Arquitetura de Software

Trabalho Prático 2

Bruno Madureira, Fábio Moura e Nuno Miranda
Departamento de Engenharia Informática
Universidade de Coimbra

I. INTRODUÇÃO

Neste trabalho foi modificada uma arquitetura de um *software* já existente. No estado em que estava o *software*, apenas era possível haver comunicação entre o servidor de base de dados e os clientes através de uma rede local. As modificações implementadas permitem que seja possível aceder à base de dados a partir de outros locais afastados das instalações principais da EPE.

II. DRIVERS ARQUITETURAIS

Os *drivers* arquiteturais são requisitos que moldam a arquitetura. Estes são constituídos por requisitos funcionais, restrições e atributos de qualidade.

A. REQUISITOS FUNCIONAIS

Os requisitos funcionais irão descrever o que o sistema deve fazer, como se irá comportar e reagir aos estímulos durante a sua execução. A seguir iremos apresentar a tabela de identificação dos casos de uso:

Nome da entidade: Exton Plantas exóticas	**ID da entidade:** EPE
Descrição: – A Exton Plantas exóticas é uma empresa que fornece plantas exóticas para estufas, museus, governos laboratórios de investigação e jardins públicos. Foram desenvolvidas duas bases de dados para suportar as aplicações para este sistema: uma que guarda todos os dados dos produtos em inventário e outra com as informações de todas as encomendas.	
Pressupostos fornecidos: – A entidade providência todas as ações que iram ser efetuadas no sistema. Por exemplo, quando é adicionada uma nova encomenda na aplicação a entidade irá tratar de ir buscar o produto e entregar ao cliente. Ou seja, todas as ações físicas são efetuadas pela entidade.	
Pressupostos exigidos: – A entidade requer que o sistema permita gerir as bases de dados de forma eficiente e segura (adicionar, consultar e modificar dados nas bases de dados com autenticação).	

Casos de uso identificados:

– **OrderApp: “**Adicionar nova encomenda”, “Consultar Inventário”, “Aceder a informações de log”, “login” e “logout”.
– **ShippingApp:** “Mostrar encomendas despachadas”, “Verificar encomendas pendentes”, “Selecionar encomenda” e “Marcar encomenda como despachada”.
– **InventoryApp:** “Listar inventário”, “Selecionar tipo de produto” e “Adicionar item a inventário”.

A. 1 - ORDERAPP

Na imagem seguinte apresentamos o diagrama de casos de uso da aplicação modificada “OrderApp” assim como as tabelas dos respetivos.

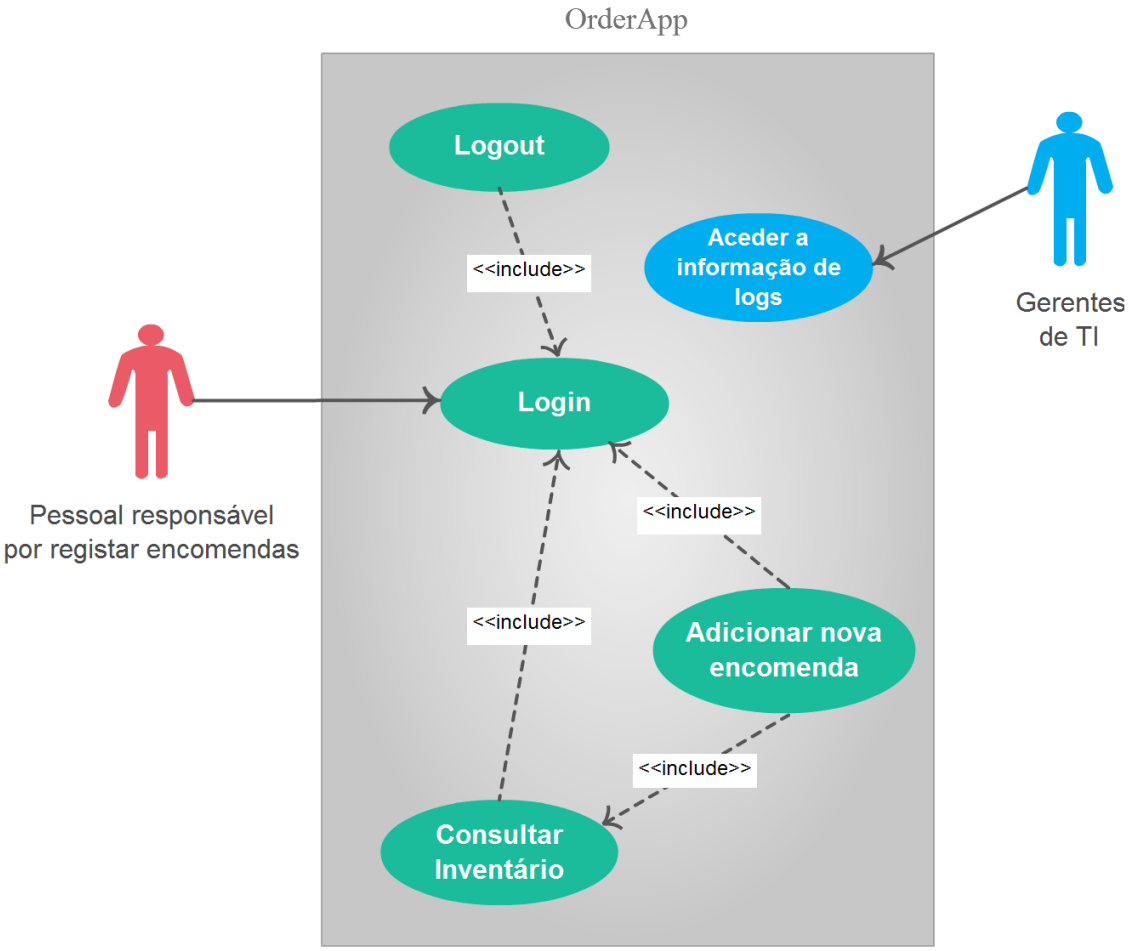

Figura 1 - Casos de uso da OrderApp

Login

Atores Primários	Pessoal responsável pelas encomendas
Atores Secundários	
Descrição	Autenticação no sistema
Trigger	
Pré-condição	Ligação de rede ao servidor
Pós-condição	Utilizador autenticado

Adicionar nova encomenda

Atores Primários	Pessoal responsável pelas encomendas
Atores Secundários	
Descrição	Este caso de uso serve para adicionar uma nova encomenda na base de dados “orderinfo”.
Trigger	Cliente pretende efetuar uma encomenda.
Pré-condição	O Utilizador tem de estar autenticado dentro do sistema.
Pós-condição	Encomenda efetuada.

Aceder a informação de logs

Atores Primários	Gerentes de TI
Atores Secundários	
Descrição	Este caso de uso serve para aceder ao histórico de todas as operações efetuadas dentro do sistema "OrderApp”.
Trigger	
Pré-condição	O gerente de TI tem de estar autenticado como tal.
Pós-condição	Gerente de TI tem conhecimento das atividades dos utilizadores.

Logout

Atores Primários	Pessoal responsável pelas encomendas
Atores Secundários	
Descrição	Termina sessão no sistema.
Trigger	
Pré-condição	Sessão autenticada.
Pós-condição	Sessão encerrada.

A. 2 – SHIPPING APP

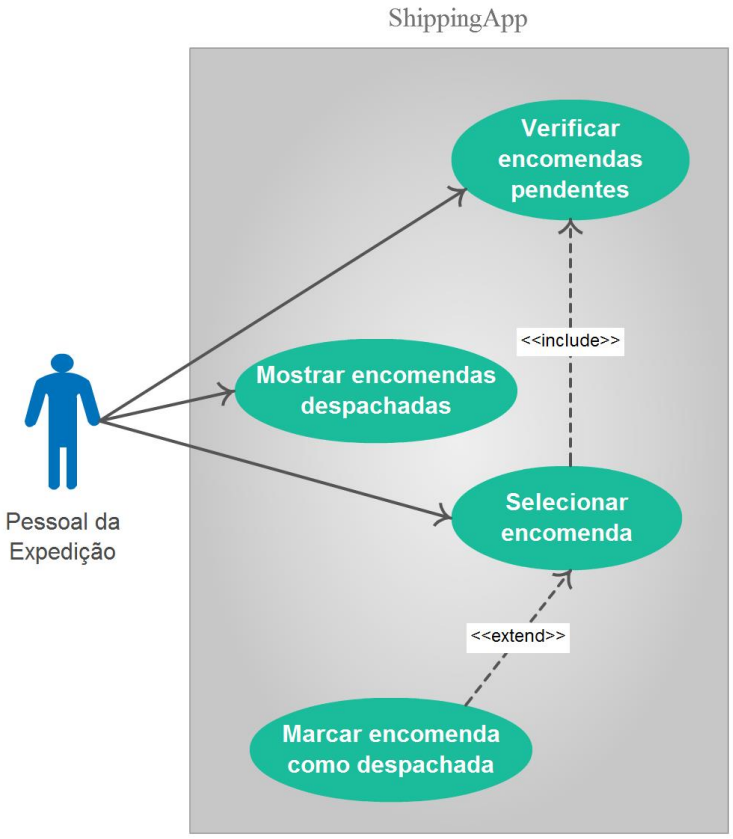

Figura 2 - Casos de uso da ShippingApp

Selecionar encomenda

Atores Primários	Pessoal da expedição
Atores Secundários	
Descrição	Pessoal da expedição seleciona uma encomenda pendente no sistema
Trigger	Necessidade de verificar os detalhes de determinada encomenda
Pré-condição	Encomenda pendente presente no sistema
Pós-condição	Encomenda selecionada com sucesso

Verificar encomendas pendentes

Atores Primários	Pessoal da expedição
Atores Secundários	
Descrição	Pessoal da expedição verifica as encomendas pendentes presentes no sistema
Trigger	Necessidade de verificar as encomendas que se encontram pendentes
Pré-condição	Ligação ao servidor
Pós-condição	Encomendas pendentes exibidas ao utilizador

Marcar encomenda como despachada

Atores Primários	Pessoal da expedição
Atores Secundários	
Descrição	Pessoal da expedição marca uma encomenda como despachada
Trigger	Encomenda embalada e enviada para o cliente
Pré-condição	Ligação ao servidor
Pós-condição	Encomenda marcada no sistema como despachada

Mostrar encomendas despachadas

Atores Primários	Pessoal da expedição
Atores Secundários	
Descrição	Pessoal da expedição verifica as encomendas despachadas
Trigger	Necessidade de verificar as encomendas despachadas
Pré-condição	Ligação ao servidor
Pós-condição	Encomendas despachadas exibidas ao utilizador

A. 3 – INVENTORY APP

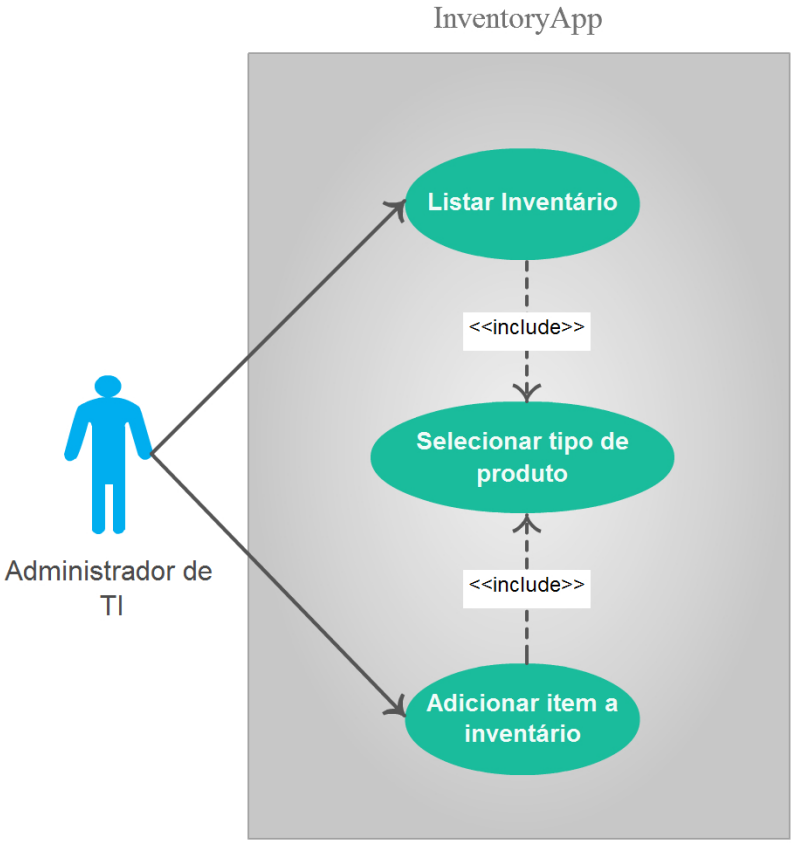

Figura 3 - Casos de uso da InventoryApp

Listar inventário

Atores Primários	Administrador de TI
Atores Secundários	
Descrição	Administrador de TI lista um inventário dos artigos presentes sistema
Trigger	Necessidade de verificar se artigo se encontra em *stock*
Pré-condição	Ligação ao servidor
Pós-condição	Inventário exibido ao utilizador

Adicionar item a inventário

Atores Primários	Administrador de TI
Atores Secundários	
Descrição	Administrador de TI adiciona um novo item ao inventário
Trigger	Necessidade de adicionar um novo item ao inventário
Pré-condição	Item não presente na base de dados
Pós-condição	Item adicionado ao inventário com sucesso

B. RESTRIÇÕES

Uma restrição é uma decisão de *design* com zero graus de liberdade, ou seja, uma decisão pré-existente. Neste caso as nossas restrições são a motivação de alteração do sistema.

Restrições de negócio

- A criação das encomendas tem de ser efetuada fora do *site* central (instalações centrais, estufas, laboratórios e armazéns).
- Comercialização apenas e exclusivamente de árvores, sementes e arbustos pois a arquitetura desenhada apenas se encontra preparada para os produtos referidos anteriormente

Restrições técnicas

- O sistema implementado requer que o java esteja instalado em todos os computadores da entidade. Caso contrário o sistema não irá funcionar (tanto a nível de interface visual como a nível de funcionalidades de *software* e rede.

C. ATRIBUTOS DE QUALIDADE

Os atributos de qualidade são características que nós pretendemos que o sistema obedeça para garantir alguns padrões de qualidade no sistema. Foram estabelecidos três tipos de atributos que iremos descrever a seguir.

– **Segurança**

Para garantir a segurança no sistema irá ser limitado o acesso remoto aos *logs* a um administrador devidamente autenticado. Para fazer qualquer encomenda será também necessário efetuar *login* por parte do pessoal responsável pelas encomendas, caso contrário não irá ser autorizada a introdução de novas encomendas assim como a visualização das encomendas atualmente efetuadas.

– **Performance**

Pretende-se que todas as operações efetuadas na aplicação "OrderApp" (aplicação a modificar) sejam realizadas em tempo exequível. O objetivo é que a performance, no mínimo, não se altere face ao sistema anterior.

– **Disponibilidade**

Outro atributo de extrema importância é a disponibilidade do sistema. Para garantir que os empregados da entidade consigam fazer todas as operações necessárias ao seu trabalho, evitando atrasos nas encomendas dos clientes ou outros problemas, é necessário garantir que a disponibilidade do sistema é máxima (dentro dos possíveis).

III. SISTEMA MODIFICADO

Para o sistema modificado foi usado o modelo MVC (*Model-View-Controller*). Este é um modelo de arquitetura de *software* que separa a representação da informação da interação do utilizador com o mesmo. Para que estes três componentes funcionassem interligados usamos o RMI (*Remote Method Invocation*) que é uma interface de programação que permite que sejam efetuadas chamadas remotas a um dado servidor. Assim existem três componentes interconectados:

- O modelo (*model*) que irá ter os dados da aplicação – neste caso a base de dados das encomendas (servidor RMI) – assim como as respetivas funções para o programa funcionar;
- A visão (*view*) que irá apresentar os dados ao utilizador – neste caso a interface java que existe na aplicação cliente;
- O controlador (*controller*) é o componente que gere a comunicação entre os dois modelos anteriores – no nosso caso o RMI.

A. VISTA COMPONENTE & CONECTOR

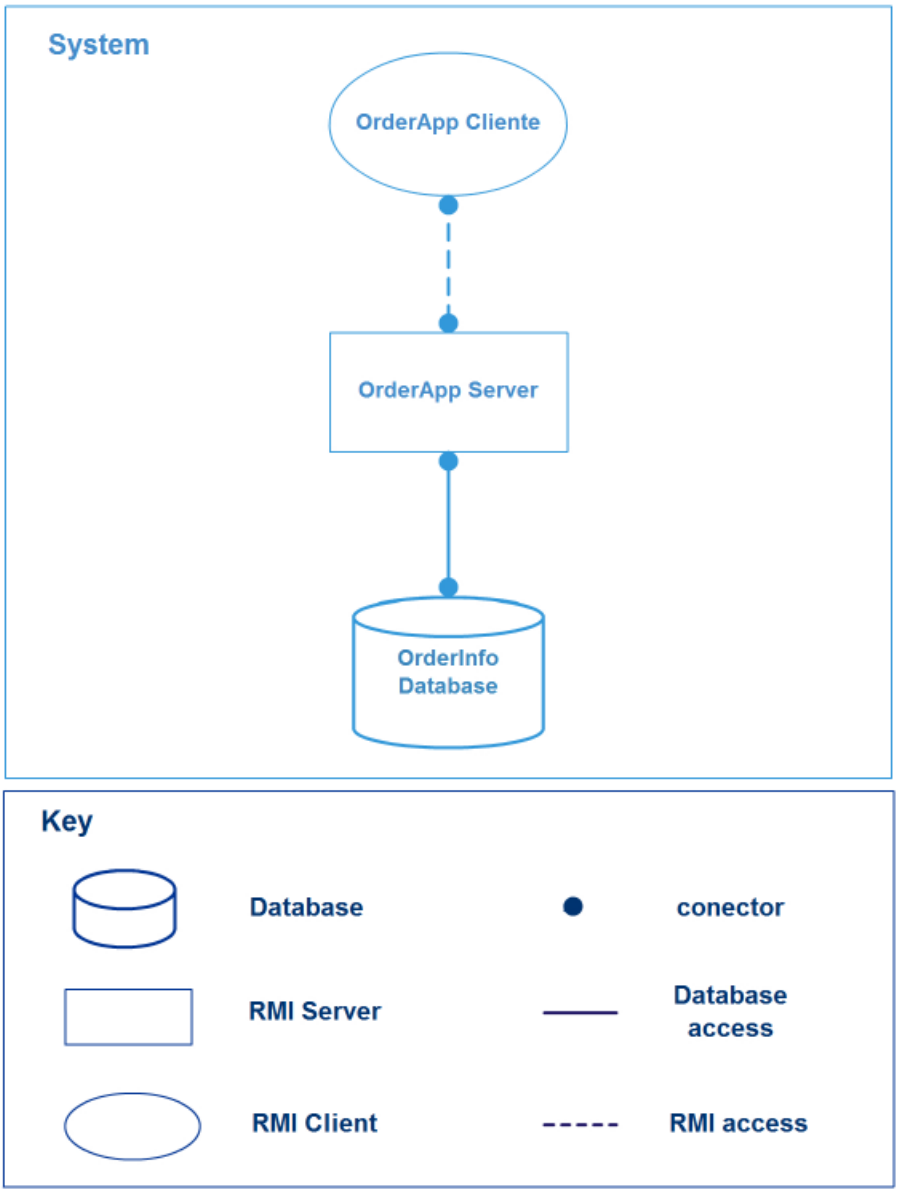

Figura 4 - Vista Componente & Conector

Na figura anterior (Fig. 4) está retratado o nosso sistema modificado segundo a vista componente e conector. Quando um utilizador da aplicação "OrderApp" se autenticar e começar a utilizá-la, o que irá acontecer é que qualquer ação vai corresponder a uma chamada remota de uma determinada função que irá ser executada no servidor da "OrderApp", que consequentemente devolve o resultado para o cliente. Todas estas ações são executadas através da comunicação por RMI. Temos portanto uma estrutura do tipo cliente-servidor.

Neste caso temos como componente principal o "OrderApp Server" que é a unidade de processamento de todas as funções que são chamadas remotamente, como já foi explicado anteriormente. Para além disso, este servidor também irá aceder à base de dados (outro componente importante) sempre que for necessário. Como conectores temos a ligação RMI que é efetuada entre o cliente e o servidor.

B. ESCOLHAS EFETUADAS

Para garantir tanto as restrições como os atributos de qualidade tivemos de pensar em diversas opções que conseguissem garantir os *drivers* arquiteturais que tínhamos planeado anteriormente. Numa primeira análise pensamos em uma outra solução que deteríamos para o uso de RMI como base.

Sockets TCP

Em primeiro lugar pensamos em usar *sockets TCP* para enviar dados entre um servidor que está ligado a base de dados e o cliente. Esta solução no entanto encarretava alterar bastante código e não nos fornecia garantias de escalabilidade sem termos de implementar nós mesmos métodos para tal efeito.

Porquê RMI ao invés dos sockets TCP?

O RMI é um RPC (*Remote Procedure Call*) que permite invocar métodos remotamente. Assim as alterações a nível de código foram poucas: foi apenas mudado o código do cliente para o servidor, poupando bastante tempo e arriscando-nos a ter menos erros de implentação dado que o número de linhas alteradas bastante reduzido. O RMI tem algoritmos internos de balanceamento da carga, permitindo assegurar também a escalabilidade. Portanto com esta solução conseguimos garantir as restrições descritas no ponto II, isto é, as encomendas agora são realizadas fora do *site* central (no servidor RMI).

Relativamente aos atributos de qualidade, procuramos diversas formas de os garantir individualmente sem afetar os outros. A seguir iremos explicar de que forma conseguiremos garantir os mesmos, no entanto compreendemos que poderão existir outras formas de conseguir garantir o mesmo atributo de forma talvez mais eficiente.

– **Segurança**

Para além da ferramenta de autenticação efetuada, o nosso sistema está protegido contra *SQL injection*. Assim a probabilidade de alguém danificar remotamente a base de dados através de uso deste tipo de *exploits* pela "Order App" é bastante baixa.

– **Performance**

A nossa arquitetura apresenta um *overhead* reduzido em relação à estrutura original, já que o RMI tem um *overhead* extremamente baixo.

– **Disponibilidade**

A base de dados que está a ser utilizada - “MySql” - garante disponibilidade. No caso da invocação remota também temos essas garantias. Uma característica do RMI é ter um ótimo *load balancing*, permitindo que o sistema consiga escalar bem, estando sempre disponível.

REFERÊNCIAS

[1] Documenting Software Architectures: Views and Beyond, Second Edition, by Clements, et al.. AddisonWesley 2011.