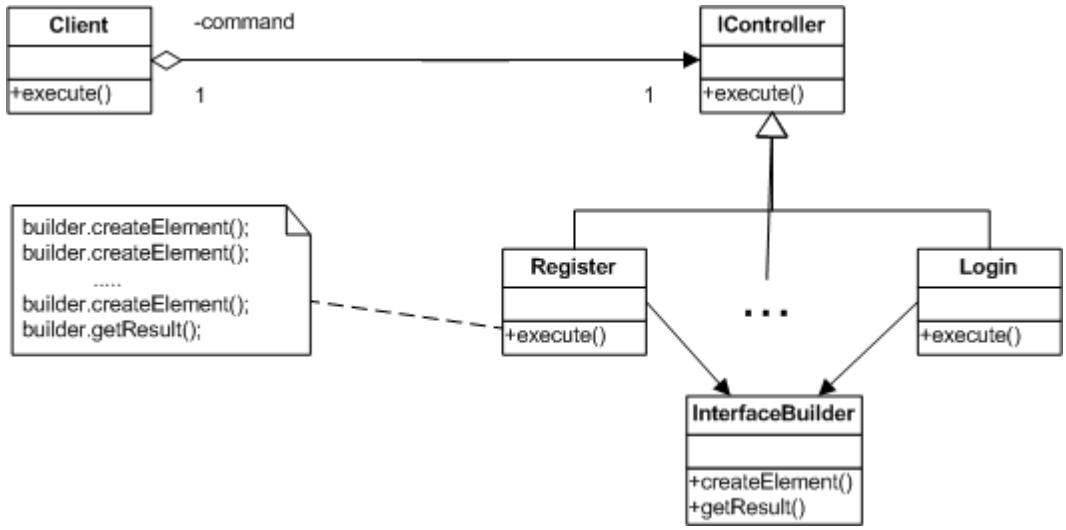
Na interface gráfica com a codificação HTML foi necessário que a construção seja feita em duas fases visto que esta codificação têm algumas exigências que só podem ser resolvidas conhecendo a hierarquia da interface antes de a codificar (o uso do formato xml em html obriga a fechar os elementos depois de colocada a codificação dos filhos no seu interior).

Para implementar estes dois passos o respectivo builder (*HTMLBuilder*) contrói a hierarquia de objectos *Element* nos seus passos intermédios, e depois é utilizado um tradutor no seu getResult para converter essa mesma hierarquia para texto html. O tradutor implementa a pattern Visitor que percorre a hierarquia visitando cada elemento (e seus filhos se for um elemento composto) e traduzindo sequencialmente para o respectivo código HTML (neste passo final já é conhecida os tipos e quantidades de filhos que cada objecto têm por exemplo).



Command

Para apoiar a lógica na camada dos controladores na arquitectura MVC era necessário encontrar uma forma de fazer corresponder a uma acção recebida pelo protocolo do cliente a uma operação abstracta (delegar a acção sem conhecer o dono do método ou os seus parâmetros). A pattern *Command* foi utilizada e uma classe criada para cada método possível (registar utilizador, fazer login, obter mural, etc). O diagrama abaixo descreve este design:

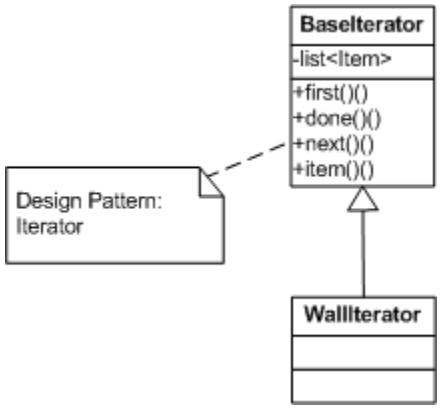


Iterator

Na construção da mural de um qualquer utilizador é necessário percorrer a lista de items de um ou vários utilizadores, por uma ordem escolhida pelo utilizador, numa quantidade não conhecida à priori. Carregar todos os items presentes na base de dados é uma impossibilidade, especialmente quando o número destes cresce muito rapidamente.

Uma forma de isolar a forma como os elementos são percorridos foi implementar a pattern *Iterator* no

topo de uma lógica mais complexa, simpficando a forma como os items são percorridos. A classe *BaseIterator* define os métodos básicos de um qualquer iterator (next, count, done) mas a classe *WallIterator* implementa o iterador especifico de percorrer os items de uma mural. Este iterator faz uso da classe *OrderStrategy* que implementa a pattern *Strategy* para definir a ordem destes items de uma forma transparente.



Strategy

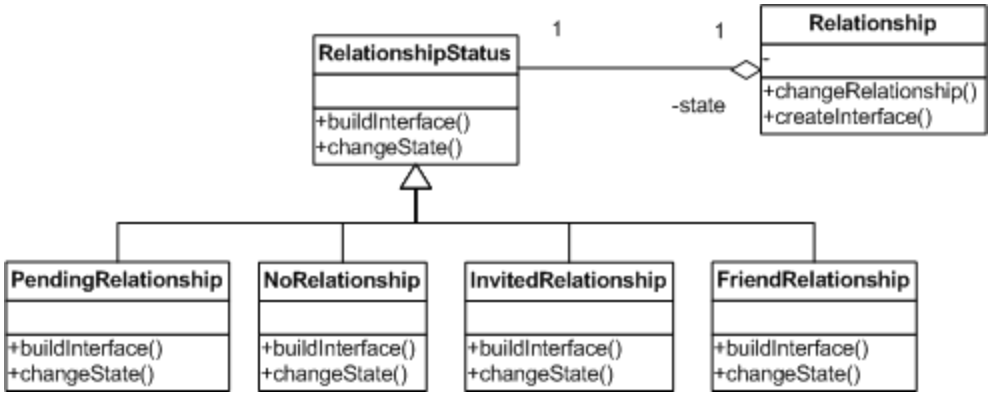
No SimpleFacebook pretendia-se criar a opção do utilizador visualizar a mural (composição de items) em várias ordens possíveis. Pretendia-se que existissem vários algoritmos de ordenação dos items na mural e permitir ao utilizador alternar entre algoritmo de uma forma transparente. Para esta tarefa foi definida uma classe OrderStrategy que define os métodos que cada uma das ordens têm de implementar e fazer a ordenação de forma transparente à ordem escolhida.

State

No SimpleFacebook cada par de utilizadores têm associada um estado da relação entre eles. Uma relação entre dois utilizadores pode estar nos seguintes estados: **Não existe** (não está presente na base de dados. Deve apresentar na interface opção para adicionar como amigo), **Fez o convite** (o utilizador já pediu, está pendente a confirmação. Deve apresentar na interface esta informação), **Foi convidado** (o outro utilizador já fez o pedido, falta a confirmação. Deve apresentar na interface opção de confirmar) e por fim **Existente** (ambos utilizadores confirmaram, deve apresentar opção de eliminar a relação).

Para descrever este estado, a pattern *State* foi utilizada e assim simplificar as operações inerentes à alteração do estado da relação. Também é útil para abstrair as diferentes implementações da construção gráfica que depende do estado.

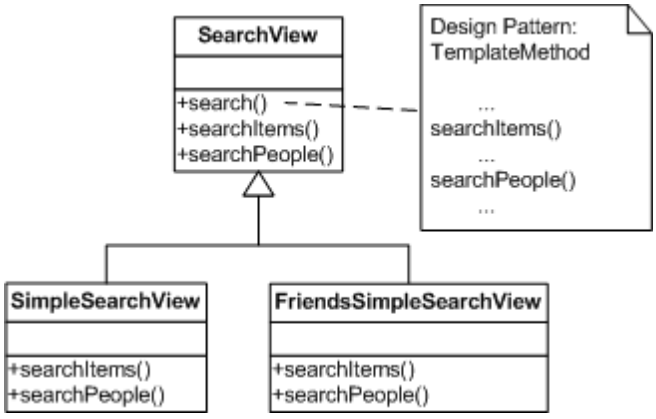
O diagrama abaixo apresenta a estrutura desta pattern na aplicação:



O gestor de utilizadores (*UserManager*) fornece o objecto da relação entre dois utilizadores (*Relationship*) que por sua vez têm associado o respectivo estado desta. Este objecto permite a qualquer momento alterar o estado desta relação.

Template

O SimpleFacebook permite ao utilizador efectuar uma pesquisa através dos vários modelos de dados da aplicação, mas precisam existir várias implementações de algumas das partes deste algoritmo de pesquisa, nomeadamente na pesquisa de pessoas pode ser feita a pesquisa em todos os utilizadores no sistema, ou apenas aqueles que são amigos do utilizador. Para ser feita esta alteração do comportamento do algoritmo foi aplicada a pattern *Template* que o diagrama abaixo apresenta:i



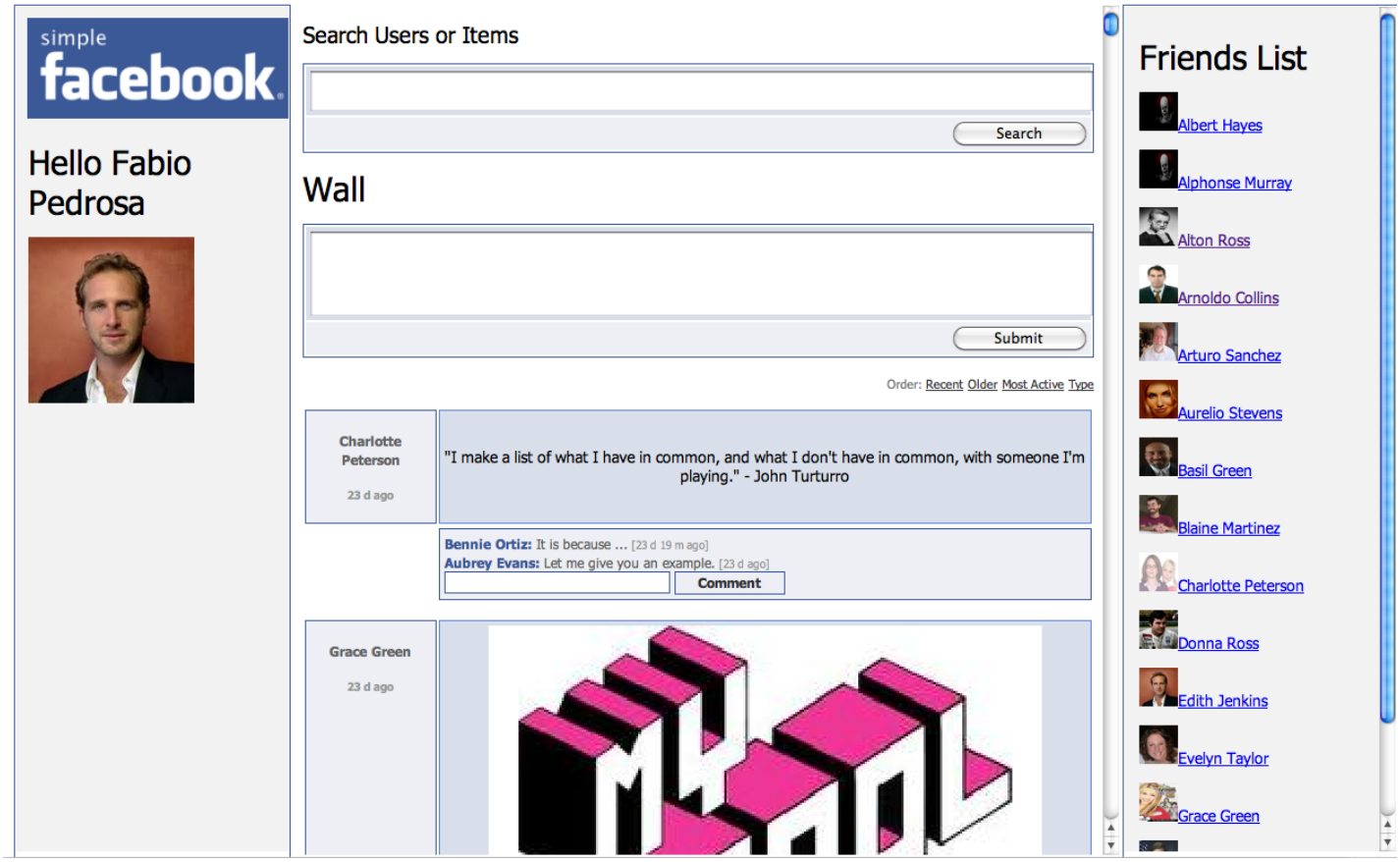
Conclusão

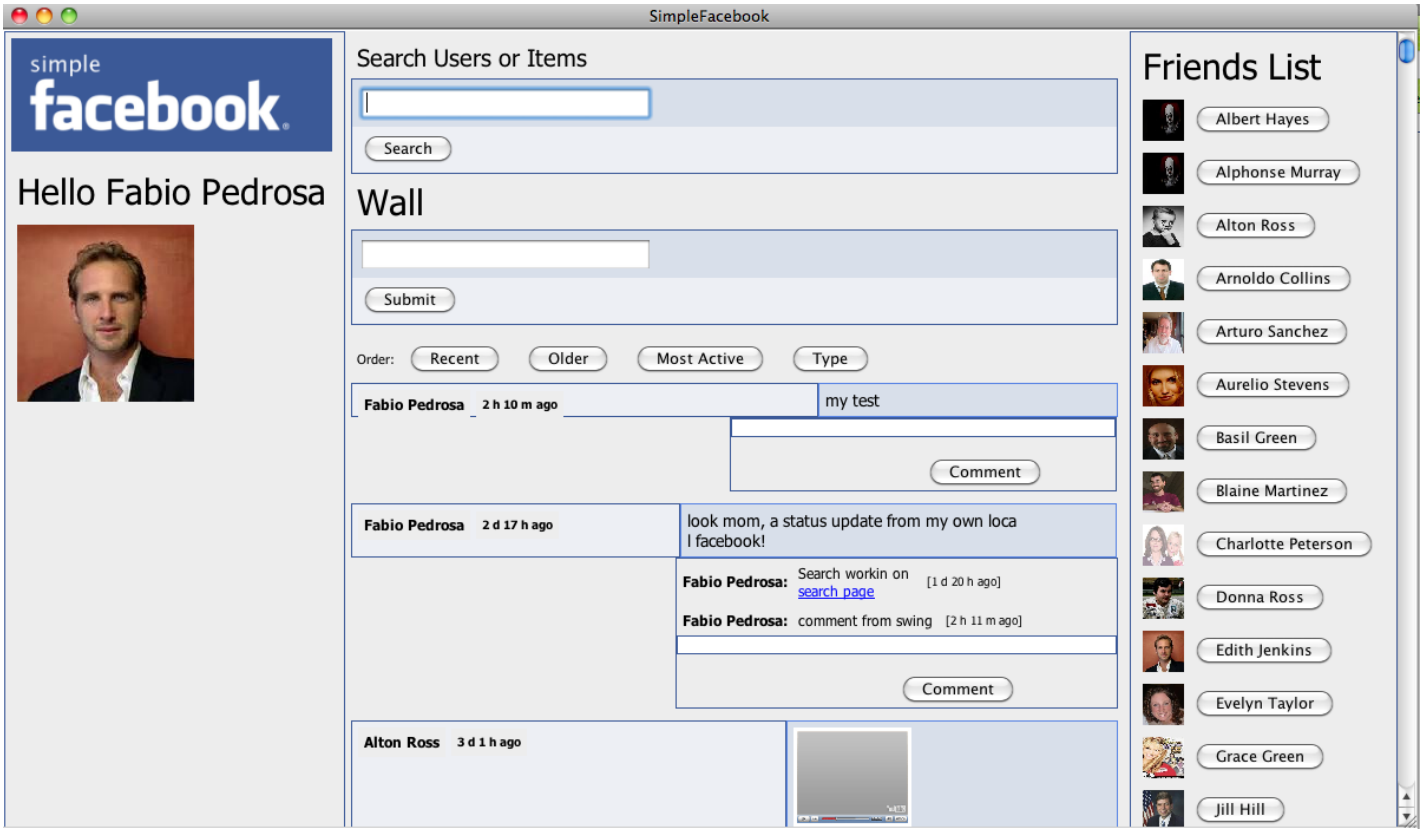
A opção de divergir do enunciado e optar por uma aplicação diferente tornou-se um desafio interessante porque as implicações que a funcionalidade gráfica têm nas necessidades arquitectuais são reais e visíveis na forma como a aplicação ficou estruturada.

A um nível mais elevado de abstracção a aplicação de patterns de arquitectura como o MVC ofereceu uma construção de camadas isoladas e como consequência um servidor mais flexível e robusto, enquanto que a um nível mais baixo, a aplicação de patterns de design ofereceram a melhor forma para tornar as funcionalidades extensíveis, as classes reutilizáveis e as interacções simples.

No final ficou demostrada a capacidade de criar uma interface gráfica agnóstica á plataforma de visualização, a capacidade de manipular milhares de items em base de dados de forma pouco intensiva, a capacidade de simplificar chamadas externas a bibliotecas num único ponto de acesso, a capacidade de alternar algoritmos da lógica do programa de forma transparente, entre outras interessantes e funcionais capacidades.

As imagens abaixo mostram as duas diferentes formas de visualizar o SimpleFacebook descritas com a mesma linguagem:




SimpleFacebook
simple facebook
Hello Fabio Pedrosa
Search Users or Items
Search
Wall
Submit
Order:
Recent
Older
Most Active
Type
Fabio Pedrosa 2 h 10 m ago
my test
Comment
Fabio Pedrosa 2 d 17 h ago
look mom, a status update from my own loca
l facebook!
Fabio Pedrosa: Search workin on search page [1 d 20 h ago]
Fabio Pedrosa: comment from swing [2 h 11 m ago]
Comment
Alton Ross 3 d 1 h ago
Friends List
Albert Hayes
Alphonse Murray
Alton Ross
Arnoldo Collins
Arturo Sanchez
Aurelio Stevens
Basil Green
Blaine Martinez
Charlotte Peterson
Donna Ross
Edith Jenkins
Evelyn Taylor
Grace Green
Jill Hill