Instituto Superior de Engenharia de Lisboa

Sistemas Distribuídos

# Relatório de Trabalho

## Brokering System (Série 1)

*Autoria:*
33724 David Raposo
33404 Ricardo Mata
32632 Pedro Pedroso

*Para:*
Engº Luís Assunção

21 de Maio de 2014

# Conteúdo

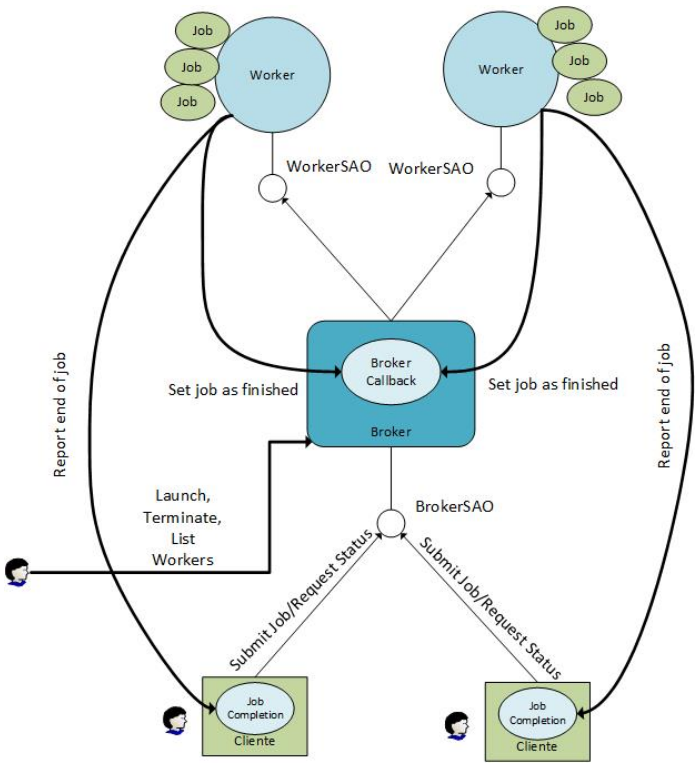
# 1 Introdução

O primeiro trabalho desta cadeira pede que desenvolvamos um sistema de brokering entre clientes e trabalhadores, sem que estes se conheçam. Serve o presente relatório para explicar a nossa implementação e para discutir as nossas soluções.

# 2 Arquitetura

A solução tem 3 intervenientes distintos. Um cliente que pretende que sejam realizadas tarefas remotamente, um (ou mais) serviço de execução de trabalhos que irá executar trabalhos submetidos pelo cliente, e um intermediário que irá delegar trabalhos aos serviços de execução. A partir deste ponto passaremos a referirmo-nos ao intermediário como *broker* e aos serviços de execução como *workers* ou *worker*.

O cliente pretende que alguém lhe faça um trabalho. Para isso envia um pedido ao *broker* a dizer qual o trabalho que pretende que seja executado. O *broker* decide então qual o *worker* mais adequado para executar essa tarefa, e envia-lhe o pedido do cliente. Assim que seja possível o worker inicia a execução do trabalho, e assim que o trabalho estiver concluído, notifica o cliente e o broker de que o trabalho foi concluído.



# 3 Estruturas

## 3.1 Armazenamento de dados

**DataManager:**
O data manager é a class que é usada para armazenamento e manipulação de dados relativos ao broker.

**JobWrapper:**

*Class* de armazenamento de informação relativa ao *Job*. Contém:

**j:** Objecto job com informação para iniciar o processo (id do job,nome do processo, ficheiro de input, ficheiro de output e *proxy* para a chamada final ao cliente para sinalizar o fim do trabalho).

**status:** *Status* que o job tem (*Queued*, *Running*,*Finished*).

**WorkerWrapper:**

*Class* de armazenamento de informação relativa ao *worker*. Contém:

**workerProxy:** *Proxy* para comunicação com o *worker*.

**currentJobs:** Número de trabalhos que o *worker* está a processar de momento.

**dictJobs:** Dicionário com todos os *jobs* que o worker está a processar de momento.

**port:** *Port* associado ao *worker*.

Esta informação é guardada em duas estruturas

***jobDict*:** Dicionário onde a chave é o *id* e o *value* é um objeto *JobWrapper*. Esta estrutura é usada para guardar o progresso dos *jobs* para eventual consulta pelo cliente.

***workerDict*:** Dicionário onde a chave é o *port* do *worker* e a chave é um *WorkerWrapper*. É usado para associar trabalhos aos respetivos *workers*. Em caso de um *worker* falhar, podemos ressubmeter os trabalhos para outro *worker*.

## 3.2 *Interfaces*

**IWorkerSAO:** Esta interface disponibiliza os serviços do *worker* para o *broker*: Submissão de *Jobs*, fecho do próprio *worker*, *ping* e requisição do número de *Jobs* a processar de momento. Visto o *worker* abstrair os clientes e o *broker* da execução de trabalhos, estes desconhecem o tempo de execução dos trabalhos, de forma a que o *worker* pode ficar bastante tempo a processar. Os acessos aos objetos dos *workers* também vão ser feitos por uma só entidade, o *broker*, o que nos levou a implementar a interface como *Singlecall* em vez de *Singleton*, pois só vai existir uma instância por *worker* com tempo de vida definido pelo *broker*.

**IBrokerSAO:** Utilizado para os clientes fazerem submissão de *Jobs* ou para verificar o seu estado. Este objeto disponibiliza serviços do *broker* e é acedido por cada cliente que queira fazer um *Job*. Como o *broker* é uma entidade central que pretende delegar os trabalhos aos *workers*, faz sentido que estado das suas estruturas de dados seja visível aos clientes que lhe acedam, por isso implementámos o *broker* como *singleton*.

# 4 Implementação

Configurámos cada uma das partes para se ligarem a uma porta *TCP* como um *Well Known Type*. Isto permite que cada uma das partes consiga comunicar entre si. Esta configuração define os *end-points* onde cada parte irá aceder para poder comunicar com outras partes, e o tipo de canal que é usado para a comunicação.

O cliente regista o *Well Known Type* do *broker* à interface partilhada. Isto permite criar uma interface de comunicação onde cada acesso feito pelo cliente vai ser tratado não localmente, mas pelo *broker*. Um pedido de trabalho é então enviado ao *broker* através dessa interface, e para o pedido de trabalho é criada uma instância do tipo *Job*. Este tipo tem a seguinte informação:

- O nome do trabalho que se quer executar.
- Os nomes dos ficheiros input e output.
- O identificador do trabalho que é dado pelo *broker*, para pedidos de estado.
- Uma interface para que o *worker* possa avisar diretamente o cliente da conclusão do trabalho.

A existência da interface de comunicação para o worker torna este tipo num *proxy*. Isto requer um cuidado extra na configuração dos canais de comunicação (o canal tem de ser configurado como "'*full*"' para que a comunicação seja delegada pelo *broker*). É também de notar que esta solução foi implementada tendo em conta que os *workers* vão estar na mesma máquina que o broker e o cliente (apesar de se simular um ambiente remoto), de forma a que uma solução completamente remota iria necessitar de outra abordagem na passagem de parâmetros/receção de resultados entre o cliente e os *workers*.

Quando o objeto chega ao broker é colocado num mapa de jobs. Este mapa contém informação sobre os *jobs* submetidos para que seja possível ressubmetê-los caso haja necessidade de tal.

## 4.1 Modo Automático

### 4.1.1 Adição de workers

A adição de *workers* em modo automático segue um só critério. Se houver demasiados *Jobs* num *worker*(definido pela variável *NUMBER_OF_MAX_SLOTS_FOR_WORKER*) no momento da submissão de um novo *Job*, o *broker* irá criar um novo *worker*.

### 4.1.2 Remoção de workers

A remoção automática é feita segundo o seguinte critério: se houver mais do que um *worker* sem trabalhos, então um deles será removido. Esta verificação acontece sempre que um *Job* seja retirado do *worker*.

## 4.2 Modo Manual

### 4.2.1 Adição de workers

A adição de *workers* é feita simplesmente ao adicionar mais um *worker* ao dicionário de *workers* disponíveis. Durante a atribuição de *Jobs* é escolhido o

*worker* com menos trabalhos de momento.

### 4.2.2 Remoção de workers

A remoção de *workers* é feita ao remover um dos *workers* disponiveis do dicionário e caso ainda haja *Jobs* nos *workers*, estes serão resubmetidos a outros *workers*. Esta remoção é feita utilizando a porta do *worker* como identificação.

### 4.2.3 Listagem de *workers*

É possível listar os *workers* a trabalhar de momento, visualizando a porta onde estão configurados.

## 4.3 Configuração dos objectos remotos

A configuração dos objetos remotos(*BrokerSAO* e *WorkerSAO*) é realizada através de ficheiros de configuração. O ficheiro de configuração do *BrokerSAO* contém o tipo de ligação, a porta da ligação, o nome do *well-known object* e a porta base usada para a criação dos ficheiros de configuração dos *workers*. Os ficheiros de configuração do *worker* são criados pelo *broker* localmente, que envia o caminho relativo a cada *worker* para este utilizar na sua configuração. A porta a utilizar por cada *worker* é escolhida incrementando a porta base que está presente no ficheiro de configuração do *broker*.

**Limitação:** Estamos a admitir que os *workers* estão a trabalhar na mesma máquina, assim podemos passar só o caminho dos ficheiros de configuração para os *workers*. No caso dos *workers* estarem a trabalhar em máquinas diferentes, seria necessário passar os parãmetros e seria o próprio *worker* a fazer a sua configuração localmente.

## 4.4 Estado do *Job*

A nossa aplicação atualiza e guarda o estado de cada *Job* ao longo do seu percurso. Estes estados refletem o estado do *job*: quando o *Job* é submetido tem o estado *QUEUED*, quando é enviado para o *worker* tem o estado *RUNNING*, e finalmente quando acaba tem o estado *FINISHED*. Estas atualizações são feitas pelo *broker* sendo a ultima feita pelo *worker* que, ao terminar o trabalho, notifica através do *proxy BrokerCallback* qual o *Job* que acabou. O cliente pode a qualquer momento fazer uma verificação do estado do *Job* através da interface partilhada *IBrokerSAO*.

**Limitação:** Um cliente pode saber o estado de um Job que não tenha sido ele próprio a criar. Prevenir esta situação implicava verificações adicionais e identificação de cada cliente que iria criar mais *stress* sobre o *broker* e isto não era o objetivo deste trabalho.

## 4.5 Suporte e deteção de falhas

A deteção de falhas de um *worker* ocorre quando há uma tentativa de ligação através do *WorkerSAO* e ocorre uma *socket exception*. Estas deteções são feitas na atribuição de um trabalho ou na verificação de estado do trabalho. Na

verificação, como temos os estados do trabalho guardados no *broker* não necessitando portanto fazer uma ligação com o *worker*, fazemos um *ping* ao *worker* associado para verificar a ligação.

No caso de um *worker* falhar, os trabalhos naquele momento a ser processados vão ser perdidos. Detetamos esta situação quando uma tentativa de acesso a um *proxy* resulta numa *Socket Exception*. Nesta situação, o *broker* vai procurar em *workerDict* os *Jobs* associados a este *worker* e vai ressubmetê-los a outros *workers* que estejam activos.