

*Instituto Superior de Engenharia de Lisboa*

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Programação em Dispositivos Móveis

Relatório do 3º Trabalho

Docente: Pedro Pereira   
Discentes: Ana Correia e Diogo Cardoso

Junho 2012

Conteúdo

[Introdução 3](#_Toc327188401)

[Notificação assíncrona 3](#_Toc327188402)

[Activities 3](#_Toc327188403)

[Comunicação entre activities 3](#_Toc327188404)

[Inicio da aplicação 3](#_Toc327188405)

[Controlo 3](#_Toc327188406)

[Activity Endpoint 3](#_Toc327188407)

[Controllers 3](#_Toc327188408)

[Bounded clients 3](#_Toc327188409)

[Controlo concorrente 3](#_Toc327188410)

[Serviços 3](#_Toc327188411)

[Bounded Services 3](#_Toc327188412)

[Content provider 3](#_Toc327188413)

[Started services 3](#_Toc327188414)

[Conclusão 5](#_Toc327188415)

# Introdução

Este relatório aborda as soluções adotadas na realização do terceiro trabalho da unidade curricular *Programação em Dispositivos Móveis.* O objetivo do trabalho era a finalização da aplicação iniciada nos outros trabalhos, sem qualquer restrição de funcionalidades da infraestrutura.

Nesta fase grande parte do módulo de negócio foi alterado para utilizar os mecanismos da plataforma *android*, nomeadamente ao que diz respeito a comunicação entre serviços e *activities*.

A aplicação desenvolvida está dividida em três módulos singulares, os mesmos que o padrão *Model View Controller* (*MVC*), este documento encontra-se dividido de forma a explicar as opções tomadas em cada um dos módulos.

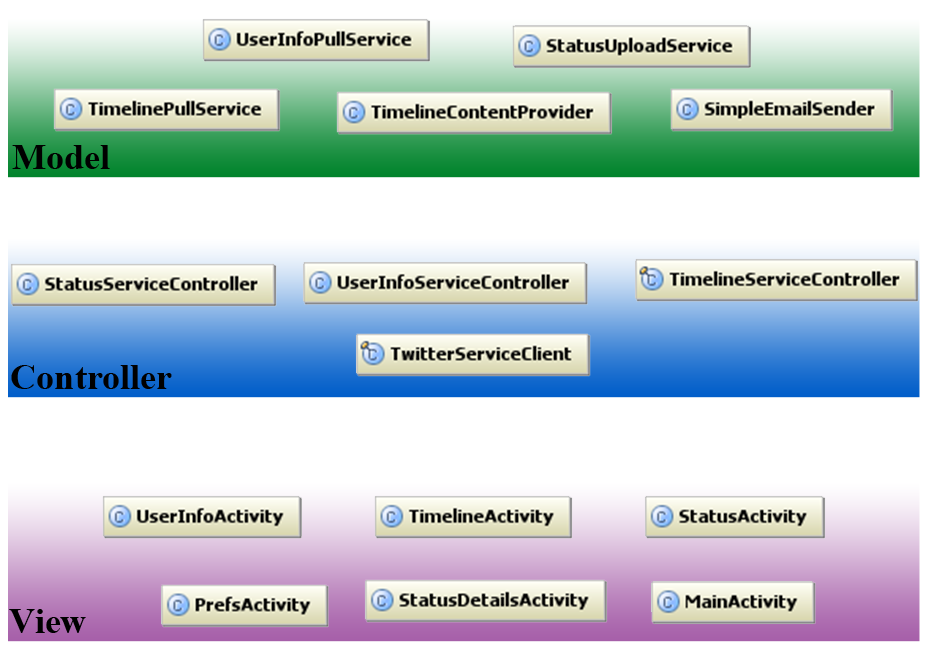


Figura - Arquitetura Geral da aplicação.

## 

## Notificação assíncrona

A Figura 2 ilustra as interfaces base do mecanismo de notificação assíncrona criado.

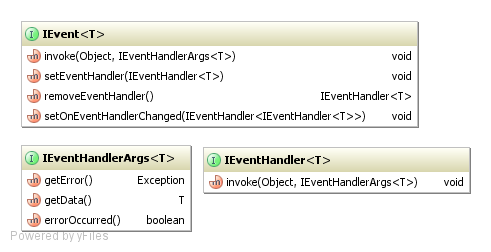


Figura - Diagrama de classes do esquema de notificação assíncrona.

A interface *IEvent* define uma "promessa" de um resultado futuro, para ser possível obter esse valor é necessário registar num objeto do tipo *IEvent* um *handler*, este irá ser invocado quando essa "promessa" for cumprida, ou seja, quando a operação assíncrona produzir um resultado. A interface *IEventHandlerArgs* serve apenas para que exista alguma forma de obter a exceção em caso de erro.

A motivação para a criação deste mecanismo foi que para cada operação assíncrona que exista no sistema, exista um evento associado a esta, evento que deverá ser chamado quando a operação for concluída. Um exemplo da utilização deste mecanismo está representado na Listagem 1.

class SomeClass

{

public IEvent<SomeReturnObject> someOperationCompleted;

public void someOperationAsync()

{

(...)

}

}

Listagem - Exemplo de utilização do mecanismo de notificação assincrona.

# Activities

As activities representam a componente visual da aplicação, foi tomado especial cuidado em não colocar qualquer lógica aplicacional dentro de activities sendo as suas responsabilidades as seguintes:

* Controlar os elementos visuais da aplicação.
* Iniciar outras activities por ordem do utilizador.
* Utilizar o mecanismo de notificação assíncrona descrito anteriormente para realizar operações de negócio.

Na Figura 3 é possível visualmente identificar as activities que realizam operações assíncronas devido à sua implementação explicita da interface *IEventHandler*.

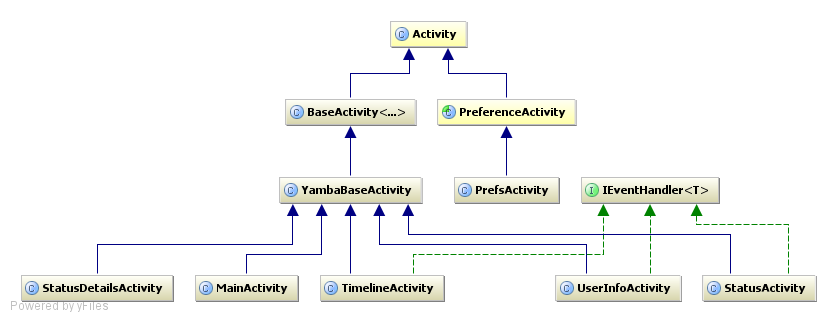


Figura - Diagrama de classes das activities.

Para generalizar e evitar repetição de código foram criadas duas classes base:

* BaseActivity<T extends Application> - tem como principal objetivo aglomerar informação e rotinas que sejam de comum utilização pelas *activities* mas não dependam da aplicação em si. Neste caso contém o acesso tipificado à instância de *Application* e à instancia de *NavigationMessenger*.
* YambaBaseActivity - tem o mesmo objetivo que a classe *BaseActivity* com a diferença de aglomerar membros que sejam dependentes da aplicação como por exemplo o menu.

## Comunicação entre activities

A plataforma *android* tem alguns mecanismos para facilitar a comunicação entre activities, nomeadamente a possibilidade de serialização de instâncias de tipos primitivos. A serialização de um tipo complexo não é aconselhada neste tipo de plataformas pelas suas características físicas, portanto desenhou-se uma solução que consiste na utilização de um contentor de objetos referenciados por uma chave, chave essa que será um tipo primitivo. O facto de a chave ser um tipo primitivo permite a utilização dos mecanismos presentes no *android* para "parametrizar" uma *activity* com informação, nomeadamente o método *putExtra* da classe *Intent*.

Foi necessário então de criar e publicar o contentor num local único e acessível a todas as *activities*. Uma instancia estática do contentor foi criada na classe *BaseActivity* e dada permissão de acesso a todas as *activities* derivadas. A classe NavigationMessenger representa este contentor e o seu diagrama encontra-se na Figura 4.

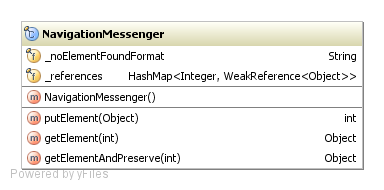


Figura - Diagrama da classe NavigationMessenger.

Na implementação da classe foi tido em conta o tempo de vida dos objetos, já que se uma instancia estática desta classe tiver qualquer referencia para um objeto este "só" irá ser "colhido" pelo *garbage* *collector* quando a aplicação terminar. Por isso a estrutura de dados utilizada uma *WeakReference*, permitindo a plataforma "colher" o objeto quando não existir mais referencias para este.

A justificação da utilização deste contentor aos outros mecanismos que a plataforma android contém para passar objetos complexos entre activities deve-se a questões de eficiencia e de aplicabilidade neste projeto. Uma solução poderia ser utilizar classes que implementam *Parcelable*, criadas para suportar a passagem de objetos complexos entre activities (nomeadamente entre processos). Nesta aplicação todas as activities se executam dentro do mesmo processo sendo o *overhead* de serialização e deserialização uma desvantagem para a performance da aplicação. Outra razão é o incomodo de criar classes *Parcelable*, além das regras não contratuais necessárias, a implementação da interface *Parcelable* "suja" a interface publica dos objetos de domínio passados pelas activities.

## Inicio da aplicação

- escolher preferences ou inicio assincrono

# Controlo

## Activity Endpoint

- responsabilidades.

## Controllers

## Bounded clients

## Controlo concorrente

# Serviços

overall

## Bounded Services

## Content provider

## Started services

# Conclusão