Universidade de Coimbra
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática

Trabalho Prático #2

Arquitectura de Software
2014/2015

André Perdigão Gonçalves
Joni Oliveira
José Miguel Alves

{apcosta,jmcalves,joniro}@student.dei.uc.pt
2010133096 , 2010132936, 2007107187

1 de Abril de 2015

Índice

1. Introdução

O problema apresentado consiste num conjunto de aplicações nas quais seria necessário efetuar alterações de forma a suportar novas funcionalidades. Para efetuar as modificações pedidas foi necessário não só executar mudanças no código fonte, mas principalmente alterações do ponto de vista arquitetural. Assim, o presente documento pretende demonstrar essas mesmas mudanças. O código fonte é entregue em anexo, pelo que, o documento foca, fundamentalmente, nas alterações arquiteturais efetuadas. Desta forma procurar-se-á demonstrar e justificar as modificações realizadas.

1.1. Vista geral do sistema

A aplicação em causa pertence à Exton Plantas Exóticas (EPE), uma empresa business-to-business especializada em árvores exóticas, arbustos e sementes fornecendo plantas para um grande número de clientes de diversas áreas de negócio. Atualmente a empresa utiliza três aplicações. Duas para agilizar o trabalho, sendo que uma serve de apoio às encomendas e outra de apoio à expedição, uma terceira aplicação para inserção de produtos na base de dados.

As encomendas são recebidas por meio de um call-center, a aplicação serve não só para registar os dados dos clientes, bem como os produtos que o cliente pretende encomendar. A secção de expedição utiliza a aplicação para visualizar as encomendas pendentes, sendo que também é possível verificar os dados de uma determinada encomenda. Após a preparação da encomenda e da sua expedição é possível marcá-la como enviada de forma a ser removida da lista de encomendas pendentes. A aplicação de inserção de produtos serve para o gestor de TI inserir produtos na base de dados.

A EPE pretende efetuar grandes investimentos para modernizar e reestruturar as suas aplicações, no entanto é necessário que as aplicações correntes continuem operacionais até o novo sistema estar totalmente desenvolvido.

1.2. Objetivos e contexto

O objetivo do projeto é alterar o sistema de forma a que este possa suportar a introdução de novas encomendas de forma remota. É ainda necessário ter um módulo de *logging* que permita manter um histórico das ações efetuadas por cada funcionário do call-center, bem como manter um registo de entrada e saída do sistema. O módulo de *logging* deve ainda manter um registo de quando cada encomenda é enviada.

As alterações a efetuar são temporárias, uma vez que tal como descrito acima a EPE pretende efetuar grandes investimentos para modernizar e reestruturar as suas aplicações. Assim é necessário analisar arquitecturalmente o sistema e efetuar as devidas alterações, na medida em que poderá ser necessário num futuro próximo adicionar mais funcionalidades ou alterar as que atualmente existem.

1.3. Drivers arquiteturais

De forma a suportar introdução remota de encomendas e registo de atividades das várias aplicações, tal como referido no ponto anterior, a arquitetura do sistema necessita de sofrer alterações. A segurança, interoperabilidade, modificabilidade e extensibilidade são fatores importantes que nos levaram à alteração da arquitetura existente. Com a nova funcionalidade de introdução de encomendas por via remota, é necessário assegurar o sistema e evitar acessos diretos à base de dados, sendo que o *software* legado possuía esse mesmo acesso. Com vista a minimizar o custo de introdução desta arquitetura e existindo já um conjunto de aplicações em utilização, foi pensada numa arquitetura que permite utilizar o *software* atual (aplicações Java e bases de dados) minimizando não só o número de alterações a executar neste mesmo software mas também a sua dimensão. Além disso, a nova arquitetura está preparada para futuras alterações, uma vez que não existe uma data definida para a introdução do novo *software* que está a ser desenvolvido.

Assim sendo é extremamente fácil adicionar novas funcionalidades, como tornar todas as aplicações remotas e construção de uma plataforma web com as funcionalidades do *software*. Sendo o orçamento reduzido e as *deadlines* curtas optámos por uma arquitetura que minimiza as modificações a realizar sobre o software já desenvolvido.

2. Arquitetura proposta

Nesta secção apresentamos a arquitetura produzida, como suporta os requisitos e vantagens e desvantagens relativamente a outras tecnologias e arquiteturas.

2.1. Descrição

A nova arquitetura é constituída por diferentes camadas (n-camadas). O objetivo é separar o armazenamento de dados, a *business logic* e o cliente. Com estes componentes a arquitetura possui então 3 camadas. Existe uma camada de gestão de dados, que possui uma base de dados em *MySQL*, uma camada que trata do processamento *da Business Logic* e uma camada relativamente ao cliente. A *Business Logic* foi implementada recorrendo a ESB (*enterprise service bus*) da plataforma Mule. Através deste *software* existem *web services* que recorrem ao protocolo SOAP para comunicação.

2.2. Suporte aos drivers e requisitos

A nova arquitetura foi escolhida pela facilidade de implementação e pelos seus atributos que suportam os drivers arquiteturais. A escolha de uma arquitetura em camadas pretende acrescentar um novo nível de segurança, maior desempenho, escalabilidade, extensibilidade e modificabilidade. No *software* legado, sendo que a sua utilização era interna (*on site*), não houve preocupação com a segurança no sentido em que os acessos à base de dados eram diretos a partir dos diferentes clientes. Com o objetivo de permitir acessos remotos, foi necessário modificar a

arquitetura de forma a impedir acessos diretos à base de dados, sendo estes pedidos processados por uma camada intermédia, a *business logic*. A escolha de uma arquitetura em camadas e uma *business logic* em *web services*, permite suportar os diversos drivers arquiteturais. O acréscimo de novas funcionalidades, como o exemplo de todos os clientes serem remotos, basta modificar a *business logic*, adicionando novos *endpoints* aos *web services*. Na eventualidade de um servidor web e uma base de dados não serem suficientes para suportar os diversos clientes, a arquitetura é escalável sendo possível adicionar novos servidores (réplicas de base de dados, *load-balancers*, etc). Além da fácil integração de novos *endpoints* nos *web-services*, passa a existir já uma plataforma para a extensão das funcionalidades, como a construção de um *website* que inclui as várias funcionalidades dos softwares já existentes. Sendo uma arquitetura com funcionalidades repartidas por diferentes camadas, o desempenho será superior sendo que cada camada tem apenas uma funcionalidade base. No caso de eventual falha de alguma camada, a sua manutenção será essencialmente mais fácil dada a única funcionalidade presente na mesma camada.

Dado o budget reduzido consideramos que esta arquitetura suporta todos os requisitos necessários, acrescentando também algumas vantagens. Apesar da arquitetura desenvolvida ser intermédia, fornece já um ponto de partida, podendo ser expandida e facilmente modificada.

2.2.1. Segurança

Para que a API oferecida pelos *web services* possa ser utilizada, é necessário que o utilizador esteja autenticado. No acto de login, caso o utilizador seja validado, é criado um token único que será usado em futuros *requests*. Uma vez que o utilizador faça logout, o token é eliminado da base de dados. Desta forma, conseguimos garantir que só os funcionários que fizeram login tem acesso às funcionalidades do sistema. Não é garantida a segurança das aplicações *on-site* (*ShippingApp* e *InventoryApp*) dado que estas comunicam diretamente com a base de dados.

2.2.2. Persistência

Persistência dos dados é garantida pela base de dados relacional *MySQL*.

2.2.3. Performance

Dependente do hardware onde se encontram as camadas. O desempenho pode aumentar substituindo por hardware diferente ou por acréscimo de servidores, tanto ao nível das bases de dados como dos *web services*.

2.2.4. Interoperabilidade

Por ser uma arquitetura com base em *web services*, a integração com outras aplicações torna-se bastante simples. Qualquer aplicação pode passar a comunicar, através de SOAP (e REST), com a *business logic* e trocar dados, podendo ser criada qualquer tipo de aplicação sobre a arquitetura existente.

2.3. Alterações realizadas

Foram necessárias alterações ao *software* existente e base de dados de forma a colocar a arquitetura funcional.

Base de dados

A base de dados encontrava-se incorretamente estruturada e com vista a futuras alterações e acréscimo de funcionalidades, foi modificada a forma como encomendas são criadas. Ao invés de criar uma tabela por encomenda com a lista de produtos, existe agora uma tabela geral ("*productsinorders*") que contém os diferentes produtos para todas as encomendas. Desta forma as *queries* são facilitadas e o uso incorreto de tabelas foi removido. Foi criada também uma base de dados para os utilizadores que contém a informação de autenticação.

Além das modificações realizadas consideramos que a base de dados (conjunto de diferentes bases de dados) deve ser reestruturada de forma a

normalizar tabelas e fazer a junção das diferentes bases de dados (*as_users*, *orderinfo* e *inventory*) em apenas uma.

OrderApp

Esta aplicação foi a que mais alterações sofreu para suportar o a funcionalidade de criação remota de encomendas. Sendo assim foram removidas todas as funcionalidades de acesso à base de dados (*queries*) passando a implementar agora os *web services* (WSDL). Uma pequena alteração na interface também foi necessária para permitir a autenticação dos empregados.

ShippingApp

Esta aplicação passa agora a ter acesso aos *web services* para poder fazer a atualização dos registos de alterações a encomendas. Como foram realizadas alterações na base de dados relativamente à forma de como os produtos em encomendas são armazenados, a *query* que seleciona os produtos numa encomenda foi alterada para aceder à nova tabela *productsinorders*.

2.4. Comparações e *tradeoffs*

Como já foi referido anteriormente, a arquitetura atual é um modelo de 3 camadas, sendo a camada intermédia a *business logic* constituída por *web services* que fazem uso do protocolo SOAP. A principal razão de escolha desta solução em detrimento de um modelo cliente-servidor está relacionada com os diversos drivers arquiteturais identificados anteriormente, nomeadamente a segurança e a escalabilidade. A arquitetura anterior (2 camadas) consistia numa ligação direta entre o cliente e a base de dados, comprometendo totalmente a escalabilidade do sistema, dado que com o crescimento esperado iria existir uma elevada latência. Desta forma, dado que a empresa está em crescimento, é necessário ter em conta que a nova arquitetura tem de ser escalável e segura, logo a opção por um modelo de 3 camadas será o ideal para cumprir esse requisito. Dado que o sistema tem 3 camadas, será também mais fácil a extensibilidade do mesmo, visto que se forem adicionadas mais

funcionalidades não será necessário actualizar individualmente cada cliente, como seria na arquitetura anterior.

Uma vez definida a arquitetura, a opção recaiu sobre um *web service* com o protocolo SOAP em vez de REST. Desta forma, no futuro, o sistema pode vir a ser modificado para que haja introdução de extensões que podem ser úteis tais como WS-Security que torna o *web service* seguro. Para além de *web services* poderiam ter sido utilizadas invocações remotas como é o caso do RMI. Porém, a arquitetura ficaria limitada a programas escritos na linguagem Java, pelo que do nosso ponto de vista seria desvantajoso. Através dos *web services* eliminamos a dependência de uma linguagem e no futuro poderão ser desenvolvidos web sites ou aplicações noutra linguagem que possam fazer uso desses mesmos web services. Outra opção poderia passar por EJB (*Enterprise Java Beans*), mas mais uma vez existiria o mesmo problema relatado para o RMI.

3. Vistas

3.1. Vista lógica

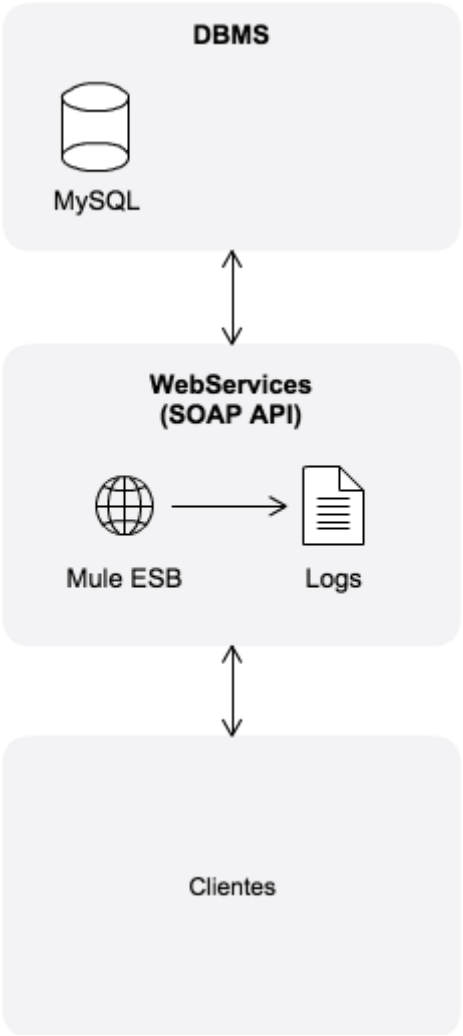

Figura 1. Vista geral da arquitetura.

3.2. *Deployment*

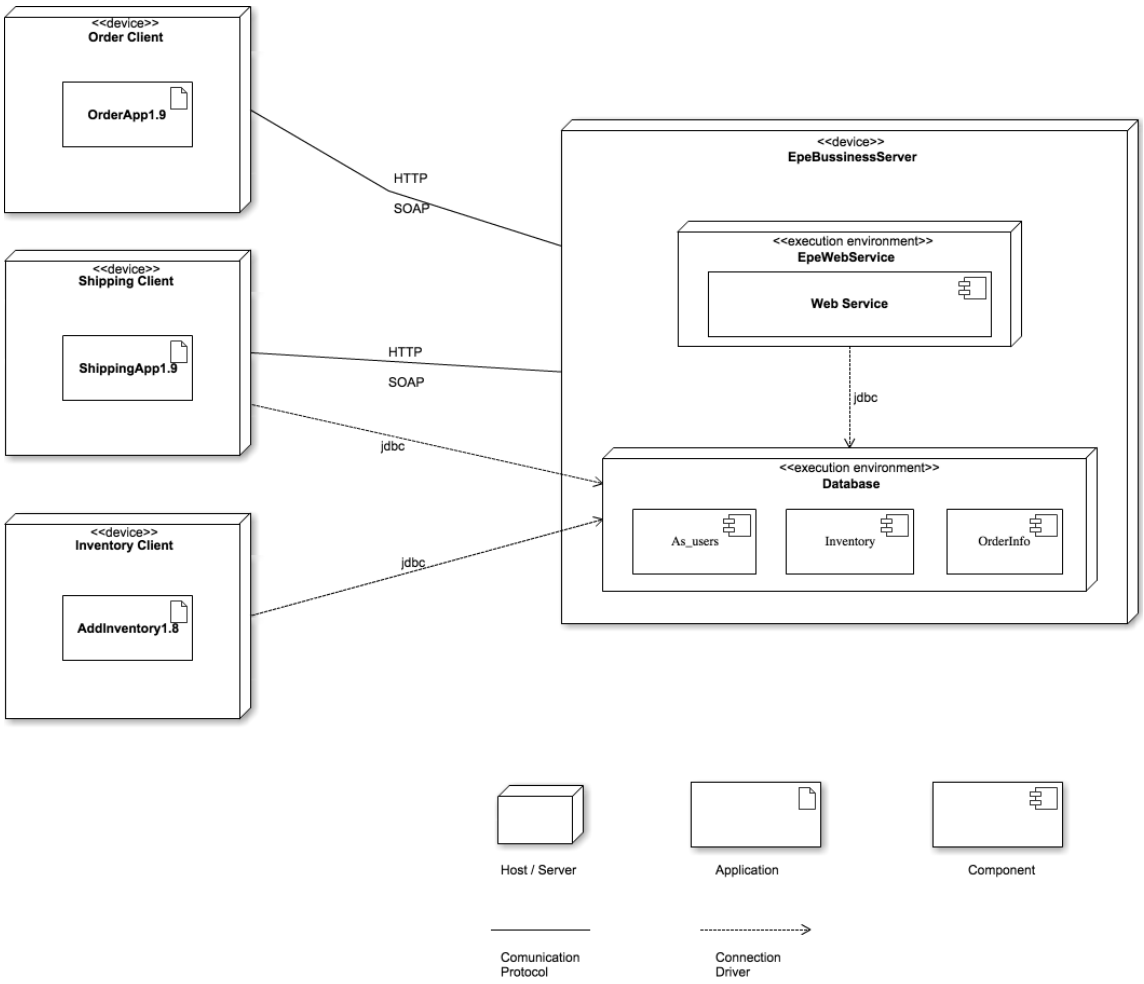

Figura 2. Diagrama de *deployment*.

3.3. *Dynamic perspective*

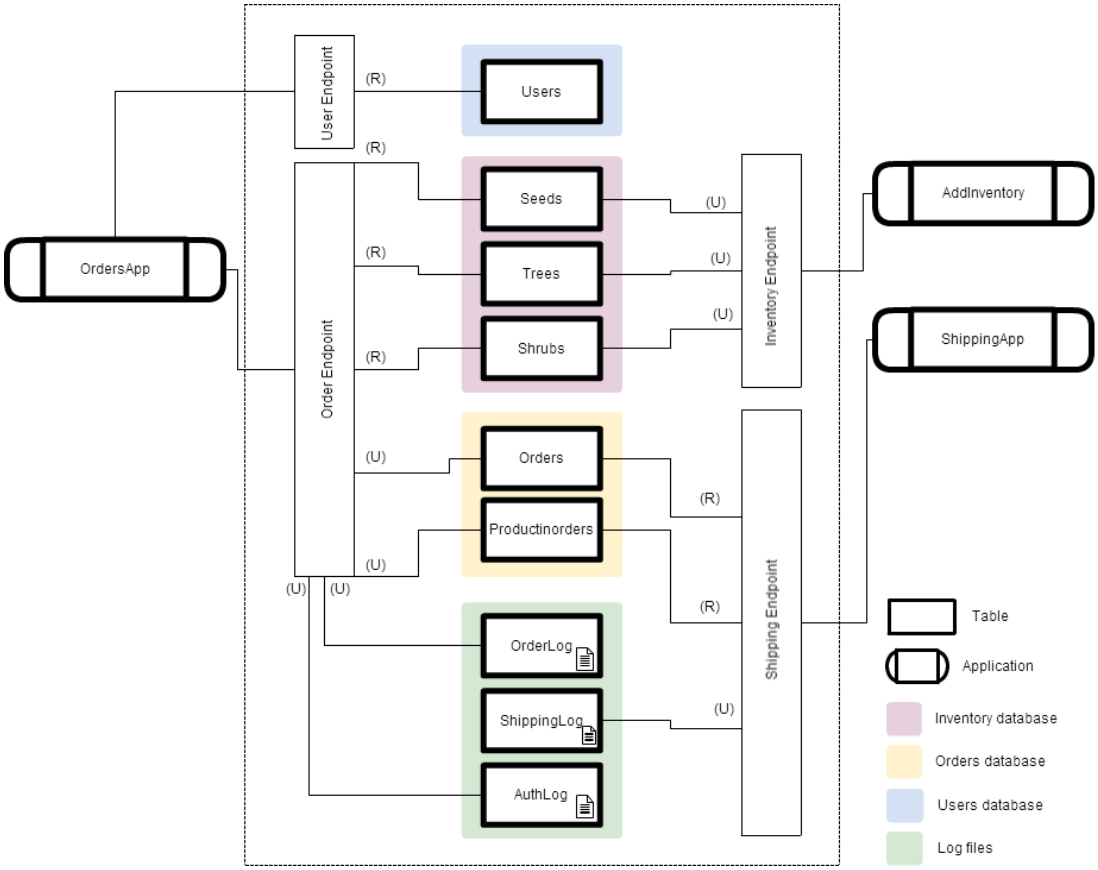

Figura 3. Interação dos clientes com *endpoints* e bases de dados.

Todos os *endpoints* podem efetuar operações CRUD mas no âmbito do projeto apenas algumas operações foram implementadas. Os *web services* permitem fácil extensão desta funcionalidade, permitindo estender a API existente.

3.4. Componentes dos *web services*

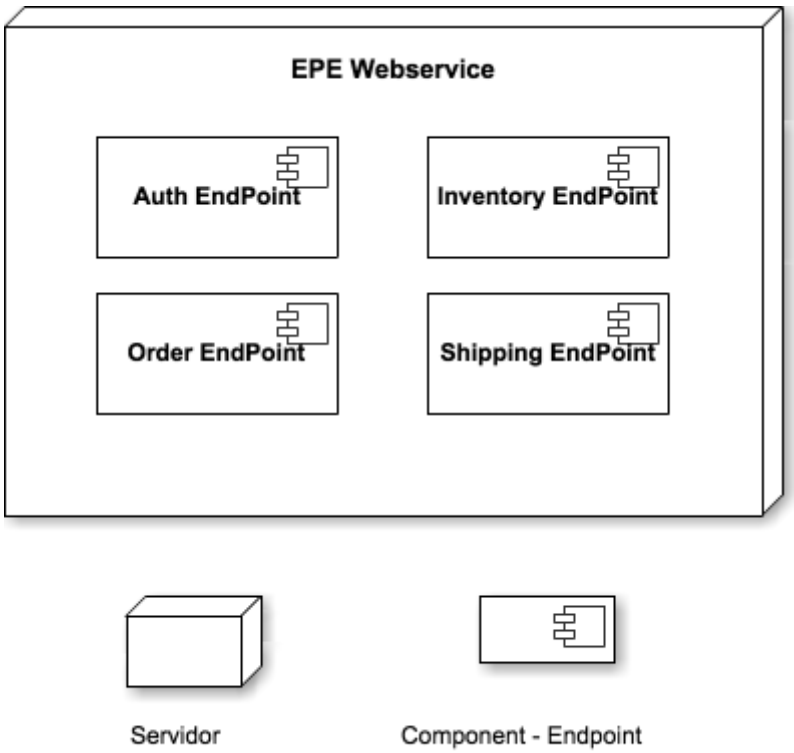

Figura 4. *Endpoints* desenvolvidos para o *web service*.

3.5. Modelo relacional

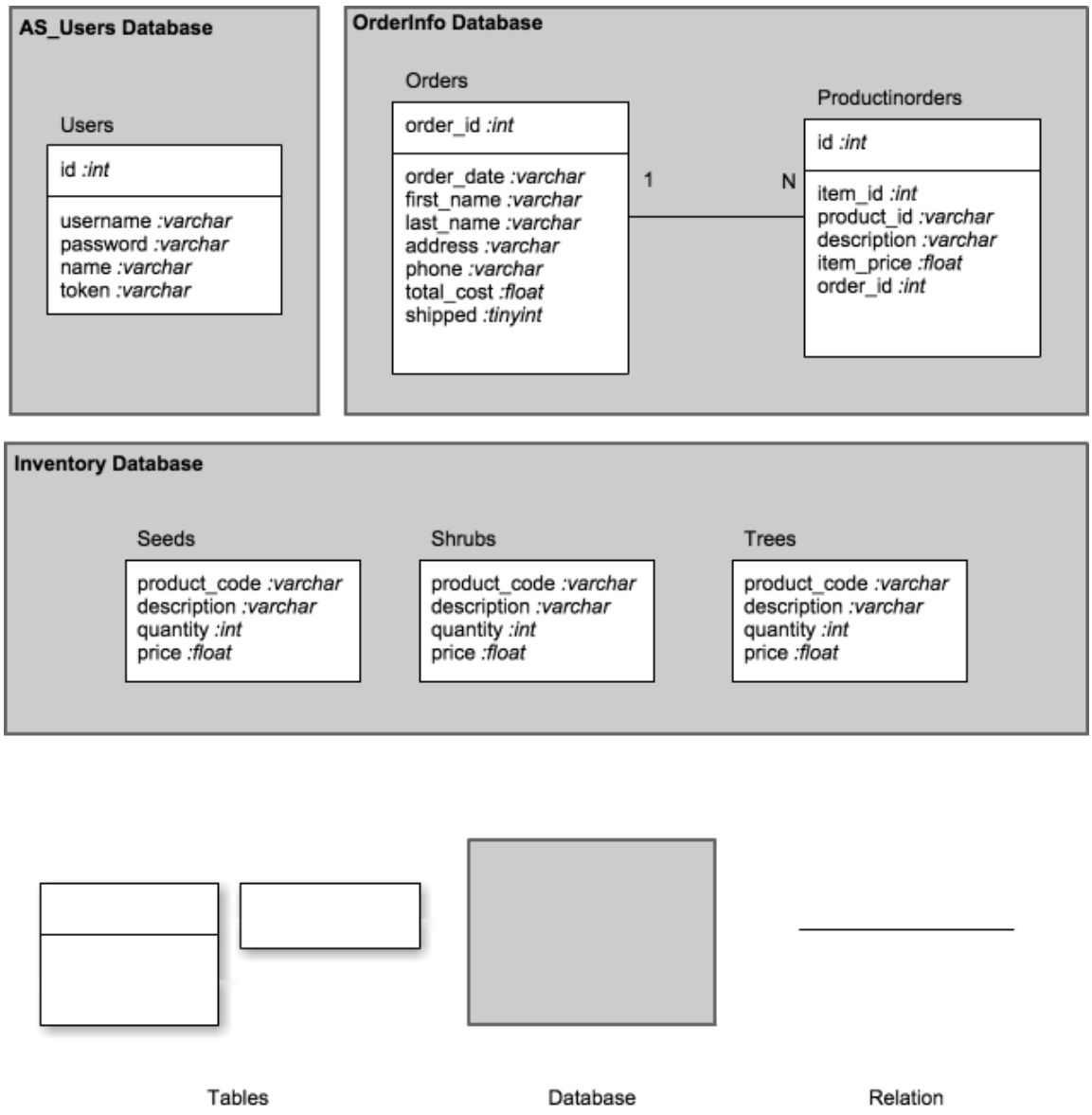

Figura 5. Modelo relacional da base de dados.

3.6. Casos de uso

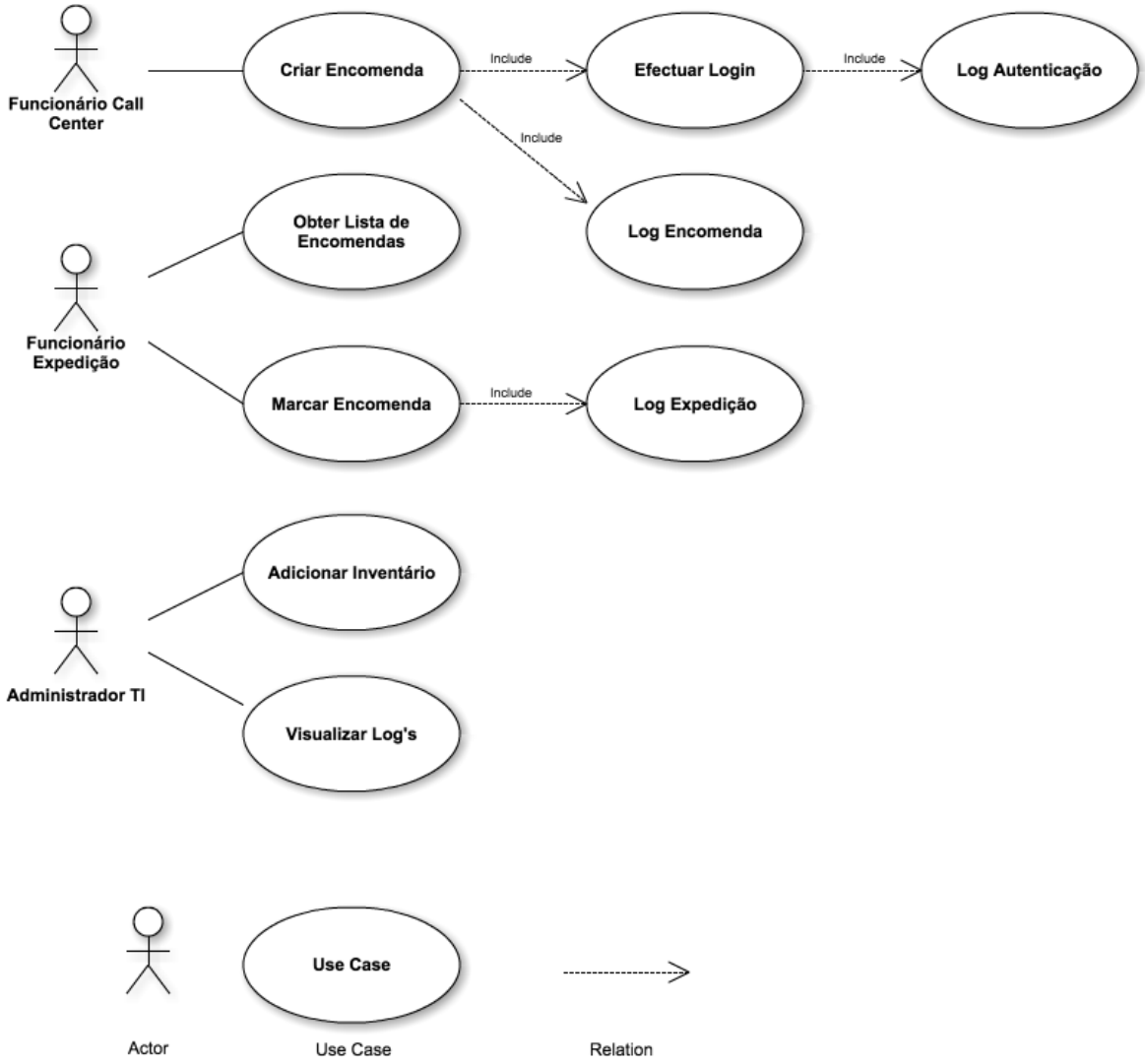

Figura 6. Diagrama de caso de usos das várias funcionalidades do sistema.

3.7. Diagramas de atividade

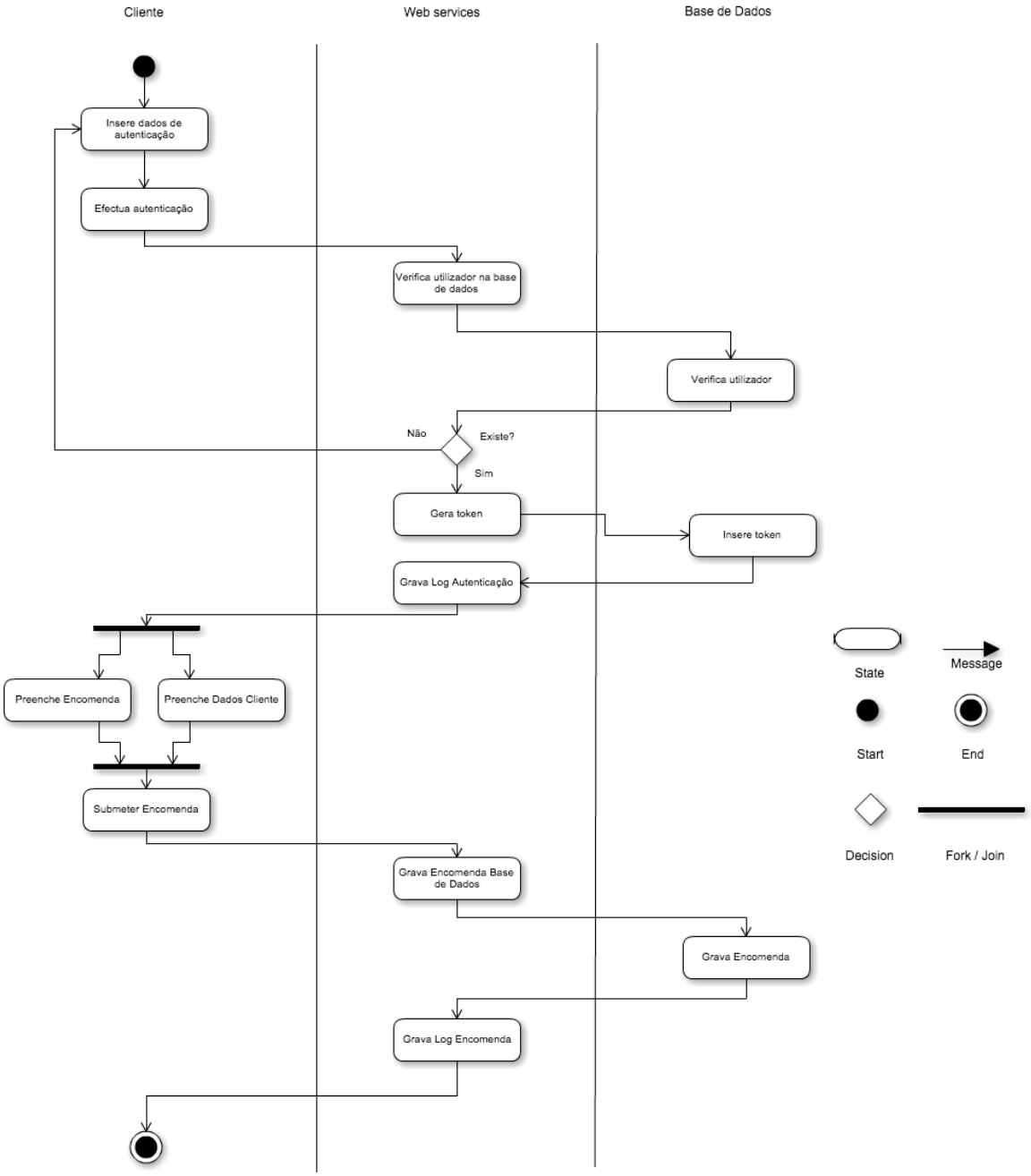

Figura 7. *OrderApp* - Criar encomenda

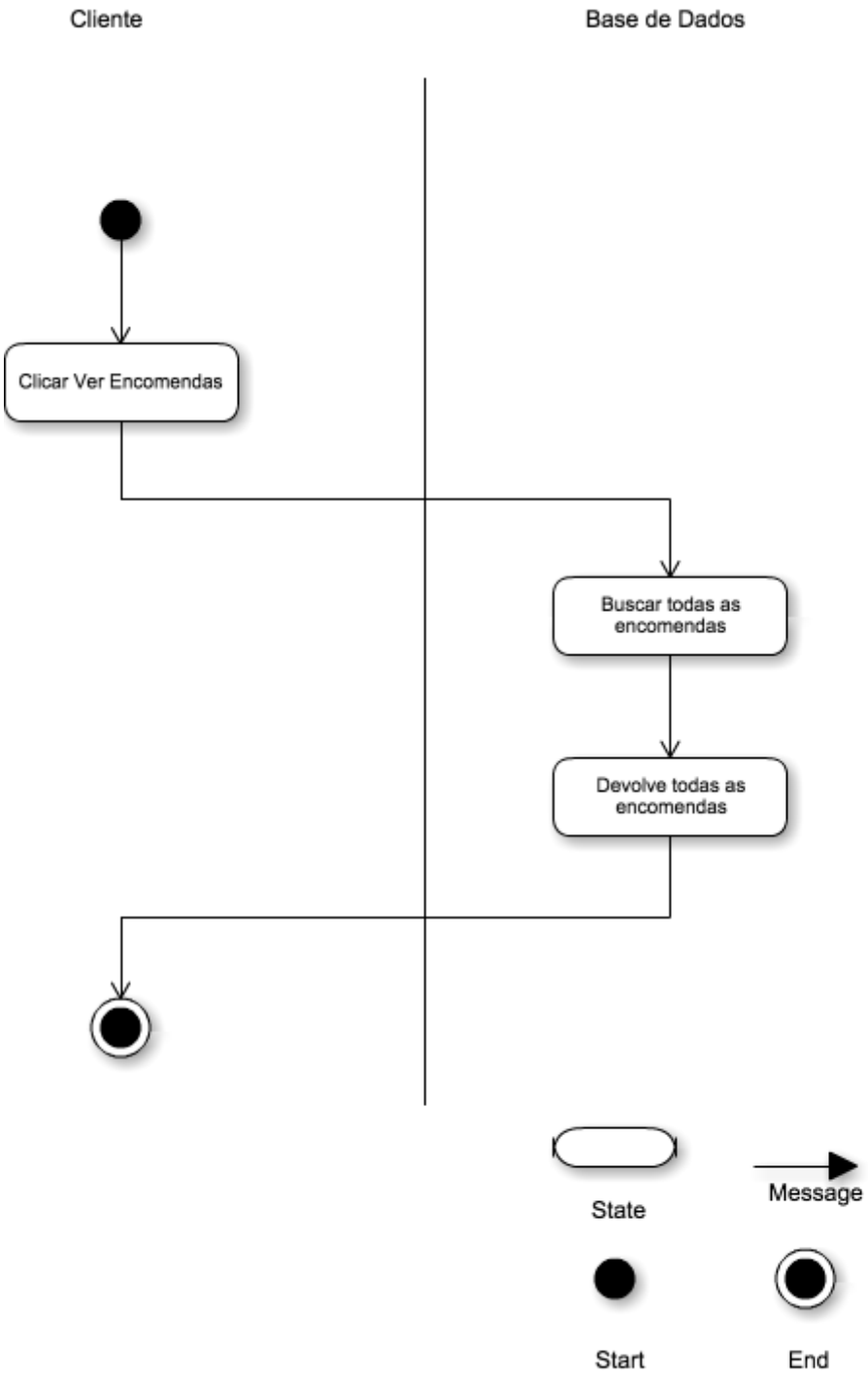

Figura 8. *ShippingApp* - Ver encomendas

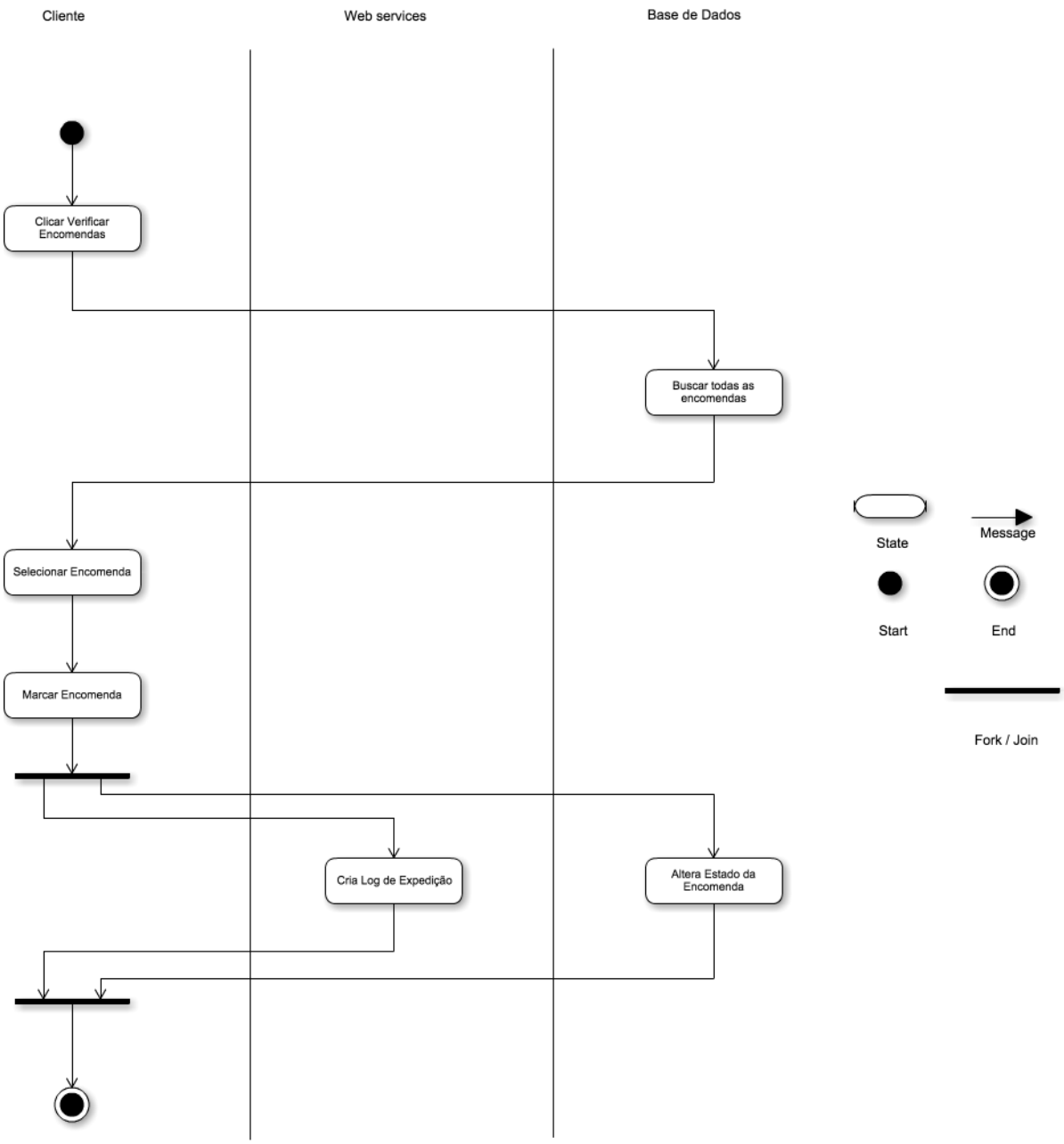

Figura 9. *ShippingApp* - Alterar estado de encomenda

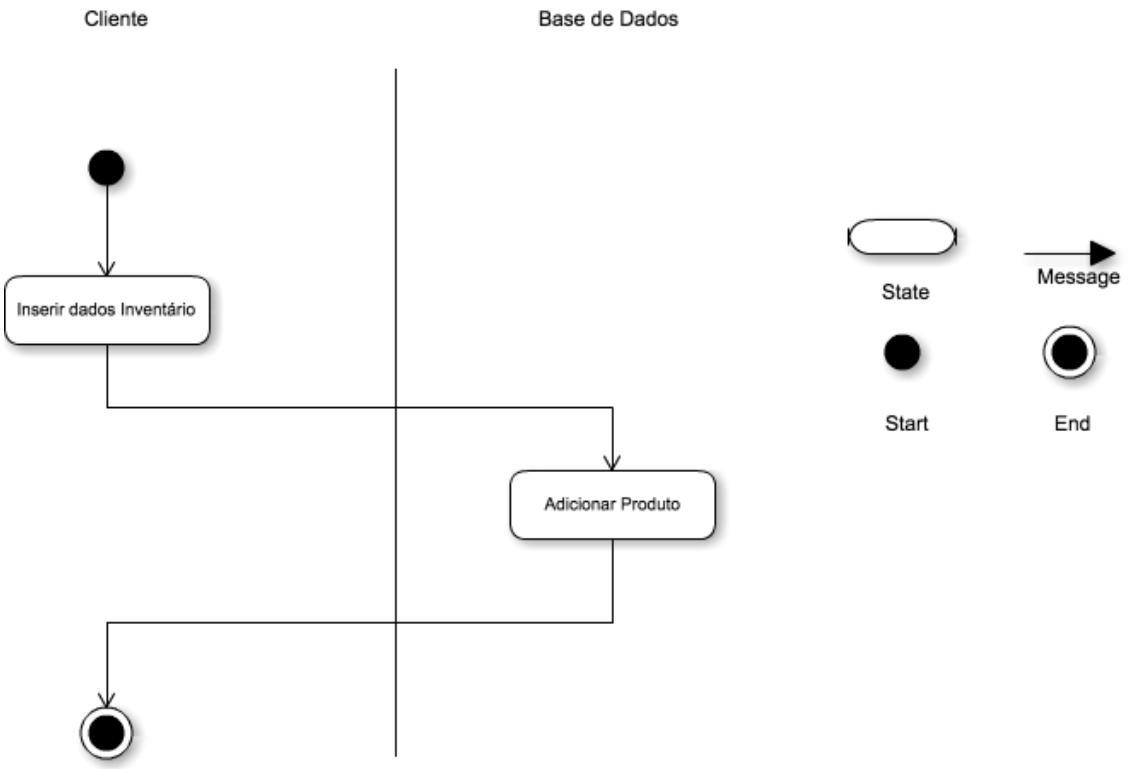

Figura 10. *InventoryApp* - Inserir inventário

4. Deployment

Para efeitos de teste os *webservices* estão a correr num servidor *cloud* fornecido pelo *software* Mule. Desta forma permitimos que a *arquitectura* seja testada facilmente.

No arquivo que contém o projeto está incluído todo o código fonte juntamente com 2 *artifacts* que permitem usar os *webservices* na *cloud* ou em servidor *localhost* mas para isso será necessário importar as *flows* do ficheiro mule-flows.xml para o software Mule ESB.

Dado que, por motivos de teste, estamos a usar o servidor *cloud*, o acesso aos ficheiros de registo torna-se impossível mas o objetivo é mostrar a arquitetura funcional. Para verificar os ficheiros de registo será necessário utilizar um servidor *localhost*, que representa o servidor local (on-site) que apenas o administrador TI pode aceder.