Universidade de Évora
Honesto Estudo com Longa Experiência Misturado
Departamento de Informática

Mestrado em Engenharia Informática

Procedimentos de Informação Portuária Electrónica

Carlos Manuel Silva Boita
<m18578@alunos.uevora.pt>

Orientador: *Luís Arriaga da Cunha*

Évora, 18 de Julho de 2008

Esta dissertação não inclui as críticas e sugestões feitas pelo júri.

Procedimentos de Informação Portuária Electrónica

Carlos Manuel Silva Boita
<m18578@alunos.uevora.pt>

Orientador: *Luís Arriaga da Cunha*

Esta dissertação não inclui as críticas e sugestões feitas pelo júri.

AGRADECIMENTOS

Deixo aqui o meu profundo agradecimento ao professor Luís Arriaga da Cunha, pelo seu apoio, disponibilidade e motivação, que me permitiu aprofundar e aprender novos conceitos.

À Indra Sistemas, que me deu a oportunidade de participar num projecto emergente e com muito bons ingredientes para explorar e aprender. A toda equipa do Pipe, pelo apoio, companheirismo e compreensão.

Aos meus amigos, pela compreensão, carinho e apoio.

Aos meus familiares que sempre me deram apoio nos momentos mais difíceis e me apoiaram em seguir em frente para alcançar os meus objectivos.

A todos os que directa ou indirectamente contribuíram.

Tabela de Conteúdo

Lista de Figuras

Lista de Quadros

Glossário e Siglas

APP	Associação de Portos de Portugal
BPEL	*Busines Process Execution Language*. Uma linguagem baseada em XML utilizada para orquestrar serviços para serviços compostos ou processos. Os serviços resultantes são *Web Services*.
BPM	*Business Process Manager*. Termo genérico que se refere a todas as actividades realizadas para controlar processos de negócio (*business processes*).
Business Process	Uma descrição estruturada de actividades ou tarefas que têm de ser feitas para resolver uma necessidade de negócio.
Business to Business (B2B)	O *Business to Business* é um conceito relativamente recente de comércio electrónico que utiliza a Internet como canal de comunicação entre entidades, nomeadamente empresas. No B2B as empresas visitam-se pela Internet, e aí podem utilizar serviços, efectuar compras,

	consultas e troca de produtos, através dos serviços e funcionalidades que disponibilizam na rede.
EAI	*Enterprise Application Integration.* Uma aproximação para integrar sistemas distribuídos que utilizam uma infra-estrutura comum. Com esta aproximação, para cada sistema fornecer basta manter apenas um adaptador específico para cada um dos sistemas com os quais ele comunica. SOA pode ser descrito como uma extensão EAI que fornece o aspecto técnico a que chamamos interoperabilidade.
EDI	*Electronic Data Interchange*, pode ser definida como o movimento electrónico de documentos padrão de negócio entre entidades (internas ou externas). O EDI usa um formato de dados estruturado de recolha automática que permite que os dados sejam transformados sem serem reintroduzidos.
EDIFACT	*Eletronic Data Interchange for Administration, Commerce and Transport* é um padrão tradicional de EDI, para descrição textual de documentos

	visando o armazenamento e envio por meios electrónicos.
ESB	*Enterprise service bus.* A infra-estrutura de um ambiente SOA que permite a interoperabilidade dos serviços. As suas principais funções são fornecer conectividade, transformações de dados, e *routing*, permitindo que diferentes sistemas possam comunicar via serviços. O ESB possui outras capacidades relacionadas com segurança, fiabilidade, controlo de serviços, e composição de processos.
ETL	*Extract Transform Load.* Ferramentas de software cuja função é a extracção de dados de diversos sistemas, transformação desses dados mediante regras de negócios e por fim a carga dos dados.
HTTP	*HyperText Transfer Protocol.* O protocolo principal da *World Wide Web.* De uma forma segura é conhecida por HTTPS.
IDE	*Integrated Development Environment.* Um ambiente orientado ao desenvolvimento de software específico. (Exemplo: NetBeans e

	Eclipse)
INE	Instituto Nacional de Estatística
Interoperabilidade	A capacidade de diferentes sistemas comunicar entre si. Interoperabilidade entre diferentes plataformas e diferentes linguagens é o objectivo principal do SOA.
IPTM	Instituto Portuário e dos Transportes Marítimos
JMS	*Java Message Service*. Uma API do Java para *message-oriented middleware* (serviços de mensagens).
JUP	Janela Única Portuária
Orquestração	Uma forma de agregar serviços a processos de negócio. A orquestração reúne vários serviços num novo serviço que tem o controlo central ao longo de todo o processo. Para *Web Services*, o BPEL é um standard para a orquestração.
PCOM	Plataforma Comum Portuária
Processo	Um conjunto estruturado de actividades/tarefas para obter uma determinada necessidade ou atingir um determinado objectivo.

RMI	*Remote Method Invocation*, é uma interface de programação que permite a execução de chamadas remotas no estilo RPC em aplicações desenvolvidas em Java. É uma das abordagens da plataforma Java para fornecer as funcionalidades de uma plataforma de objectos distribuídos.
RPC	*Remote Procedure Call*. É um protocolo no qual um programa pode chamar um serviço de outro programa localizado noutro computador numa rede sem ter que entender os detalhes desta.
SEF	Serviço de Estrangeiros e Fronteiras
Serviço	Em tecnologias da informação, pode ser designado como a resolução de funcionalidades de negócio. Tecnicamente, um serviço é a descrição de uma ou mais operações que utilizam troca de informação entre um fornecedor e um cliente.
SOAP	*Simple Object Access Protocol*, é um protocolo para troca de informações estruturadas numa plataforma descentralizada e distribuída, utilizando tecnologias baseadas em XML.

Workflow	Na maioria das organizações, principalmente empresariais, os procedimentos que cada interveniente assume no contacto com a documentação, informação ou tarefas destas, devem estar sob uma estrutura organizacional adequada aos objectivos de negócio da organização. O *workflow* surge para responder à necessidade de criar um conjunto de regras que venham ajudar na obtenção destes objectivos. *Workflow* controla todas as actividades que constituem um processo, sugerindo o melhor fluxo a seguir, tendo em conta regras implementadas à partida.
XML	*Extensible Markup Language,* é uma linguagem capaz de descrever diversos tipos de dados. O seu propósito principal é a facilidade de partilha de informações através da Internet. Entre linguagens baseadas em XML incluem-se XHTML (formato para páginas Web), por exemplo.

Resumo

A integração de sistemas é uma necessidade cada vez mais necessária face à exigência e competitividade dos negócios. Uma forma de respondermos a estes factos é através da tecnologia, acompanhando-a e aprofundando-a.

O presente documento apresenta uma proposta de solução tecnológica a estes factos. O caso que iremos abordar trata-se da necessidade de renovar o sistema de gestão portuária a nível nacional, garantido uma continuidade de negócio, assim como a implementação de novas funcionalidades que permitam a esta plataforma ser totalmente independente e integrável com outros sistemas e/ou adaptável a outras organizações no futuro.

SOA, *Service Oriented Architecture*, foi a filosofia adoptada para garantir a facilitação do tráfego marítimo através da harmonização de processos e procedimentos entre os vários portos, fundamentalmente no referente à interconexão e interoperabilidade, e a partilha e troca electrónica de informação processual entre os vários membros da comunidade marítima e portuária.

Procedures for Port Electronic Information

System integration is a necessity progressively more required due to the business demand and competitiveness. It is possible to respond to the latter facts with technology, by following it and developing it.

This study proposes a technological solution for the abovementioned facts. The case to be addressed is the need to renovate the port management system in a nationwide scale, assuring simultaneously the continuity of the business and the implementation of new functionalities that allow for this platform to be utterly independent and applicable to other systems and/or adaptable to other organizations in the future.

SOA, *Service Oriented Architecture*, was the theory adopted to guarantee the simplification of the maritime traffic, by harmonising the processes and the procedures among the several ports, especially concerning the interconnection and interoperability, and the electronic sharing and exchange of information referent to processes amongst the numerous members of the maritime and port community.

1 Introdução

No tempo corrente, verificando a existência de uma forte informatização global e com a contínua evolução tecnológica, deparamo-nos com a necessidade de seguir este ritmo vertiginoso atentamente, de forma a melhorar incessantemente o funcionamento das organizações atendendo à competitividade e ao crescimento dos negócios. Para que isto possa acontecer, é fundamental quebrar as barreiras organizacionais e geográficas, que impedem que os actores intervenientes tenham um melhor desempenho no seu trabalho. Uma arquitectura orientada a serviços (SOA) apresenta-se como uma solução viável para a obtenção de equilíbrio entre instituições/organizações, reduzir custos, melhorar a produtividade e sustentabilidade, evitar o desperdício e optimizar processos e a comunicação.

A APP, Associação dos Portos de Portugal, com o desejo de contribuir para o desenvolvimento e modernização do Sistema Portuário Nacional, lançou o desafio de desenvolver e actualizar as aplicações informáticas de gestão portuária (aplicação PCOM) de todos os portos nacionais, tendo como objectivo a criação de uma nova aplicação denominada JUP (Janela Única Portuária), que centralizará todos os processos de gestão portuária dos portos Douro e Leixões, Lisboa e Sines.

Partindo deste desafio, e tendo a possibilidade de fazer parte da equipa integrante deste projecto surgiu a motivação para a realização de um estudo sobre a melhor forma de abordar o problema tendo em vista os aspectos acima referidos.

1.1 Objectivos

O objectivo deste trabalho é apresentar uma abordagem SOA sobre o sistema JUP, expondo a arquitectura utilizada e as tecnologias e ferramentas envolventes, assim como efectuar um estudo em torno desta.

1.2 Enquadramento Institucional

A Indra Sistemas Portugal S.A é uma subsidiária da Indra Sistemas, multinacional espanhola, líder no sector das Tecnologias de Informação. A Indra proporciona aos seus clientes apoio e serviços para as mais diversas áreas de mercado, tanto na Industria e Comércio em geral como em áreas mais específicas, nomeadamente; Transporte e Tráfego, Energia, Utilidades e Operadores, Administração Pública e Saúde, Defesa e Forças de Segurança, Finanças e Seguros.

A APP – Associação dos Portos de Portugal é uma Associação sem fins lucrativos constituída em 1991, com o objectivo de ser o fórum de debate e troca de informações de matérias de interesse comum para os portos e para o transporte marítimo.

1.3 Organização da Dissertação

No próximo capítulo apresentaremos a JUP, Janela Única Portuária, expondo a descrição do sistema assim como os seus intervenientes. Seguidamente, no capítulo 3 faremos a exposição dos conceitos e tecnologias abordadas no projecto. No capítulo 4 discutiremos as metodologias e procedimentos adoptados na implementação do sistema JUP. Posteriormente, no capítulo 5, apresentaremos um caso de estudo sobre um dos processos implementados no sistema JUP. Por fim, discutiremos e concluiremos acerca das escolhas e decisões tomadas no projecto.

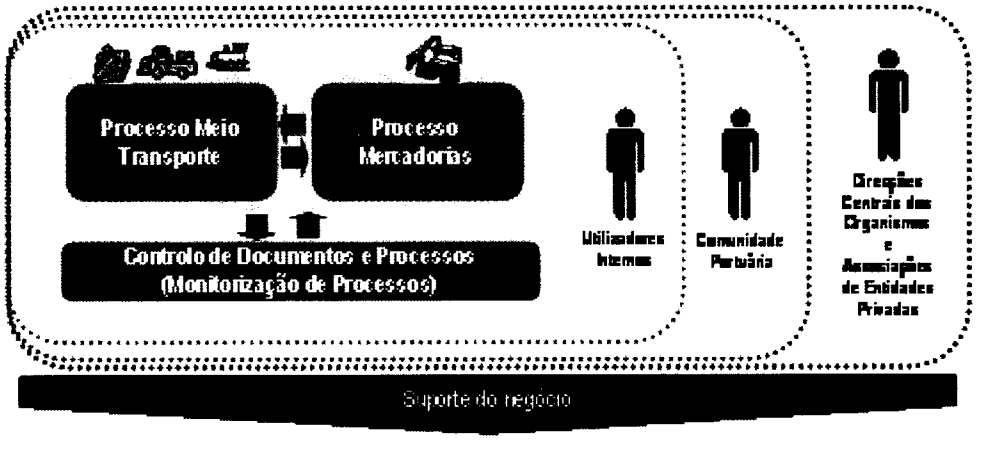
2 Janela Única Portuária

Neste capítulo iremos apresentar o sistema JUP especificando as suas características principais, assim como os seus intervenientes. Na secção 2.1 iremos apresentar a descrição do sistema e na secção 2.2 os actores do sistema e a forma como estes interagem no sistema.

2.1 Descrição do sistema JUP

O sistema permite gerir toda a operação portuária, não só do ponto de vista da autoridade portuária como também do ponto de vista das várias entidades que compõem a comunidade portuária.

Podemos dividir o sistema em três conjuntos:



JUP – Janela Única Portuária

Ilustração 1 – Visão geral do negócio.

- **Meios de Transporte** – Neste conjunto constam todos os processos relacionados com as operações dos meios de transporte realizadas dentro da área portuária: Comboio, Camião e obviamente Navio.

- **Mercadorias** – Neste conjunto constam os processos relacionados com o registo de entrada e saída da mercadoria, bem como as obrigações declarativas referentes a esta, mais especificamente a declaração de mercadorias à alfândega através de um conjunto de documentos dos quais se destaca o Manifesto.

- **Controlo de Processos** – Este conjunto compreende a monitorização, por parte da autoridade portuária das actividades relacionadas com os meios de transporte e mercadorias.

A principal missão do sistema é responder às necessidades quer do meio de transporte, quer da mercadoria nele transportada, de uma forma célere e eficiente. Para tal, deve permitir a solicitação de todo o tipo de produtos e serviços necessários numa escala e fornecer meios às entidades competentes para responder a esses pedidos.

O sistema está intimamente ligado ao sistema da Autoridade Aduaneira, existindo uma relação *Business to Business* (B2B) entre os dois sistemas. A informação registada é partilhada por ambos e deve também estar acessível às restantes entidades da comunidade portuária.

Deve também conseguir comunicar com os sistemas das várias instalações portuárias, sejam estas de operadoras de contentores ou de carga geral/granéis.

Finalmente, o sistema deve disponibilizar a informação adequada às várias autoridades nacionais competentes (IPTM, INE, VTS Costeiro, entre outros).

Além de fomentar a integração de toda a comunidade portuária (Autoridade Portuária, Autoridade Aduaneira, Capitania, SEF, Sanidade, Operadores Portuários, etc.), para que os pedidos sejam efectuados uma só vez, sendo a informação compartilhada por todos, com as questões de confidencialidade da informação asseguradas, o sistema também permite a integração com sistemas internos, como sistemas de facturação, estatística e recursos humanos. Na figura abaixo podemos verificar todas estas integrações assim como uma visão geral das características da JUP.

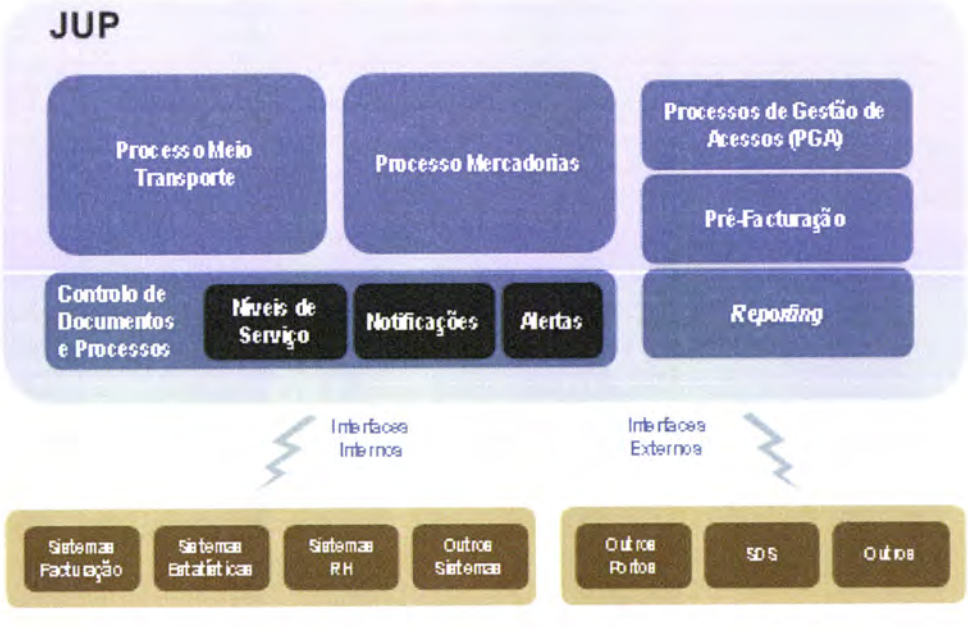


Ilustração 2 – Caracterização da plataforma JUP.

2.2 Actores do sistema e interacção com a JUP

Podemos dividir os actores intervenientes no sistema nos seguintes grandes grupos, de acordo com o tipo de intervenção que têm no sistema:

Agentes	São os representantes perante a restante comunidade do navio, comboio, camião e até mesmo da mercadoria. São os responsáveis por entregar um conjunto de declarações e pedido de despacho que são geralmente denominados por obrigações declarativas.
Autoridades	São responsáveis pela emissão de autorizações para a maioria das actividades do navio ou mercadoria. Sem as autorizações deste grupo um navio não pode entrar ou sair de porto, descarregar/carregar ou até mesmo manobrar em porto. Neste grupo encontram-se a capitania do porto, a autoridade de saúde, a alfândega e a administração do porto.
Prestadores de Serviços	Dentro da área portuária existe um conjunto de serviços que podem ser executados sobre o navio ou sobre a carga. São exemplos de serviços, o abastecimento de provisões, a recolha de resíduos de um navio ou o aluguer de equipamento. Estes serviços são prestados por um conjunto de entidades bem identificado ou até pela própria autoridade portuária.

Tabela 1 – Intervenientes do sistema.

Na figura abaixo podemos ver, de forma esquematizada, como estes actores intervêm no sistema:

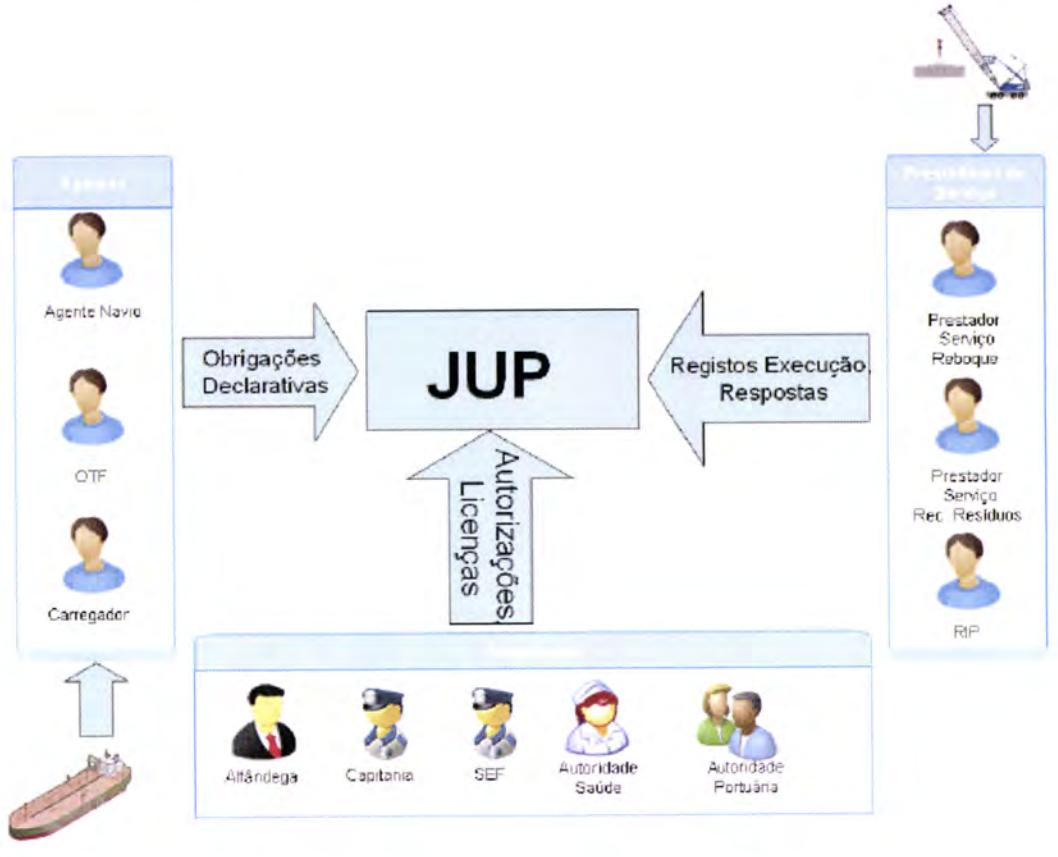


Ilustração 3 – Interacção da Comunidade Portuária com a JUP.

2.3 Síntese

Neste capítulo ficámos a conhecer o sistema JUP e os seus intervenientes.

O sistema divide-se em três conjuntos principais: meios de transporte, mercadorias e controlo de processos. O sistema integra outros sistemas (como facturação) e comunica com sistemas externos como o SDS.

Os seus intervenientes principais são os Agentes, Autoridades e Prestadores de Serviços.

3 Conceitos e Tecnologia

No presente capítulo iremos mergulhar no contexto tecnológico que envolve o sistema JUP. Iremos abordar vários conceitos em torno das tecnologias utilizadas no nosso trabalho e traçar a ponte para a metodologia adoptada, descrita no capítulo seguinte.

Começaremos com uma apresentação genérica de sistemas distribuídos e integração de sistemas na secção 3.1, mencionando alguns exemplos como as bases de dados distribuídas, processos distribuídos e sistemas de troca de mensagens (nas secções 3.1.1, 3.1.2 e 3.1.3).

Seguidamente, apresentaremos o paradigma SOA (*Service-Oriented Architecture*) na secção 3.2, realçando conceitos importantes que o caracterizam, como *Web Services* na sub-secção 3.2.1, BPM (*Business Process Manager*) na sub-secção 3.2.2 e ESB (*Enterprise Service Bus*) na sub-secção 3.2.3. Terminamos esta secção com uma comparação entre SOA e *Silo Architecture*, na sub-secção 3.2.4.

De seguida, introduziremos a tecnologia BPEL (*Business Process Execution Language*) e JBI (*Java Business Integration*), nas secções 3.3 e 3.4, respectivamente, para um melhor entendimento das plataformas de integração, monitorização e orquestração utilizadas na JUP – o OpenESB e JCAPS, explicadas nas duas secções seguintes (3.5 e 3.6).

Posteriormente, na secção 3.7, apresentaremos a tecnologia *Java Enterprise Edition* (Java EE), enfatizando o modelo multi-camada para a construção de aplicações SOA. Nesta secção especificaremos os seus componentes principais: o JSF (*Java Server Faces*), na secção 3.7.1, e EJB (*Enterprise Java Beans*), na secção 3.7.2.

De seguida, abordaremos a linguagem de modelação utilizada neste projecto, o UML, com a descrição dos diagramas utilizados (na secção 3.8).

Por fim, apresentaremos uma tecnologia para automatizar o desenvolvimento de software, o Apache ANT, exposta na secção 3.9.

3.1 Sistemas Distribuídos e Integração de Sistemas

Existem várias definições literárias para sistemas distribuídos, e nenhuma delas é completamente satisfatória ou está de acordo com qualquer uma das outras. Segundo Tanenbaum (1994), a melhor forma de caracterizar um sistema distribuído é a seguinte:

"A distributed system is a collection of independent computers that appear to the users of the system as a single computer"

Esta definição sugere dois aspectos fundamentais. O primeiro relacionado com o hardware (a autonomia das máquinas) e o segundo com o software (os utilizadores pensam no sistema como um computador individual).

Um exemplo simples de um sistema distribuído pode ser um sistema bancário. Imaginemos um grande banco com centenas de filiais em todo o mundo. Cada uma tem um servidor que guarda as contas locais, assim como todas as transacções. Cada computador está apto para comunicar com todos os outros computadores e com o servidor. Se uma transacção pode ser feita sem permissão para um cliente, e os utilizadores não notificam qualquer diferença entre este sistema e o sistema central, este faz a sua actualização. Podemos ter então um sistema cujas bases de dados estão distribuídas por todos e quando se verifica uma alteração, essa informação é automaticamente partilhada pelos utilizadores do sistema. (Tanenbaum, 1994)

A integração de sistemas é uma aproximação estratégica para ligar vários módulos/sistemas de informação, permitindo trocas de informação suportando eventos comuns, de modo a obter um sistema final funcional. (Linthicum, 2004)

O conceito de integração de sistemas, ou AI *(Application Integration)*, pode ser interpretado de várias formas. Uma delas pode interpretar o conceito como uma referência à integração de aplicações umas dentro das outras. Outro seria estabelecer a integração de sistemas apenas por transferência/partilha de dados entre aplicações, ou seja, uma aplicação envia os dados e a outra recebe-os, havendo interacção entre os sistemas. (Linthicum, 2001)

Alguns exemplos de tecnologias associadas a integração de sistemas podem ser *Web Services*, SOAP, XML, SOA, *message brokers, hub and spoke, enterprise*

bus, ETL, EAI, RPC, RMI, *workflow*, monitor de transacções, entre outros. (Cunha, 2007)

Partindo destes dois conceitos (distribuição e integração), salta à vista uma característica bastante importante deste tipo de sistemas – a interoperabilidade. Os sistemas distribuídos oferecem-nos estruturas para partilha de métodos provenientes de locais distintos para serem processados através de aplicações e também pela interoperabilidade entre processos, isto é, o sistema oferece mecanismos normalizados para aceder aos objectos partilhados, objectos que são executados em vários servidores. (Linthicum, 2004)

Portanto, a interoperabilidade está associada à particularidade de um sistema conseguir comunicar de uma forma perceptível e funcional com outro sistema. Para que isto aconteça facilmente, a existência de mecanismos normalizados é essencial, independentemente do tipo de sistema.

Num sistema distribuído a forma como é apresentada a informação é bastante importante, pois o objectivo fulcral é a transmissão de dados de uma organização para outra. Apresentemos então alguns modelos de sistemas distribuídos.

3.1.1 Bases de Dados Distribuídas

Imaginemos um conjunto de sistemas diferentes, cada um com uma base de dados. Uma vez tratando-se de um sistema distribuído, este conjunto de base de

dados diferentes formariam apenas uma base de dados (remota) que permitiria aceder à informação de qualquer um dos fragmentos.

Assim, de forma transparente e aberta, qualquer utilizador poderia consultar a informação registada num outro sistema integrado.

Um dos problemas principais desta tecnologia é o facto de a inserção/actualização de dados na base de dados remota ser um processo que pode causar muita lentidão e uma baixa performance no seu acesso. (Cunha, 2007)

3.1.2 Processos Distribuídos

Por vezes, há situações em que a troca de informação é demasiado reduzida, o que não justifica a utilização de mecanismos complexos para que isso aconteça. Quando temos uma situação de *request/reply*, por exemplo, transacções curtas, acabamos por conseguir uma melhor performance se utilizarmos este tipo de processos do sistema distribuído, que pode ser um *web service*. Exemplo: RPC. (Cunha, 2007)

3.1.3 Troca de Mensagens

Este paradigma destina-se a situações menos reconhecidas em que há troca de informação a nível da lógica aplicacional, isto é, informação que as aplicações conhecem e decidem enviar ou receber de outros actores.

A integração neste mecanismo tem a característica interessante de ser auditável e controlável. Exemplo: EDI/XML. (Cunha, 2007)

3.2 SOA (*Service-Oriented Architecture*)

O termo SOA tem origem em 1994, com o antigo analista da Gartner, Alexander Pasik, período em que ainda não existia XML nem *Web Services*. (Josuttis, 2007)

Com base no modelo cliente/servidor e no software que poderia envolver cada uma destas partes, Pasik realçou *server orientation* estimulando o desenvolvimento de aplicações SOA e, juntando a necessidade, cada vez mais viva, das empresas integrarem os seus sistemas com outros sistemas de outras entidades, o SOA tornou-se um conceito futurista no desenvolvimento de software. (Josuttis, 2007)

A grande vantagem de uma arquitectura orientada a serviços parte da redução da redundância na construção de sistemas aplicacionais, graças à reutilização de serviços, que serão referidos muitas vezes como *Web Services*. O SOA permite às organizações o desenvolvimento de aplicações tendo como enfoque os processos de negócio, fazendo uso de serviços já existentes e orquestrando-as tendo em vista as necessidades de cada aplicação. (Erl, 2005)

O SOA facilita a interacção entre serviços e permite tirar melhor partido das capacidades de distribuição que estes oferecem. Outro ponto importante que determina este conceito refere-se ao controlo destas capacidades de distribuição

poder ser feito por diferentes domínios de diferentes proprietários, ou seja, podemos ter sistemas complexos onde diferentes equipas, departamentos ou organizações interagem simultaneamente, utilizando diferentes plataformas também, e cada uma destas pode modificar os ambientes em que cada um se insere sem causar transtorno para as outras partes. (Josuttis, 2007; Erl, 2005)

Do ponto referido anteriormente aproximamo-nos de uma outra característica do SOA – suporte da heterogeneidade. Em sistemas com um grau de complexidade elevado podemos encontrar diferentes plataformas, diferentes linguagens e paradigmas de programação, e por vezes, diferente *middleware*, isto é, sistemas heterogéneos. (Josuttis, 2007)

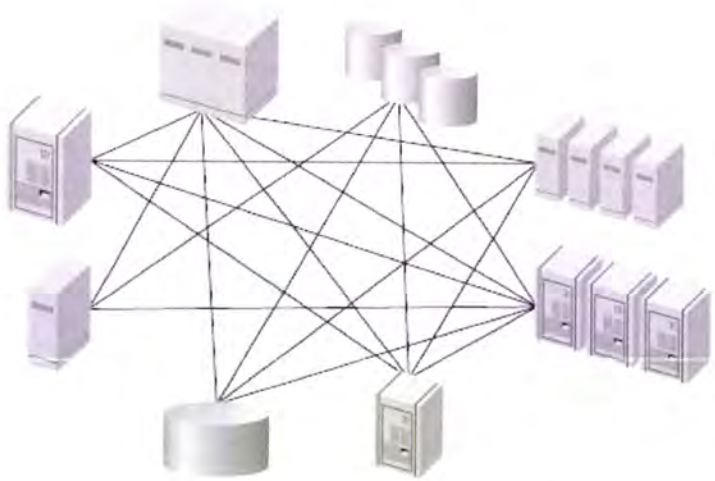


Ilustração 4 – Representação de um possível cenário num sistema distribuído (heterogeneidade). (Fonte: Josuttis, 2007)

Com sistemas heterogéneos e a capacidade de poder ligá-los facilmente, o SOA permite um grande avanço no que toca à interoperabilidade de sistemas,

estendendo o conceito de *Enterprise Application Integration* (EAI). (Josuttis, 2007)

3.2.1 SOA e Web Services

À partida deduzimos uma forte ligação entre SOA e serviços. Um serviço, numa perspectiva de negócio, consiste numa funcionalidade tecnológica autónoma, associada a actividades de negócio. Tecnicamente, um serviço é uma interface de mensagens que podem ser trocadas entre fornecedores e clientes.

Cada serviço possui um contexto de execução próprio e é responsável pela execução de um conjunto bem estanque de funções. (Newcomer e Lomow, 2005)

Vejamos como Natis (2003) mostra a importante relação entre SOA e *Web Services*:

Although Web Services do not necessarily translate to SOA, and not all SOA is based on Web Services, the relationship between the two technology directions is important and they are mutually influential: Web Services momentum will bring SOA to mainstream users, and the best-practice architecture of SOA will help make Web Services initiatives successful. **(Natis, 2003)**

Segundo Newcomer e Lomow (2005), o facto dos *Web services* serem difusos (no sentido em que possuem a capacidade de poderem estar presentes, ou "espalhados", por diferentes aplicações), simples e autónomos contribuem vantajosamente para a implementação de aplicações SOA. A arquitectura básica de um *Web service* (ver ilustração 5) consiste em especificações (WSDL, SOAP e UDDI) que suportam a interacção entre *requester* (cliente) e *provider* (fornecedor) e o potencial conteúdo do *Web service*.

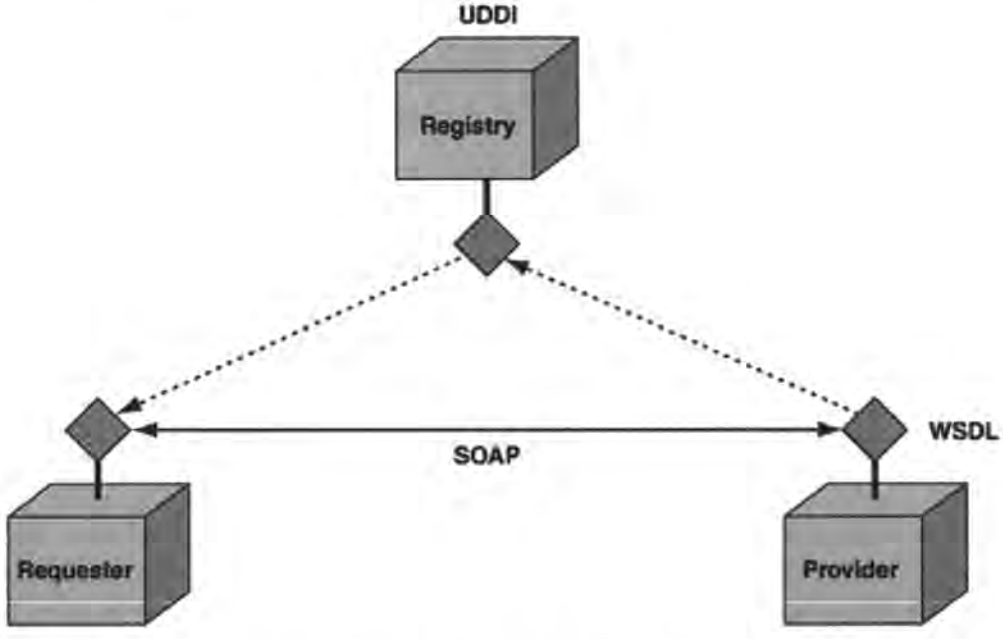


Ilustração 5 – Arquitectura básica de um Web service. (Fonte: Newcomer e Lomow, 2005)

O *provider* tipicamente publica a descrição do WSDL do seu *Web service* e o *requester* acede a esta descrição através da UDDI, ou de outro tipo de registo, pedindo a execução do serviço com o envio de uma mensagem SOAP. (Newcomer e Lomow, 2005)

Para um melhor entendimento desta arquitectura, passemos a especificar SOAP, UDDI e WSDL.

- **SOAP (*Simple Object Access Protocol*):** Protocolo para troca de informação num ambiente distribuído. Pedidos de clientes e as respostas de um *Web Service* são transmitidos como mensagens SOAP por HTTP, permitindo uma troca perfeitamente interoperável entre clientes e *Web Services*, executados em plataformas e locais diferentes na Internet. As mensagens SOAP definem, com base em XML, a descrição da mensagem e a forma como processá-la, inclui regras de codificação (em XML também) para indicar instâncias de tipos de dados dentro da mensagem e, uma convenção para representar os pedidos remotos e a resposta resultante. (Gudgin et al., 2007; Box et al., 2000)

- **WSDL (*Web Services Description Language*):** Descrição de um serviço de rede no formato XML (nome/localização do serviço e meios possíveis para comunicar com este). Esta descrição pode ser guardada em registos UDDI ou publicados na Web. (Hansen, 2007)

- **UDDI (*Universal Description Discovery and Integration*):** Formato XML standard que permite publicar informação de negócio na Internet sobre produtos e *Web Services*, onde a informação pode ser prontamente e globalmente acedida pelos clientes. (Hansen, 2007)

Vejamos de seguida uma ilustração de uma plataforma de *Web Services* com as componentes básicas e estendidas necessárias para dar suporte a uma

arquitectura orientada a serviços, através uma *Enterprise Service Bus* (ESB) para ligar os serviços. (Newcomer e Lomow, 2005)

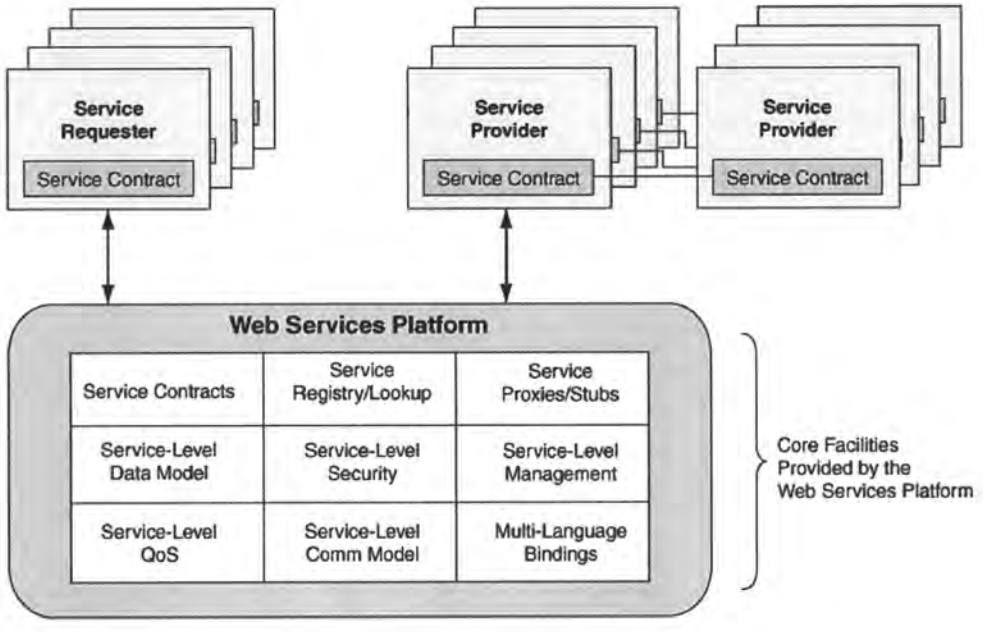


Ilustração 6 – Plataforma de *Web Services*. (Fonte: Newcomer e Lomow, 2005)

Existem várias versões diferentes de *Web Services* especificados por diferentes organizações, o que pode tornar a interoperabilidade num problema. No entanto, a *Web Services Interoperability Organization* (WS-I) tenta resolver este pormenor pela promoção de diferentes perfis para determinados conjuntos de standards. (Erl, 2004)

3.2.2 SOA e BPM (*Business Process Manager*)

Numa arquitectura orientada a serviços, os serviços são tipicamente partes de um ou mais processos de negócio distribuídos. Estes serviços são depois executados sobre uma certa ordem para atingir um determinado fim. Esta sequência de execução de serviços forma um fluxo, o qual pode ser representado sobre a forma de um BPM (*Business Process Manager*).

BPM, consiste numa tecnologia que permite conhecer, visualizar e controlar os processos de negócio de uma ou mais organizações e, a sua relação com o SOA é bastante evidente, na medida em que as actividades de baixo nível de um processo são serviços. Do ponto de vista do negócio, não interessa quando é que os serviços são básicos ou complexos, mas sim que os serviços oferecem as funções de negócio necessárias. (Josuttis, 2007)

3.2.3 SOA e ESB (*Enterprise Service Bus*)

Uma parte do SOA corresponde à infra-estrutura que permite a comunicação entre serviços. Esta funcionalidade é conhecida por ESB, *Enterprise Service Bus*.

Além de permitir aos clientes chamar os serviços fornecidos, o ESB impulsiona a interoperabilidade uma vez que permite a integração de diferentes plataformas e linguagens de programação, onde um dos pontos fundamentais é a transformação de dados, isto é, a habilidade para transformar dados de um determinado formato para outro, permitindo assim que qualquer serviço seja

acessível e entendido por qualquer parte envolvida no sistema. Também assume um papel importante no que toca a *routing*, segurança, monitorização e orquestração (normalmente por BPEL), autenticação e fiabilidade. (Chappel, 2004; Kinnumpurath, 2005)

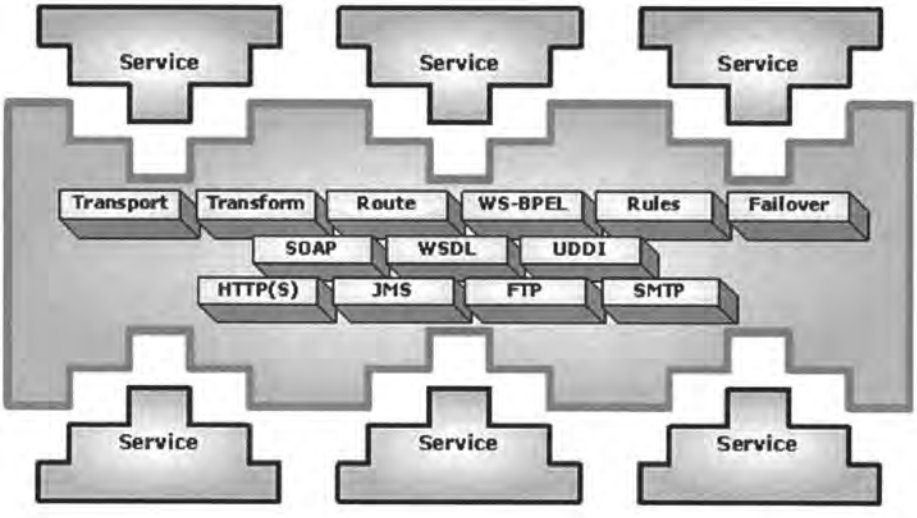


Ilustração 7 – Ambiente ESB. (Fonte: Kinnumpurath, 2005)

No mundo ESB, os serviços não interagem directamente entre si. No entanto, em tempo de execução, o ESB actua como um mediador entre os serviços, podendo implementar protocolos de ligação, tradução e manuseamento de mensagens, entre outras actividades. (Kinnumpurath, 2005)

3.2.4 *Service Oriented Architecture* versus *Silo Architecture*

Uma aplicação *silo* típica traça as suas funcionalidades sobre uma aplicação já existente. Basicamente, existe uma aplicação com uma interface que acede aos dados de negócio através de aplicações de *mainframe.* A lógica de negócio está distribuída ao longo de um *mainframe* e outros computadores descentralizados. A sua interface tende a ser volátil, o que força a que quem utiliza os serviços tenha que se adaptar a estes sempre que haja alterações. Isto pode causar algum caos, começado pela duplicação de sistemas, regras de negócio e dados. Na figura abaixo podemos ver a ilustração deste cenário. (Gévaudan, 2005)

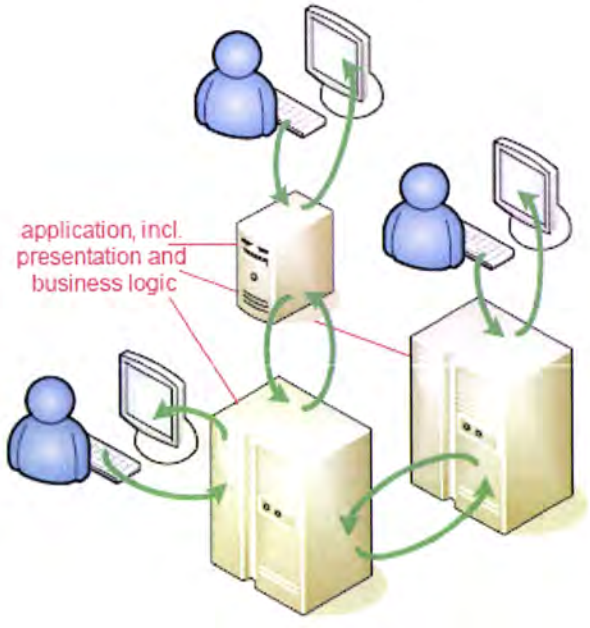


Ilustração 8 – Desenho da *Silo Architecture.*

Por outro lado, numa arquitectura orientada a serviços, as aplicações possuem apenas a apresentação da lógica de negócio. Existem serviços que normalmente oferecem funcionalidades de negócio sem apresentação. É a própria aplicação que controla os grupos de utilizadores e as sequências de negócio através de *business processes* (processos de negócio), reduzindo a complexidade e levando a um desenvolvimento e *deployment* mais rápido. (Gévaudan, 2005)

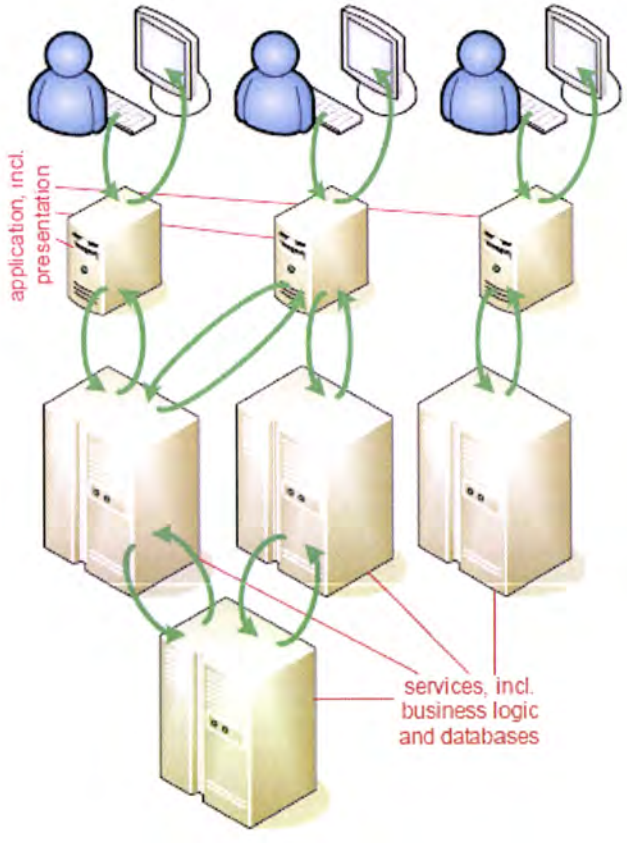


Ilustração 9 – Desenho da arquitectura orientada a serviços (SOA).

Numa arquitectura orientada a serviços, o ponto de interacção entre o cliente e fornecedor do serviço é definido por uma função de negócio, e não por aplicações entre componentes, como acontece na *silo architecture*. No SOA, um serviço fornece todo o conteúdo desta função de negócio (ou grupo de funções) e expõe uma interface aos seus clientes. Isto significa que a lógica de negócio pode ser modificada sem afectar os clientes. (Gévaudan, 2005)

Desta forma, as aplicações numa arquitectura orientada a serviços não contêm a lógica de negócio. Estas aplicações utilizam serviços, muito mais fáceis de implementar e com um grau de simplicidade superior, e este é um factor extremamente importante para quem desenvolve software, representando uma grande vantagem face aos custos elevados na manutenção de aplicações com uma arquitectura *silo*. (Gévaudan, 2005)

Na figura abaixo podemos ver um exemplo concreto do desenho da arquitectura de um sistema antes do SOA e depois do SOA:

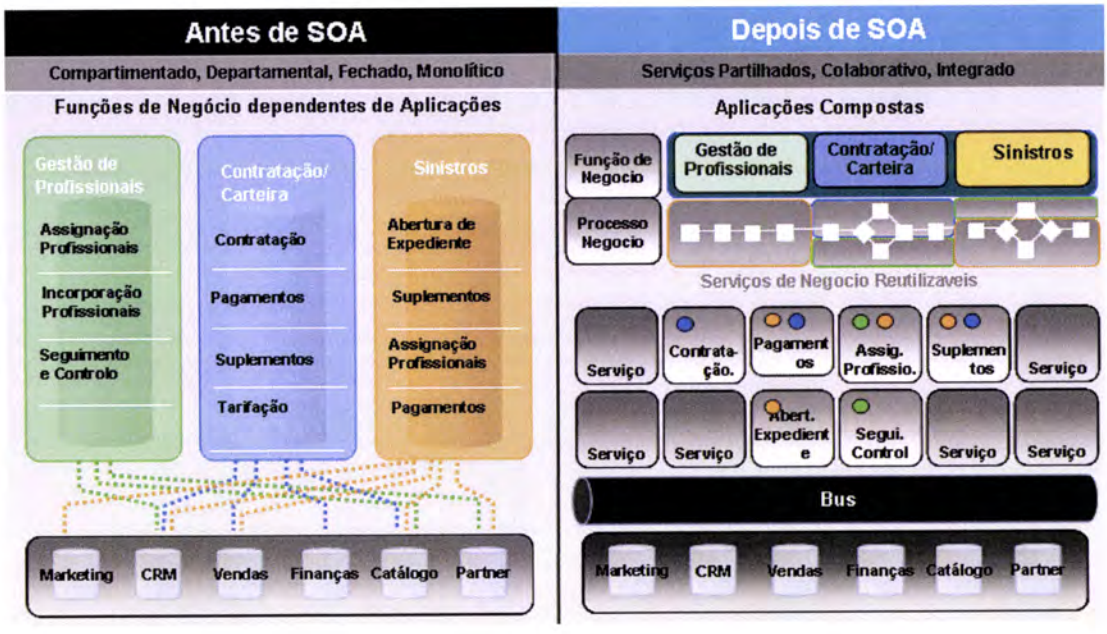


Ilustração 10 – Antes e depois de SOA.

Observando a imagem, podemos verificar que em arquitecturas tradicionais (antes do SOA), todos os componentes do sistema (actividades de processos, aplicações e dados) estão normalmente fechados em blocos independentes e incompatíveis. Como já referimos, esta característica implica uma grande dificuldade na manutenção destes sistemas, onde os utilizadores são obrigados a navegar em redes, aplicações e bases de dados separadas para conseguirem concretizar os seus objectivos. (Sun Microsystems, 2008b)

Com o surgimento do SOA, através de um serviço integrado, o utilizador consegue aceder a dados relevantes numa única aplicação e numa única sessão. (Sun Microsystems, 2008b)

3.3 BPEL (*Business Process Execution Language*)

Segundo Iyrngar et al. (2008), BPEL, também conhecido por WS-BPEL (*Web Service Business Process Execution Language*), consiste numa linguagem (baseada em XML) de execução com a finalidade de especificar processos de negócio e protocolos de interacção de negócio. Por Jordan e Evdemon (2006), esta interacção ocorre através de *Web Services*, e a estrutura do relacionamento a nível da interface é encapsulada num *partnerLink*.

Um processo BPEL define como múltiplas interacções entre serviços com os seus parceiros são coordenados para atingir objectivos de negócio, assim como o estado e toda a lógica necessária para esta coordenação. Por outro lado, introduz mecanismos sistemáticos para lidar com excepções de negócio e falhas de processamento, possuindo também um mecanismo para definir como é que actividades individuais ou compostas, inseridas numa unidade de trabalho, estão a ser compensadas em casos de ocorrência de excepções ou de solicitações reversas por parte dos parceiros. (Jordan e Evdemon, 2006)

BPEL é um processo reutilizável, com elevado nível de abstracção, que pode ser introduzido em diferentes cenários, enquanto mantém um comportamento uniforme a nível da aplicação para todos eles. Outro aspecto refere-se à diminuição considerável da complexidade. (Jordan e Evdemon, 2006; Salter e Jennings, 2008)

Segundo Salter e Jennings (2008), actualmente, a adopção do BPEL é o meio mais elegante para orquestrar *Web Services* inseridos num processo de negócio significativo. Vejamos esquematicamente onde se insere o BPEL:

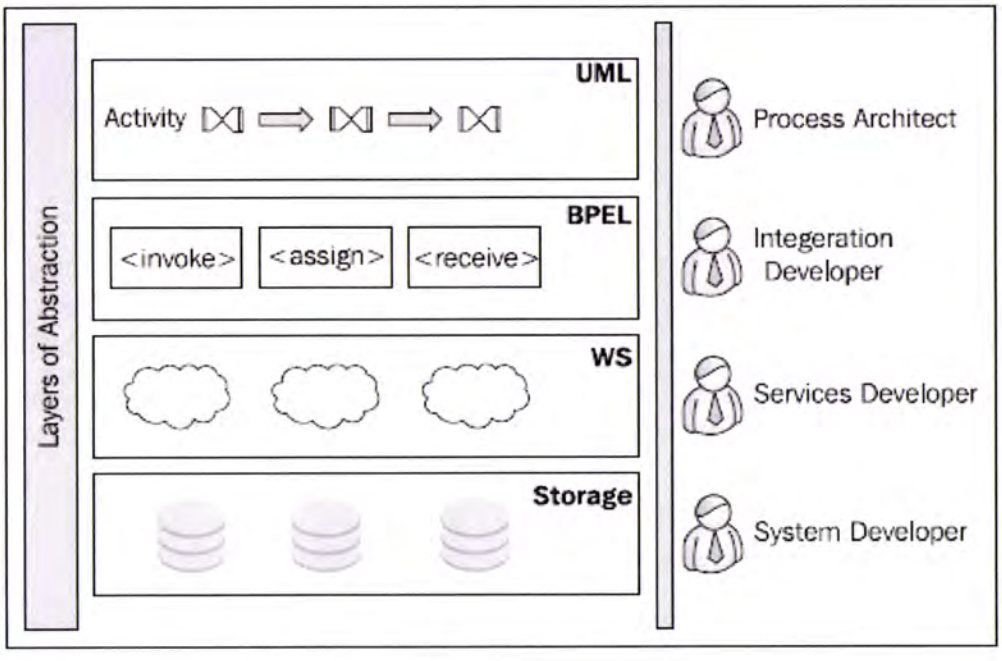


Ilustração 11 – Identificação do BPEL numa arquitectura aplicacional. (Fonte: Salter e Jennings, 2008)

Podemos verificar que na primeira camada a modelação de actividades e *workflow*, através de UML (ver secção 3.8), seguido dos nossos processos de integração, com a tecnologia BPEL e por fim, os *Web Services* (WS) que fazem a ponte entre os anteriores e as bases de dados (*Storage*).

3.4 JBI (*Java Business Integration*)

JBI é um standard do Java para estruturar a integração de sistemas de negócio. Define um ambiente para conectar componentes que interagem usando um modelo de serviços baseados em WSDL.

Na figura seguinte está representado o ambiente desta tecnologia, onde os rectângulos azuis representam as componentes de ligação, dentro do ambiente JBI. Estes componentes estão ligados a um *router* de mensagens, através dos chamados *delivery channels* (a amarelo), e a sua função tanto pode ser serviços por parte do fornecedor ou do cliente, ou ambos. (Ten-Hove, 2006)

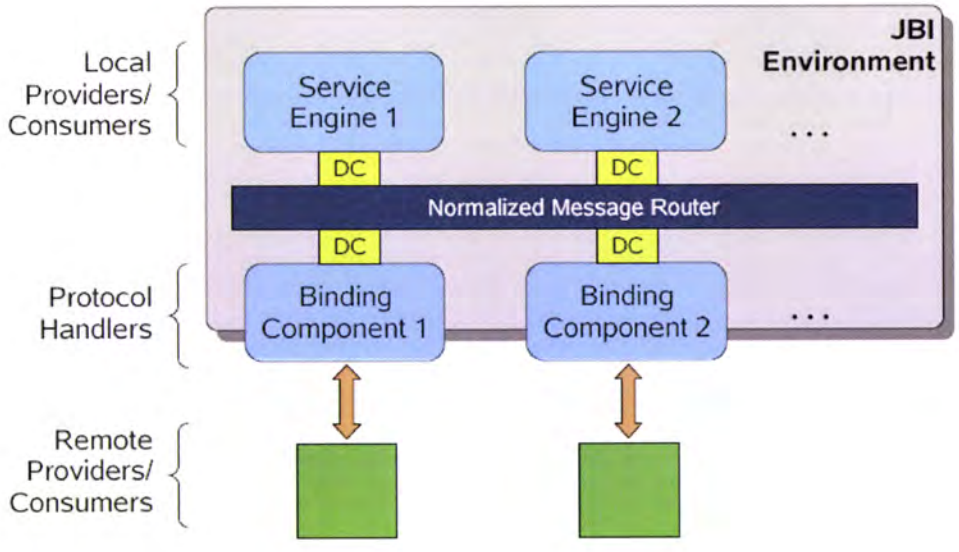


Ilustração 12 – Ambiente JBI. (Fonte: Ten-Hove, 2006)

Os componentes que fornecem ou recebem serviços localmente (dentro do ambiente JBI) são chamados de *service engines* e os componentes que fornecem

ou recebem serviços através de algum protocolo de comunicação ou outra tecnologia remota, são chamados *binding componentes* (em português, componentes de ligação). (Ten-Hove, 2006)

Os componentes JBI, uma vez instalados num ambiente JBI, interagem entre eles através de um processo de troca de mensagens, que é descrito por documentos WSDL, publicados pelo serviço fornecido. Estes descritores de serviço contêm informação necessária para que haja interacção entre serviços cliente e serviço fornecedor. O JBI possui uma infra-estrutura leve de mensagens, conhecida por *Normalized Message Router*, que fornece o mecanismo de troca de mensagens, garantindo interoperabilidade entre os componentes. (Raj et al., 2006; Ten-Hove, 2006)

3.5 OpenESB (*Open Enterprise Service Bus*)

Um dos exemplos de uma plataforma ESB para *Business Integration*, *Enterprise Application Integration* (EAI) e SOA é o OpenESB (*Open Enterprise Service Bus*). OpenESB é um projecto *open source* que inclui o padrão *Java Business Integration* (JBI) e diversos componentes e tecnologias desenhadas para maximizar a agilidade dos negócios e reduzir os custos de integração. Para além disto, conta com novas funcionalidades e ferramentas que possibilitam o desenvolvimento de aplicações compostas, ao mesmo tempo, em que impulsionam as aplicações e sistemas já existentes. A plataforma oferece aumento de interoperabilidade e integração com outros softwares,

nomeadamente o Java CAPS, que abordaremos no seguinte subcapítulo. (OpenESB, 2008)

Na figura seguinte podemos observar a arquitectura desta plataforma, com cada uma das suas componentes identificadas:

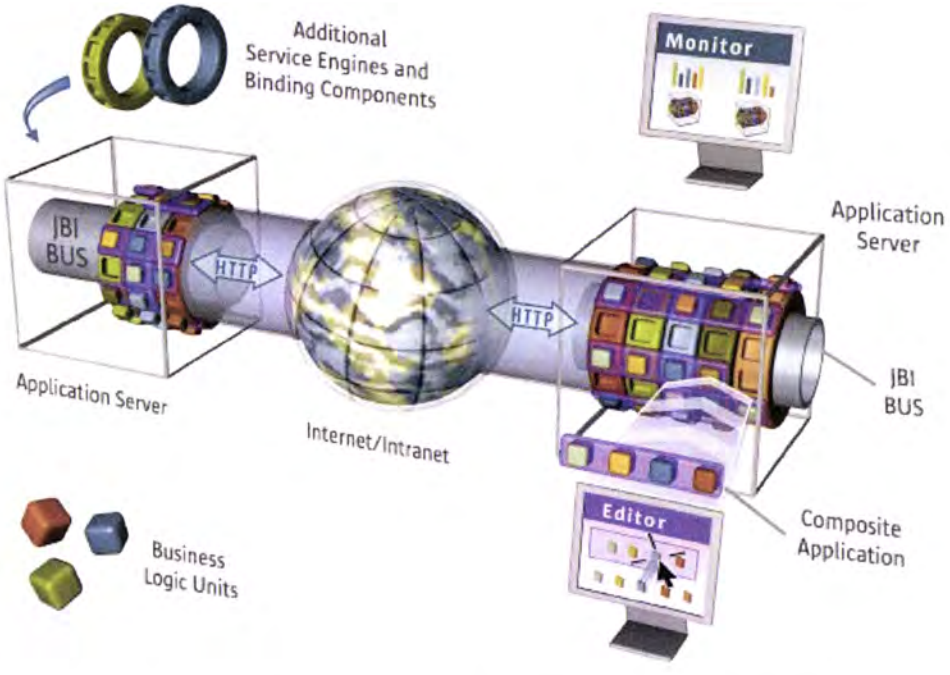


Ilustração 13 – Arquitectura do OpenESB. (Fonte: OpenESB, 2008)

Segundo OpenESB (2008):

- ***Application Server*:** Servidor aplicacional (GlassFish), que permite a fiabilidade, escalabilidade, resiliência, implantação (em inglês, *deployment*) e controlo de aplicações Java EE.

- ***Composite Application*:** Simples artefactos autónomos que contém sub-artefactos que podem ser serviços que compõem a lógica de negócio num serviço global. Uma *Composite Application* permite-nos criar serviços de integração ou orquestração com uma nova lógica de negócio, sem ter que interferir nos serviços já existentes, comunicando com outras aplicações através de protocolos HTTP, SOAP, REST, JMS, FTP, entre tantos outros. A orquestração de serviços é feita através de BPEL.

- **JBI Bus:** Alberga uma série de componentes de ligação que integram todo o ambiente de negócio. Este é o grande responsável pela interoperabilidade do sistema. Quando o envio/recepção de mensagens ocorre fora de um ambiente JBI, a comunicação é feita através do *Normalized Message Router* (NMR), uma infra-estrutura de mensagens, e passadas para o cliente através de um componente de ligação apropriada. No caso de ocorrer num ambiente JBI, não é necessária qualquer conversão, serialização ou normalização, pois todas as mensagens estão normalizadas e num formato standard, chamado WSDL.

- ***Service Engines and Binding Components*:** *Services Engines* fornecem a lógica de negócio e serviços de transformação a outros componentes. Os *Binding Components* fornecem serviços externos ao ambiente OpenESB, permitindo envolver outros protocolos de comunicação ou serviços.

- ***Business Logic Units:*** Elementos que implementam certos aspectos dos serviços globais. Por exemplo, um processo de negócio BPEL criado para implementar uma transacção *e-commerce,* a partir de serviços internos ou externos.

3.6 Java CAPS (*Java Composite Aplication Platform Suite*)

A plataforma escolhida para implementar o conceito SOA neste projecto é o *Java Composite Application Platform Suite* (Java CAPS, ou apenas JCAPS) da Sun Microsystems. Esta ferramenta está preparada para endereçar projectos complexos, onde a diversidade de sistemas e aplicações e das respectivas interfaces seja grande e em que se pretenda encapsular rapidamente essas aplicações como *Web Services* compostos e orquestrados. (Sun Microsystems, 2008)

Sendo baseada no conceito SOA e *Enterprise Service Bus* (ESB), o JCAPS disponibiliza ferramentas para o rápido desenvolvimento de novos serviços compostos (*Composite Applications*), integração baseada em *Web Services*, serviços de *Business Process Management* (BPM) e standards abertos. Além disto, os seus módulos são totalmente interoperáveis e fornecem o mais vasto conjunto de funcionalidades de integração e reutilização, transformação de dados (ETL) e um completíssimo conjunto de conectores de serviços e aplicações. (Sun Microsystems, 2008)

Esta suite inclui ainda funcionalidades como *Business Activity Monitoring* (BAM) para a criação de sofisticados *dashboards* e sistemas de alerta, serviços de integração B2B e implementa o conceito de Vista Global de Clientes e Recursos (*Master Customer Index*). (Sun Microsystems, 2008)

Segundo Sun Microsystems (2008a), a última versão JCAPS 6 possui os seguintes componentes principais:

- *Sun Java ESB Suite*: Baseada no OpenESB (abordado no capítulo anterior) e JBI, a Sun Java ESB Suite é uma plataforma modular e aberta que permite a rápida criação de aplicações e a interligação a vários componentes e protocolos para maior flexibilidade SOA.
- *Intelligent Event Processor (IEP)*: Oferece a capacidade para identificar ameaças em tempo real, permitindo às organizações responder a negócios críticos e tomar acções correctivas de forma pro-activa.
- *Business Process Management*: Suporta BPEL 2, inclui novas características de *failover* e alta disponibilidade, e fornece processos de monitorização através da integração com IEP.
- *GlassFish and NetBeans (IDE) Software Support*: Inclui servidor GlassFish, utilizando uma interoperabilidade líder de indústria com .NET e WSIT/Project Metro bem como o software unificado NetBeans IDE para maior controlo.

Na figura abaixo podemos visualizar todos os componentes que compõem o JAVA CAPS:

Ilustração 14 – Resumo dos componentes Java CAPS. (Fonte: Sun Microsystems, 2008)

3.7 Java EE (*Java Enterprise Edition*)

Java EE ou J2EE é uma plataforma Java vocacionada para aplicações multi-camada empresariais e que sejam portáveis, escaláveis, robustas e seguras. Esta plataforma oferece um poderoso conjunto de APIs que proporciona aos criadores de software a redução de complexidade e aumento de performance no seu trabalho. (Ball et al., 2006)

Na figura seguinte podemos verificar as 3 camadas que constituem uma aplicação Java EE:

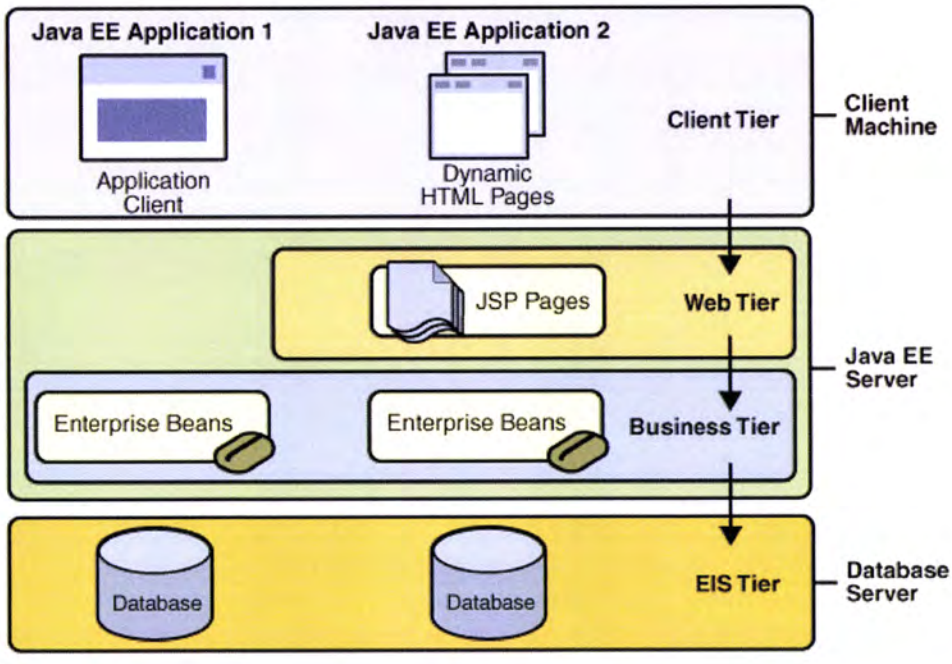


Ilustração 15 – Aplicações multi-camada. (Fonte: Ball et al., 2006)

Arquitecturas orientadas a serviços (SOA) envolvem a implementação de aplicações compostas capazes de oferecer transparência e segregação da lógica de negócio. Com base no que já mencionamos (e no que ainda iremos abordar) sobre Java EE, facilmente entendemos que esta plataforma encaixa perfeitamente no conceito SOA. Por exemplo, esta plataforma proporciona meios para implementar, facilmente, *Web Services* (JAX-WS, Java API for

XML *Web Services*) usando protocolos HTTP/SOAP, como é definido pela especificação WS-I. (Raj et al., 2006)

Passemos a abordar cada um dos componentes principais da plataforma Java EE.

3.7.1 JSF

O *Java Server Faces* é uma *framework* para o desenvolvimento de aplicações *Web*, tipicamente integrados numa aplicação Java EE.

Com base em componentes predefinidos, esta tecnologia permite estender estes à medida do utilizador. Temos o exemplo do projecto ICEfaces (http://www.icefaces.org) ou do MyFaces (http://myfaces.apache.org), projectos que disponibilizam *frameworks* com componentes estendidos. Estes componentes são acedidos através de *tags* JSP (Java Server Pages). No nosso projecto, este foi um dos aspectos mais explorados a nível da camada de apresentação da aplicação.

Por detrás desta apresentação, encontram-se os *Backing Beans* que definem um conjunto de propriedades e funções próprias, interferindo no comportamento de cada componente inserido numa página JSP. É através destes que é feita a comunicação com as restantes camadas de uma aplicação Java EE. Para além da vantajosa ligação a outras camadas de uma aplicação Java EE, é possível guardar os estados dos componentes JSF, o que simplifica por exemplo manter o valor de um determinado campo. (Mann, 2005; Ball et al., 2006)

Durante a execução e após a recepção do pedido do cliente para mostrar uma determinada página, o conjunto de componentes, em simultâneo com as propriedades definidas nos *Backing Beans*, são mostrados, dando origem a uma página HTML. (Mann, 2005)

Toda esta actividade processa-se num *Application Server*, por exemplo o Glassfish (https://glassfish.dev.java.net/), que fornece um conjunto de funcionalidades de suporte a outras camadas de uma aplicação Java EE.

Na figura abaixo podemos ver, resumidamente, o que acontece no lado do cliente e servidor, quando utilizamos esta tecnologia. Temos as nossas páginas JSP que correspondem à apresentação da nossa aplicação, a parte do modelo (que podem ser EJBs) e um controlador implementado através de um *Servlet*. Tudo isto do lado do servidor. Este esquema permite que quando ocorra um erro numa das partes, as outras não sejam afectadas. Outra vantagem é a implementação de cada uma delas ser independente, podendo ser integradas ao longo do tempo. (Mann, 2005)

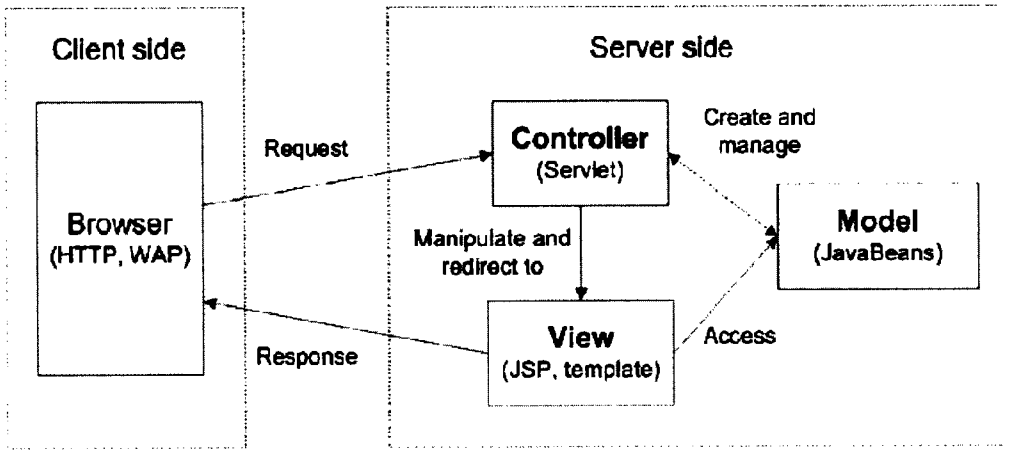


Ilustração 16 – Comportamento JSF. (Fonte: Mann, 2005)

3.7.2 EJB (*Enterprise Java Beans*)

Enterprise Java Beans (EJB) consiste numa plataforma para construção de aplicações portáveis, reutilizáveis e escaláveis através da linguagem de programação Java. É uma componente que actua no lado do servidor e que tem o papel fundamental de encapsular a lógica de negócio numa aplicação, servindo de intermediário entre a camada de apresentação e a camada de persistência. (Panda et al., 2007; Ball et al., 2006)

Este componente vem simplificar, e muito, o desenvolvimento de aplicações distribuídas complexas. Primeiro, porque o EJB *container* oferece aos *enterprise beans* serviços a nível do sistema para o controlo de transacções, segurança e autorização. Segundo, porque facilita o trabalho do programador para que ele não tenha de se preocupar com aspectos de infra-estrutura e apenas

tenha de se concentrar em resolver problemas de negócio. (Panda et al., 2007; Ball et al., 2006)

Para além de ser um suporte à camada de persistência, processar e controlar as transacções, outros serviços referem-se ao controlo de eventos através de JMS (*Java Message Service*), chamadas a procedimentos remotos via RMI, serviços JNDI (*Java Naming and Directory Interface*), serviços de segurança – através de JCE (*Java Cryptography Extension*) e JAAS (*Java Authentication and Authorization Service*) – e exposição de métodos de negócio em forma de *Web Services*. (Panda et al., 2007)

Segundo Panda et al. (2007), a especificação EJB contempla 3 tipos de *beans*, são eles os *Session Beans*, os *Message Driven Bean* (MDB) e as *Entities*.

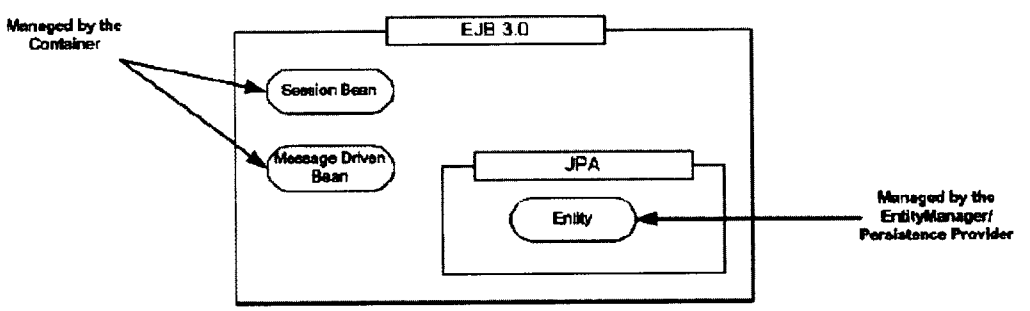


Ilustração 17 – Organização do EJB 3.0 API. (Fonte: Panda e tal., 2005)

3.7.2.1 Session Beans

Tipicamente é criado por um cliente, estando a sua existência dependente da duração da sessão que o cliente originou. Existem dois tipos de *session beans*:

stateful e *stateless*. O primeiro automaticamente guarda o estado do *bean* entre cada invocação realizada pelo cliente, sem que este tenha que escrever código adicional. Por outro lado, o *Stateless Session Bean* não guarda o estado e os serviços da aplicação podem ser completados apenas numa simples invocação por parte do cliente. (Panda et al., 2007)

3.7.2.2 Message Driven Beans (MDB)

São objectos distribuídos que permitem a execução de funções de uma forma assíncrona. Ao contrário dos anteriores, o cliente nunca invoca directamente métodos destes *beans*. MDBs são desencadeados por mensagens enviadas a um servidor de mensagens, o qual permite a troca de mensagens assíncronas entre as componentes do sistema. Estes *beans* foram desenhados com vista à sua utilização com o JMS (*Java Messaging System*). (Panda et al., 2007)

3.7.2.3 Entities e JPA (Java Persistence API)

A persistência em EJB é gerida pela JPA, *Java Persistence API*. Esta *framework* fornece um conjunto de métodos que facilitam a ligação entre a aplicação em construção e o SGBD, garantindo a persistência dos dados. (Panda et al., 2007)

A base do JPA é o *Object Relational Mapping* (ORM), que consiste no mapeamento dos dados em objectos Java para a base de dados relacional. Tipicamente nestes objectos que mapeiam elementos do modelo relacional

(particularmente tabelas são classes simples de Java) as suas instâncias correspondem a entradas da tabela mapeada. (Panda et al., 2007)

A partir do mapeamento acima referido é possível efectuar operações sobre a informação presente na base de dados, tendo a garantia que as operações são mais tarde persistidas na base de dados. (Panda et al., 2007)

3.8 UML (*Unified Modeling Language*)

O UML, *Unified Modeling Language*, é uma linguagem para especificação, visualização e documentação dos componentes de uma aplicação de software, sendo independente da linguagem e na maioria dos casos independente da plataforma. (Larman, 2002; OMG, 2007)

Hoje em dia a sua utilização tornou-se comum na modelação da maioria dos projectos envolvendo software. A sua estrutura apesar de bastante extensível e genérica permite que seja facilmente entendida pelos programadores, direccionando-os para uma maior preocupação e dedicação na arquitectura e no desenho do sistema. (Larman, 2002; OMG, 2007)

Embora exista uma gama bastante alargada de diagramas, foram usados no projecto apenas os diagramas de Caso de Uso e respectiva descrição, Diagramas de Actividades, Digramas de estados, Diagramas de Sequência e ainda Diagramas de Classes.

3.8.1 Diagramas de Casos de Uso

Os diagramas de casos de uso são usados, tipicamente para definir os requisitos do sistema. Os conceitos chave associados a estes diagramas definem-se como os actores, os casos de uso e o assunto. Este assunto é a parte do sistema a que o caso de uso se aplica. Os utilizadores que interagem com o sistema são definidos como actores, estes são entidades que permanecem exteriores ao sistema. As actividades que o actor pode realizar com o sistema são denominadas de casos de uso (*use case*). Uma instância de um caso de uso corresponde a uma ocorrência de um comportamento que corresponde a um tipo de caso de uso. (Larman, 2002; OMG, 2007)

Na figura abaixo está representado um diagrama de casos de uso, que representa o acesso à JUP.

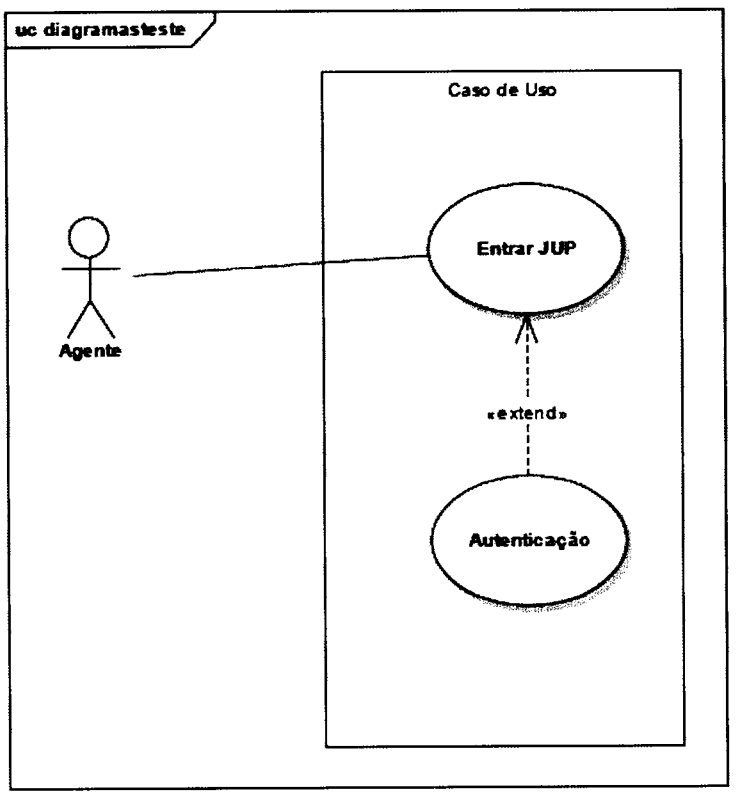


Ilustração 18 – Exemplo de Diagrama de Caso de Uso.

3.8.2 Diagramas de Classes

Os Diagramas de Classes são uma representação da estrutura e respectiva relação de correspondência, caso se trate de uma linguagem orientada a objectos. Estes diagramas têm relevante importância para a construção de outros como é o caso dos diagramas de sequência. (Larman, 2002; OMG, 2007)

As classes contêm atributos bem como funções e as ligações entre si são denominadas de associações. (Larman, 2002)

A próxima figura apresenta um exemplo de um diagrama de classes onde um despacho é composto por 0 ou mais avaliações, tem uma Autoridade associada e estende um documento.

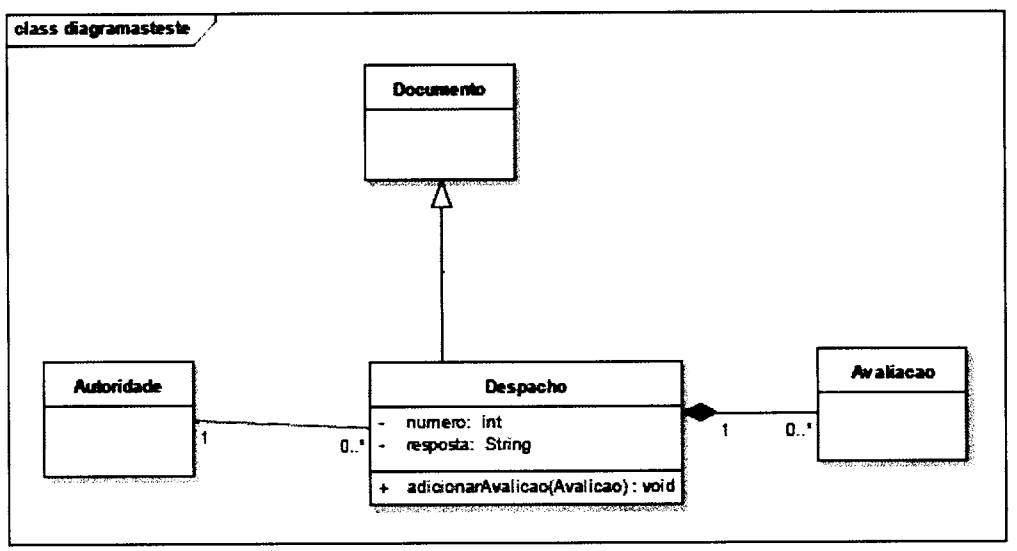


Ilustração 19 – Exemplo de Diagrama de Classes.

3.8.3 Diagrama de Estados

Um diagrama de estados reflecte um conjunto de conceitos que podem ser usados para modelar um comportamento através de um conjunto finito de transições entre estados. (Larman, 2002)

Está patente na figura abaixo um exemplo de um diagrama de estados.

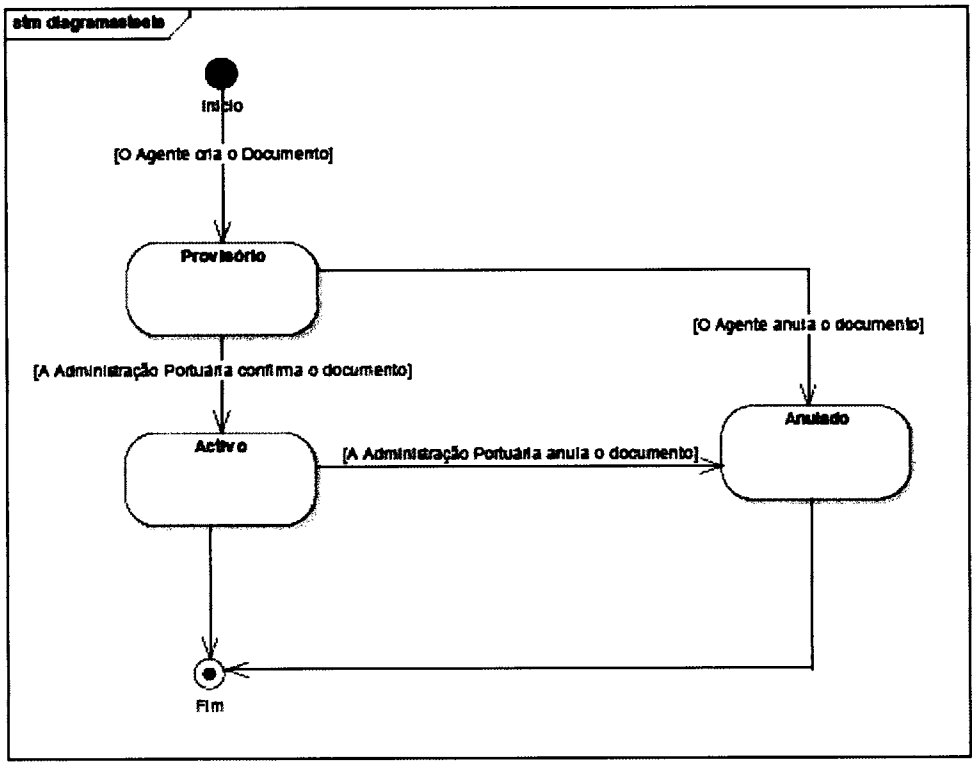


Ilustração 20 – Exemplo de Diagrama de Estados.

3.8.4 Diagrama de Actividades

O Diagrama de Actividades relata a sequência e as condições com que estas são executadas dentro do sistema. (Larman, 2002)

Podemos ver um exemplo na seguinte figura.

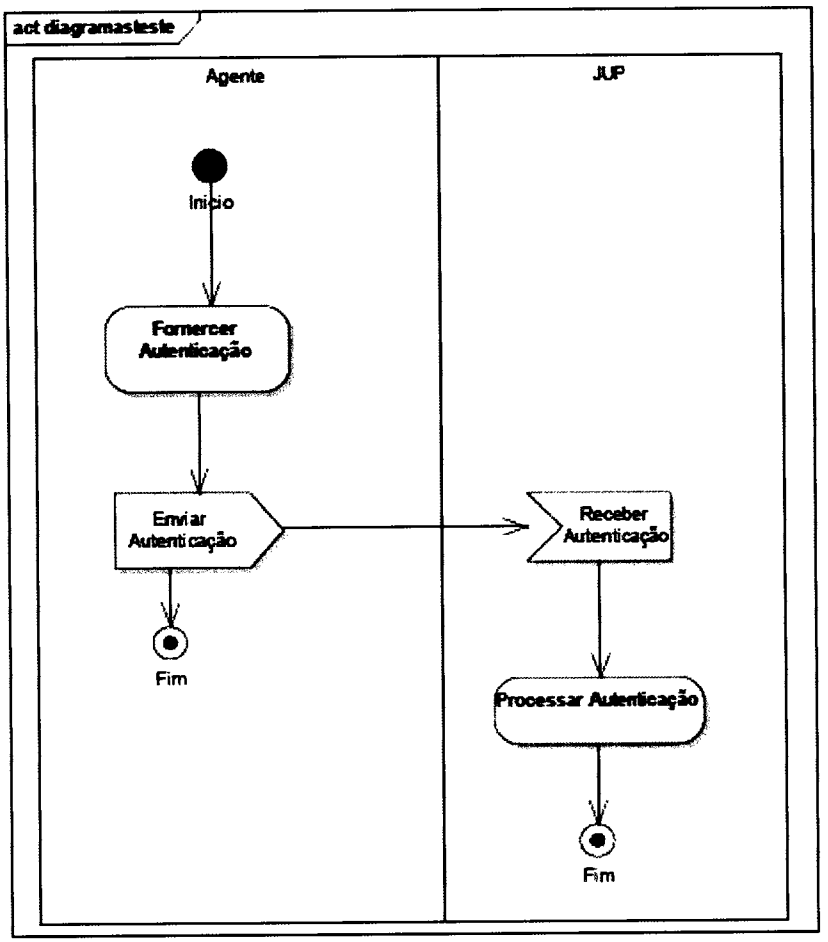


Ilustração 21 – Exemplo de Diagrama de Actividades.

3.8.5 Diagramas de Sequência

Um diagrama de sequência mostra em linhas verticais, os diferentes processos ou objectos que funcionam em simultâneo. As mensagens trocadas entre estes e a ordem em que elas ocorrem, são representadas pelas linhas horizontais. (Larman, 2002)

O exemplo seguinte pretende mostrar a sequência de um potencial sistema de autenticação.

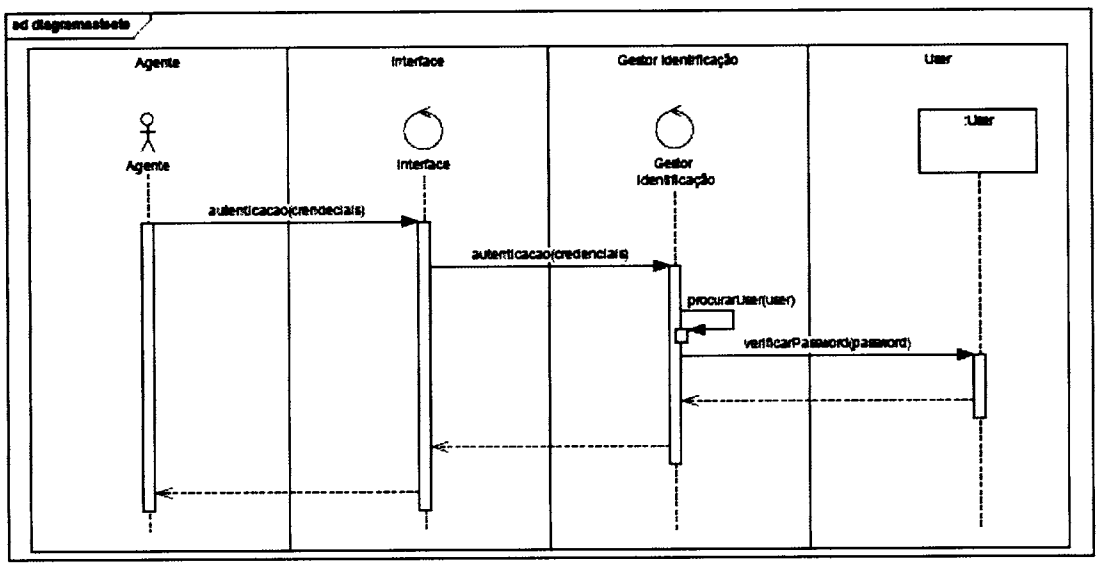


Ilustração 22 – Exemplo de um Diagrama de Sequências.

3.9 Apache ANT

Segundo Bailliez, S. et al. (2004), *Apache Ant* é uma ferramenta baseada em Java utilizada para automatizar o desenvolvimento de software e, teoricamente, é comparado a um género de *Make*. No entanto, o *Apache Ant* apresenta uma grande vantagem que se refere à portabilidade, permitindo-nos desenvolver software independentemente da plataforma. Outra vantagem é a facilidade que oferece a nível de configurações, uma vez que esta é feita em ficheiros baseados na linguagem XML, chamando uma árvore de *targets* onde várias tarefas são executadas, cada uma por um objecto que implementa uma interface particular.

Apache Ant foi a tecnologia utilizada para automatizar o processo de desenvolvimento do software no projecto da JUP.

3.10 Síntese

No presente capítulo abordámos uma série de conceitos e tecnologias utilizados neste projecto. Iniciámos com uma abordagem geral sobre sistemas distribuídos e integração de sistemas, onde especificámos as bases de dados distribuídas, processos distribuídos e sistemas de troca de mensagens. Após esta introdução, penetrámos no SOA (*Service-Oriented Architecture*), estabelecendo as ligações deste paradigma com outras tecnologias, nomeadamente os *Web Services*, BPM (*Business Process Manager*) e ESB (*Enterprise Service Bus*). Ficámos também a entender as vantagens principais de uma arquitectura orientada a serviços face às arquitecturas tradicionais. Partindo deste conhecimento aprofundamos os componentes principais que caracterizam as tecnologias/ferramentas para construção de sistemas SOA: a tecnologia BPEL (*Business Process Execution Language*), JBI (*Java Business Integration*) e as plataformas de integração, monitorização e orquestração utilizadas na JUP – o OpenESB e JCAPS (*Java Composite Appication Platform Suite*). Por fim, descrevemos a tecnologia *Java Enterprise Edition* (Java EE), utilizada na implementação do sistema JUP, e outra tecnologia utilizada para automatizar a implementação deste – o Apache ANT.

4 Metodologias e Procedimentos

No presente capítulo iremos apresentar os métodos e procedimentos adoptados ao longo do desenvolvimento do sistema JUP.

Na secção 4.1 iremos descrever a arquitectura do sistema JUP focando, na subsecção 4.1.1, a abordagem efectuada sobre a plataforma J2EE. Seguidamente, na secção 4.2, enunciaremos um método adoptado que teve bastante impacto no desenvolvimento da interface da aplicação – a extensão de componentes JSF. Na secção 4.3 e 4.4 apresentaremos o papel do JCAPS no sistema JUP e a forma como construímos os *Business Processes* para comunicar com sistemas externos e a forma como estarão ligados todos os portos. Finalmente, na secção 4.5 apresentaremos um objecto transaccional e na secção seguinte um método utilizado para o registo de acções e geração de tarefas e notificações – *Business Transaction Actions*.

4.1 Arquitectura do sistema JUP

A arquitectura do sistema JUP assenta sobre uma filosofia SOA (*Service Oriented Architecture*), com os serviços (*Web Services*) implementados em J2EE.

O diagrama seguinte descreve a arquitectura aplicacional do sistema JUP, em que se realça as três camadas da JUP: a camada de apresentação, a camada de

negócio com as componentes dos *Web Services* de integração e do mecanismo de orquestração SOA e a camada de persistência (base de dados).

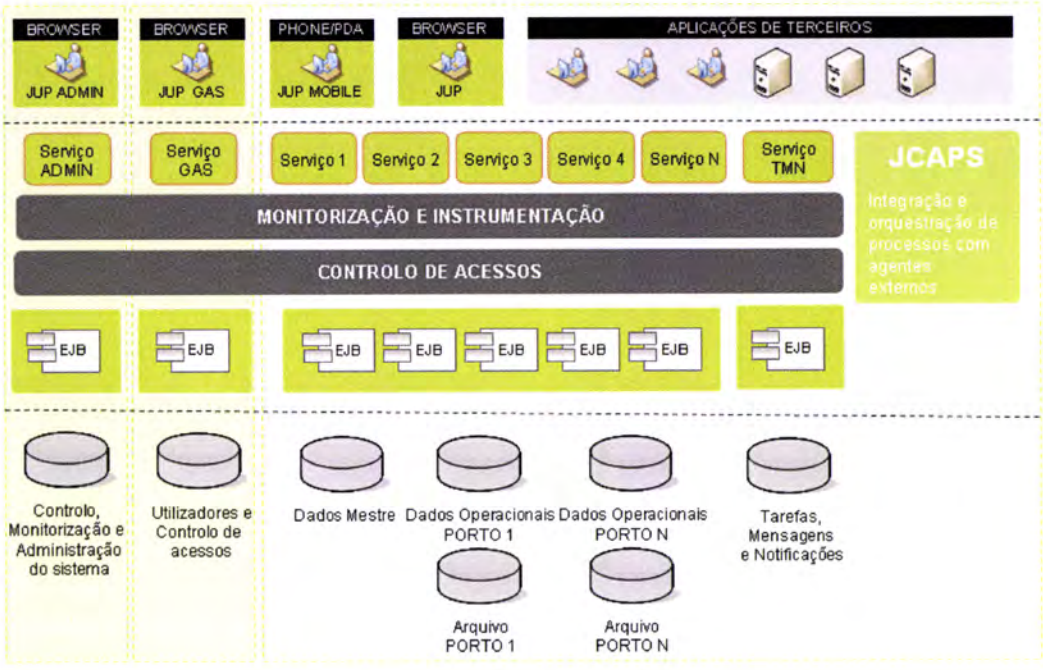


Ilustração 23 – Arquitectura aplicacional do sistema JUP.

Como podemos verificar pela figura anterior, o sistema JUP divide-se em JUP ADMIN (Administração do sistema), JUP GAS (Gestão de Acessos e Segurança), JUP Mobile (JUP para aplicações móveis) e JUP aplicação, sendo a camada de apresentação acedida tanto por terceiros (companhias e agentes de navegação) como pela Administração Portuária.

Uma das componentes principais da arquitectura apresentada anteriormente é a plataforma de integração e orquestração SOA, responsável por implementar um

Enterprise Service Bus (ESB) para conectar recursos e aplicações, disponibilizar *Web Services* e implementar mecanismos de interacção entre eles, com garantia de entrega de todas as transacções e/ou mensagens.

Adicionalmente, esta plataforma é responsável por orquestrar todos os processos de negócio implementados e de disponibilizar serviços de apresentação interactivos e "*machine-to-machine*" assente em tecnologias de *Web Services* e/ou protocolos B2B.

Uma arquitectura em três camadas apresenta diversos benefícios, dos quais destacamos os seguintes:

Segurança – Acessos restritos e controlados entre e em cada nível de camadas;

Modularidade – Vários módulos com objectivos específicos e agrupados por camada. Tornando mais fácil a manutenção, adicionar mais funcionalidades e principalmente reutilizar componentes;

Escalabilidade – Capacidade de dimensionar cada um dos níveis de camadas, de forma diferenciada, respondendo a novos requisitos como o aumento de utilizadores ou de funcionalidades;

Eficiência – Cada módulo é desenhado para executar de forma optimizada uma dada tarefa;

Centralização das Regras de Negócio – Uma nova regra ou a alteração de uma já existente é feita apenas numa das camadas;

Uso eficiente das ligações à base de dados – Exposição de métodos simples que escondem a complexidade da solução;

4.1.1 Modelo J2EE

Como referimos anteriormente a JUP baseia-se numa arquitectura em multi-camada (camada de apresentação, camada de negócio, camada de persistência), que corresponde ao padrão de desenvolvimento J2EE.

Vamos especificar estes componentes representados, assim como as suas características:

4.1.1.1 Camada de Apresentação

Esta camada corresponde à interface do utilizador, que realiza a interacção com os serviços de negócio, validações, entre outros. É o ponto de contacto entre o utilizador e a aplicação, onde este pode visualizar e inserir dados. Pode ser, por exemplo, uma página HTML, JSP, JSF, ou uma aplicação Swing. Na JUP usámos *Java Server Faces*, mais especificamente, *Facelets* com páginas em JSPX com *Java Beans* de suporte.

Nos anexos A e B podemos visualizar alguns ecrãs da JUP.

4.1.1.2 Camada de Negócio

Esta é a camada que concretiza a lógica de negócio. A camada inclui as ligações às entidades de negócio. As camadas de domínio e persistência são desenvolvidas com *Enterprise Java Beans* 3.0 (EJB 3.0).

Componentes principais:

- *Facade* que encapsula a lógica do negócio, utilizando os objectos *Service Beans* que são expostos como *Web Services*, de modo a permitir o acesso pela camada de apresentação e também a orquestração através do SOA.
- As regras de negócio definidas recorrendo ao BPEL ou com recurso a *Service Beans*, sempre que tal se considere vantajoso. A execução destas regras é suportada por um motor englobado na plataforma de SOA utilizada. O acesso à base de dados é realizado com recurso à camada de persistência.
- Integração e orquestração baseada no motor SOA JCAPS e composta por diversos *Web Services*, para desempenhar determinadas funções de integração e orquestração.

4.1.1.3 Camada de Persistência

Nesta camada encontram-se os mecanismos de persistência, que tratam do armazenamento dos dados, da sua transformação em objectos de BD e vice-

versa. Esta camada utiliza *Entity Beans*, que são classes Java que representam tabelas e vistas da Base de Dados. As ferramentas usadas na JUP são a solução *Java Persistence API* com a ferramenta Hibernate, recorrendo a uma base de dados Oracle.

4.2 Extensão de componentes JSF

Inicialmente iniciou-se o desenvolvimento da interface com a utilização de componentes do projecto ICEfaces (www.icefaces.org) mas, devido ao facto desta ser uma tecnologia emergente e pouco estável, decidiu-se estender as componentes do JSF, permitindo-nos ser autónomos. As componentes do ICEFaces iriam afectar todo o comportamento da aplicação, não nos dando muita liberdade para controlar e manipular este comportamento, além de que estes componentes apresentavam vários *bugs* a nível das suas funcionalidades. Para aplicações complexas, como a JUP, o ideal é mesmo criar as componentes próprias, para evitar correr riscos aquando a necessidade de acrescentar novas funcionalidades, que podem ter implicações graves no desenvolvimento da aplicação.

O *Java Server Faces*, disponibiliza algumas componentes standard que permite-nos criar as nossas próprias extensões: componentes, *renderers*, validadores e conversores. Assim, podemos sempre manipular e controlar o comportamento destas mediante as necessidades que surjam durante o desenvolvimento da

aplicação. Isto é muito vantajoso, numa aplicação de negócio exigente e com características pouco vulgares. Problemas como validações e outras funcionalidades extra que sejam necessárias, são facilmente contornados, optimizando o trabalho do programador, que evita ter que implementar estas funcionalidades no *bean* ligado à página.

Um exemplo de uma componente estendida na JUP é a *i:lookup*, que passamos a especificar de seguida. A componente *i:lookup*, é dos componentes mais complexos que a JUP utiliza e foi criada para combater uma necessidade da aplicação que é comum em quase todas as páginas. Vejamos o seguinte caso: estamos a criar uma ficha para registar um navio no sistema e um dos campos do formulário refere-se ao seu porto de registo. Este porto de registo tem um código e uma descrição que o identifica. Havendo centenas de portos registados no sistema, faz todo o sentido haver um esquema de pesquisa que nos permita procurar portos rapidamente. Deparados com esta situação surgiu a ideia de uniformizar estes 3 pontos num só – a componente *i:lookup*. Que permite ao utilizador editar um código com o fim de este ser avaliado posteriormente, ou aceder por um botão a uma *popup* de pesquisa. Vejamos o seu aspecto:

Ilustração 24 – Aparência da componente i:lookup.

Esta componente estende uma simples *InputText* e possuí várias propriedades extra, nomeadamente *hiddenValue* (valor "escondido" passado pela *popup*, que corresponde ao *id* do objecto), *descriptionValue* (valor correspondente à descrição), *popupUrl* (url da *popup* de pesquisa), *descriptionFieldDisabled* (propriedade responsável por bloquear a zona de descrição), entre outras. Na tabela abaixo, podemos ver a aplicação directa da *i:lookup* numa página JSP (exemplo de um porto):

```
<i:lookup id="lkpPorto"
          binding ="#{bean.bindings.porto}"
          value="#{bean.portoCodigo}"
          maxlength="20"
          hiddenValue="#{bean.portoId}"
          descriptionValue="#{bean.portoDescricao}"
          hiddenDescValue="#{bean.portoDescricao}"
          descriptionFieldDisabled="true"
          displayDescriptionField="true"
          popupUrl="../popups/popupPortos.jsp"
          valueChangeListener="#{bean.preenche}"
          width="70%">
</i:lookup>
```

Tabela 2 – Código JSP da componente *i:lookup*, utilizada na JUP.

4.3 JCAPS na JUP

O objectivo da plataforma JCAPS/OpenESB na JUP direcciona-se para a integração com ERP's, envio de listas de passageiros para SEF, encapsulamento das regras de negócio, envio de tarefas e notificações, serviços de envio e

recepção de mensagens para entidades externas, serviços de integração de mensagens entre a administração portuária e a Alfândega e vice-versa e serviço de transformação de mensagens.

As ferramentas utilizadas nesta fase foram o JCAPS 6 e OpenESB Nightly. A integração das duas ferramentas ocorreu através do Netbeans 6.5.

Os sistemas integrados foram:

- DGITA – SDS (Mensagens M, IFCSUM/CUSRES - Manifesto, Mensagens P)
- SafeSeaNet
- VTS local
- VTS costeiro
- Terminais
- Agentes/OTM
- SAP-WS/SAP/GIAF
- Outros portos

Basicamente o processo adoptado consiste na criação de BPELs, com a definição de cada interface por WSDL, para interligar os diversos componentes do sistema. Este WSDL é conectado, através de um *partner-link* ao nosso *Business Process*. É em cada BPEL que definimos o comportamento e processamento das mensagens e/ou de outras integrações/orquestrações.

Finalmente, após validarmos os nossos *Business Processes*, construímos, a partir destes, as *Composite Applications*, onde é configurado o servidor de execução e posteriormente o *build* e *deploy* desta.

Vejamos um exemplo de um BPEL simples que representa a recepção de uma mensagem, através de um serviço (WSDL), do processamento desta e finalmente da resposta:

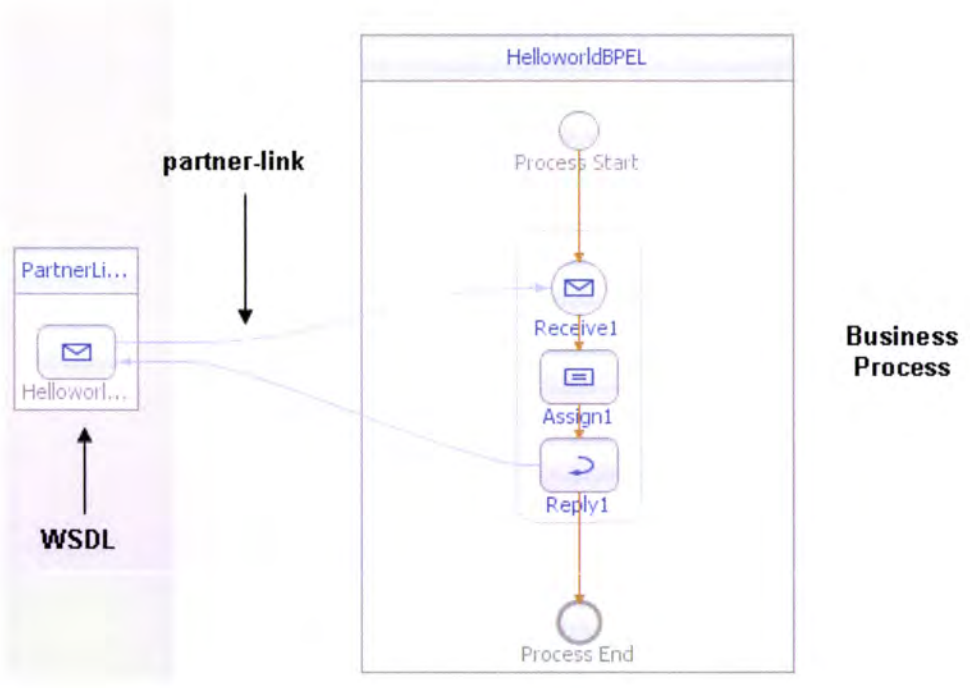


Ilustração 25 – Exemplo simples da construção de um BPEL.

Este é o aspecto da mensagem recebida pelo serviço – resultado do teste da *Composite Application*:

```
<soapenv:Envelope urlSchemaLocation="http://endereco">
  <soapenv:Body>
    <hel:HelloworldWSDLOperation>
      <part1>Hello World</part1>
    <hel:HelloworldWSDLOperation>
  <soapenv:Body>
</soapenv:Envelope>
```

Tabela 3 – Mensagem recebida em XML (Input).

4.4 Interoperabilidade entre todos os portos

Ilustração 26 – Portos da APP.

A troca de informação entre todos os portos da Associação dos Portos de Portugal é um dos grandes desafios desta aplicação. Esta característica permite uma optimização do trabalho efectuado pelos intervenientes envolvidos no sistema, evitando a introdução duplicada de dados.

O processo de comunicação ocorre sempre que o destino seguinte de um navio com escala num porto da APP, seja outro porto da APP.

A comunicação entre portos é efectuada pela invocação de *Web Services*. Cada porto disponibiliza um *Web Service* a ser invocado pelos restantes portos. Este *Web Service* será composto por um método que recebe o conteúdo da mensagem passado por parâmetros.

Em caso de erros na execução do método estes são comunicados ao porto que invocou o método através de estruturas *fault* do protocolo SOAP, definidas para o efeito.

A utilização de um método do *Web Service* por mensagem, origina uma estruturação do serviço que permite uma concretização modular do mesmo. Esta estruturação oferece vantagens significativas em termos de separação funcional e semântica das diferentes funcionalidades disponibilizadas. Isto repercute-se favoravelmente no processo de desenvolvimento e depuração. Desta forma consegue-se um serviço com um grau de robustez superior. Torna-se também possível controlar as funcionalidades disponibilizadas, podendo, se necessário, impedir certas operações e mais facilmente disponibilizar novas. Este modelo de concepção encontra-se em harmonia com a filosofia SOA, adoptada no projecto, e é o JCAPS o responsável por esta comunicação.

4.5 Objecto *Business Transaction*

Business Transaction (BT), tal como o nome indica, representa uma transacção de negócio. Concretamente na nossa aplicação, a criação/edição/leitura/anulação de processos. Este foi um conceito bastante explorado ao longo do

desenvolvimento da aplicação e surgiu a partir da necessidade de melhorar a performance no carregamento de dados para apresentar na camada de apresentação.

Antes da criação das BT's o carregamento dos dados necessários a cada processo era feito por passos e evocado pela camada de apresentação. Em casos simples, onde são feitas apenas duas ou três pesquisas à base de dados, o comportamento da aplicação era rápido e estável. Quando esta marca era ultrapassada (mais de três acessos) a performance caía bastante e começou-se por criar *store procedures* para obter os dados necessários apenas num acesso à base de dados, sem aceder pelos *Entity Beans*. Não estando ainda satisfeitos com a performance obtida (que melhorou em cerca de 40% - 50%) e tendo em conta que havia em muitos casos (dos mais complexos) interesse em obter o objecto inteiro e alguns dados de outros objectos agregados a este, surgiu o objecto *Business Transaction*, no qual é carregada toda a informação necessária para cada processo. Este objecto é criado na camada de negócio e é invocado apenas por um acesso desde a camada de apresentação. Assim deixámos de ter vários pedidos na camada de apresentação passando a ter apenas um, no qual se obtém apenas um objecto (BT) já carregado com toda a informação necessária. E um ecrã que demorava cerca de 20 segundos a ser carregado passou para 1 ou 2 segundos, dependendo dos processos que seriam posteriormente despoletados.

4.6 *Business Transaction Actions*

Ao longo de toda a aplicação podemos verificar a utilização de *Business Transaction Actions* (BTA). Uma BTA consiste no registo de uma acção/actividade que ocorra no sistema. No caso da JUP, o conteúdo de uma BTA é composto por uma descrição de uma acção que é executada (criação/actualização de uma declaração, por exemplo), pela identificação do utilizador que executou a mesma e pela data e hora dessa acção.

Este mecanismo permite-nos, facilmente, ter conhecimento de todas as intervenções que aconteceram no sistema.

Além desta vantagem que as *Business Transaction Actions* nos oferecem, existe uma outra funcionalidade essencial ao sistema JUP onde estas têm um papel fundamental. Referimo-nos ao serviço de tarefas e notificações.

Aquando a execução de uma série de acções na JUP, muitas outras são desencadeadas através da recepção de notificações e tarefas. Um dos melhores exemplos, que será explorado em pormenor no próximo capítulo, é o anúncio da chegada de um navio ao porto. Sendo este processo executado com sucesso, algumas entidades têm que ser notificadas e, consequentemente, têm tarefas a cumprir. O que acontece no sistema é a criação de uma BTA que regista a acção "Criar Aviso de Chegada". Posteriormente a esta actividade, o sistema olha para a BTA "Criar Aviso de Chegada" e processa o mecanismo de geração de notificações e tarefas evocando um *Web Service* que posteriormente comunica com as respectivas entidades a serem notificadas de que um navio está a chegar

ao porto, alertando-as das tarefas que têm que realizar. Neste preciso momento também é invocado o serviço de comunicação com o exterior, através do JCAPS.

4.7 Síntese

No 4.º capítulo descrevemos e justificámos os métodos e procedimentos adoptados ao longo do desenvolvimento do sistema JUP. Ficámos a conhecer qual a abordagem SOA feita sobre o sistema explicando a metodologia adoptada para a sua implementação. Especificámos o papel do JCAPS no sistema JUP, exemplificando a utilização desta ferramenta na parte de integração/comunicação com sistemas externos. Por fim apresentámos o conceito de *Business Transaction* e explicámos como foi feito o motor de geração de tarefas e notificações na JUP.

5 Caso de Estudo

Este capítulo apresenta um caso de estudo de um dos processos da JUP – Aviso de Chegada. Na secção 5.1 iremos descrever este processo, seguindo para a apresentação da modelação UML do mesmo (na secção 5.2). Seguidamente, apresentaremos a interface desenhada para o processo Aviso de Chegada (secção 5.3), o controlo de tarefas e notificações (secção 5.4) e, finalmente, a integração/orquestração envolvida neste processo (secção 5.5).

5.1 Descrição do processo Aviso de Chegada

Aquando a aproximação de um navio a um porto, é necessário anunciar a sua chegada ao sistema JUP. O processo do Aviso de Chegada é o responsável por registar este acontecimento, podendo ser visto como o processo principal da JUP, uma vez que representa o início do ciclo de vida de uma escala desencadeando uma série de actividades, correspondentes a outros tantos processos do sistema. Mais especificamente, as autorizações, declarações e serviços que são efectuados no âmbito de uma escala de navio.

Passemos a especificar este processo através da sua modelação em UML.

5.2 Modelação UML

Este subcapítulo pretende apresentar através de modelação UML as actividades necessárias à elaboração de um aviso de chegada.

5.2.1 Diagrama de Classes – Aviso de Chegada

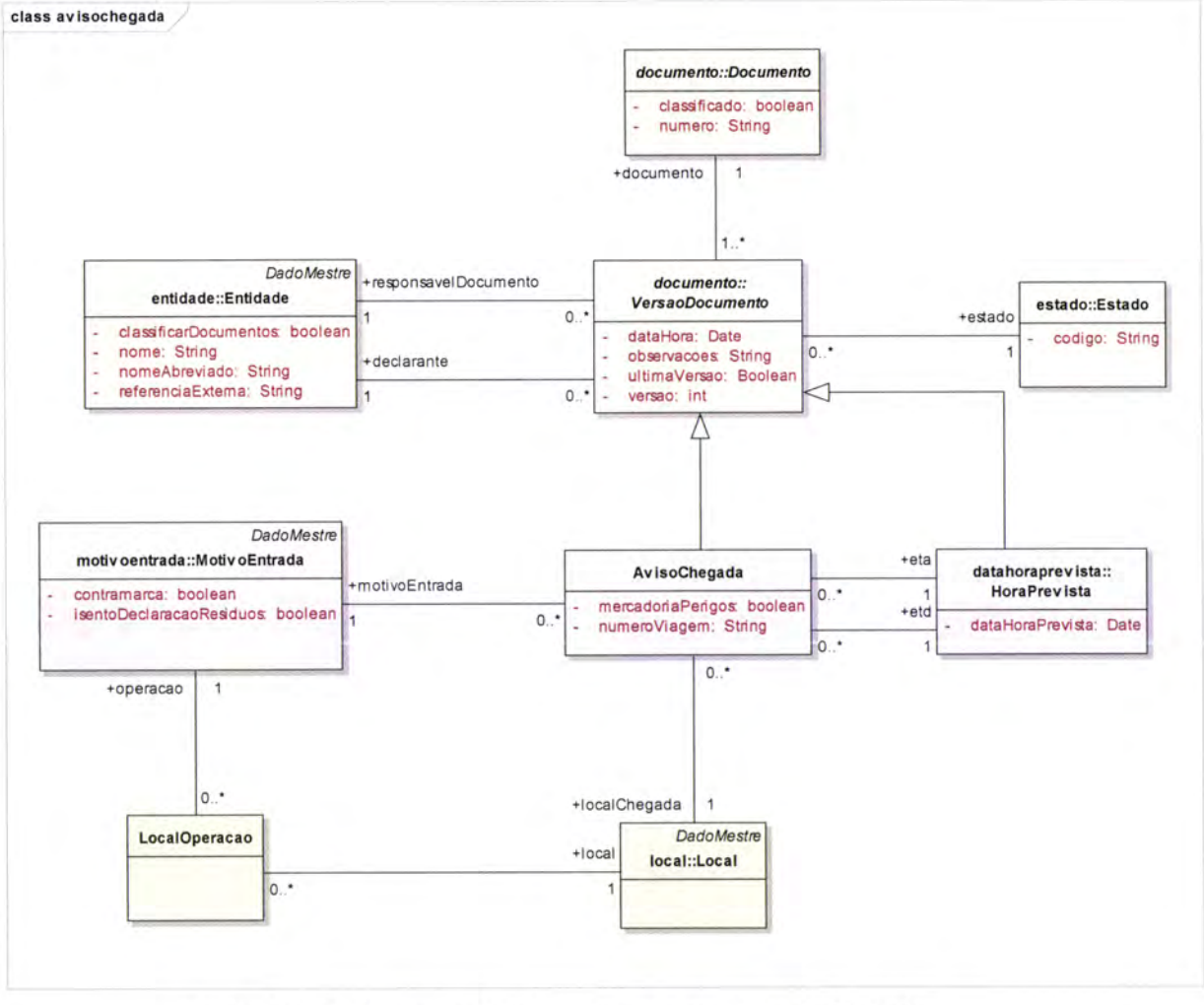


Ilustração 27 – Diagrama de Classes do processo Aviso de Chegada.

5.2.2 Caso de Uso – Aviso de Chegada

O caso de uso abaixo representa todas as acções associadas ao processo Aviso de Chegada – criar, consultar, actualizar e anular. Iremos concentrarmo-nos mais na primeira acção, uma vez que é uma das que despoleta mais acontecimentos dentro do sistema e que nos permite entender melhor as vantagens de termos uma arquitectura orientada a serviços.

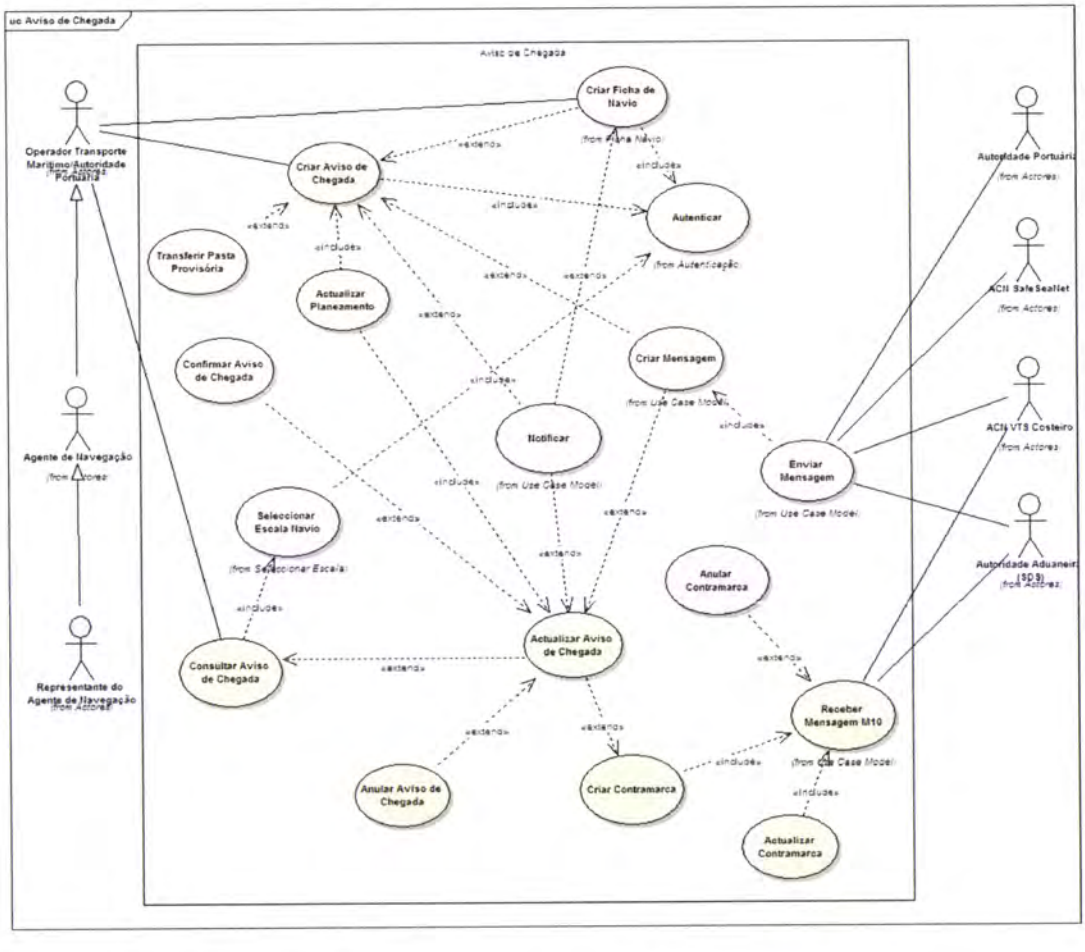


Ilustração 28 – Caso de uso do processo Aviso de Chegada. (Ver anexo D)

Vejamos agora a especificação da acção "Criar Aviso de Chegada":

Criar Aviso de Chegada	
Actores	Operador Transporte Marítimo, Agente de Navegação, Representante do Agente, Autoridade Portuária. Em caso de NRP, considera-se que assume o papel de agente do navio a Autoridade Marítima local.
Descrição	Este caso de uso especifica os passos para a criação de um Aviso de Chegada.
Pré-condições	1. Actor tem um perfil que lhe permite efectuar esta operação.
Caminho Principal	1. *Include*: Autenticação. 2. O actor escolhe a opção de criar um aviso de chegada, 3. Se o actor for um representante do agente, deverá indicar qual a entidade que está a representar. Para o efeito, a JUP exibirá uma janela onde o actor seleccionará a entidade. A janela só exibe as entidades parametrizadas na JUP para este representante. 4. O actor selecciona o navio da escala através do mecanismo de pesquisa que consiste numa lista de valores. 4.1 Se a ficha do navio pretendido estiver no estado provisório, o actor é notificado de que não pode utilizar esse navio porque tem uma ficha no estado provisório, ficando impedido de

	criar o aviso de chegada 4.2 Se a ficha do navio pretendido estiver no estado provisório, o actor não pode propor uma nova versão 5. É apresentado um ecrã com a informação que o actor preenche de acordo com a especificação. 6. O actor grava o documento e o sistema valida o preenchimento dos campos, ficando o aviso de chegada no estado provisório. 6.1. O sistema gera notificações para as entidades parametrizadas e o aviso de chegada passa a estar disponível para consulta dos outros actores. 6.2 É criada uma mensagem M1 para o SDS (DGAIEC) sendo o actor notificado da entrega ou não da mensagem. São geradas também mensagens para outros sistemas externos (Escala_Not para o ACN VTS Costeiro e Port_Not para o ACN SafeSeaNet). 6.3 É gerado um registo para o planeamento de um "Navio Anunciado" 6.4 Se já existirem escalas abertas para o navio em causa o sistema notifica o utilizador. 7. Depois de o documento estar gravado com sucesso é possível o actor confirmar o aviso de chegada.
Caminho Alternativo	1° Caminho alternativo 1. O actor pode no decorrer da criação do aviso de

	chegada propor a criação de uma nova versão da ficha de navio quando o estiver a seleccionar na lista de valores. 2. O sistema gera uma nova versão da ficha de navio no estado provisório para posterior validação por parte da Autoridade Portuária. 3. Se o actor não encontrar o navio pretendido através da lista de valores, pode criar uma ficha de navio que fica também no estado provisório para posterior validação por parte da AP. 2º Caminho alternativo 1. Após escolha do navio o sistema sugere a importação de dados, caso exista uma pasta provisória para esse navio. O actor pode seleccionar a importação dos dados para o sistema pré-preencher alguns campos do aviso de chegada. 3ª Caminho alternativo 1. No ponto 4 do caminho principal o actor pode cancelar a criação do aviso de chegada.
Caminho de Excepção	1º Caminho de excepção: 1. No ponto 6 do caminho principal não estão preenchidos os campos obrigatórios. Neste caso, o sistema apresenta uma notificação a informar qual/quais os campos em falta. 2º Caminho de excepção: 1. No ponto 6 do caminho principal, os dados preenchidos não são válidos. Neste caso, o sistema

	apresenta uma notificação dos campos incorrectos.
Pós-condições	1. A escala fica criada. 2. É possível efectuar pedido de manobras e marcações de manobras nos estados provisório ou previsto. 3. É possível efectuar actos declarativos previstos: • Declaração Hazmat (obrigatório caso tenha indicado que tem mercadorias perigosas) • Declaração ISPS • Declaração de Resíduos • Declaração Geral de Carga • Manifestos de Mercadorias • Pedidos de Licenças • Declaração Marítima de Saúde (obrigatório se o porto precedente não for nacional) • Lista de Bond Stores • Listas de Tripulantes e Passageiros • Pedido de Isenção de IVA • Requisições de Serviços 4. É possível a criação de autorizações de entrada nos diversos estados. 5. O aviso de chegada fica no estado provisório.

Tabela 4 – Descrição do caso de uso do processo Aviso de Chegada.

5.2.3 Diagrama de actividades – Aviso de Chegada

Ver anexo D.

5.2.4 Diagrama de estados – Aviso de Chegada

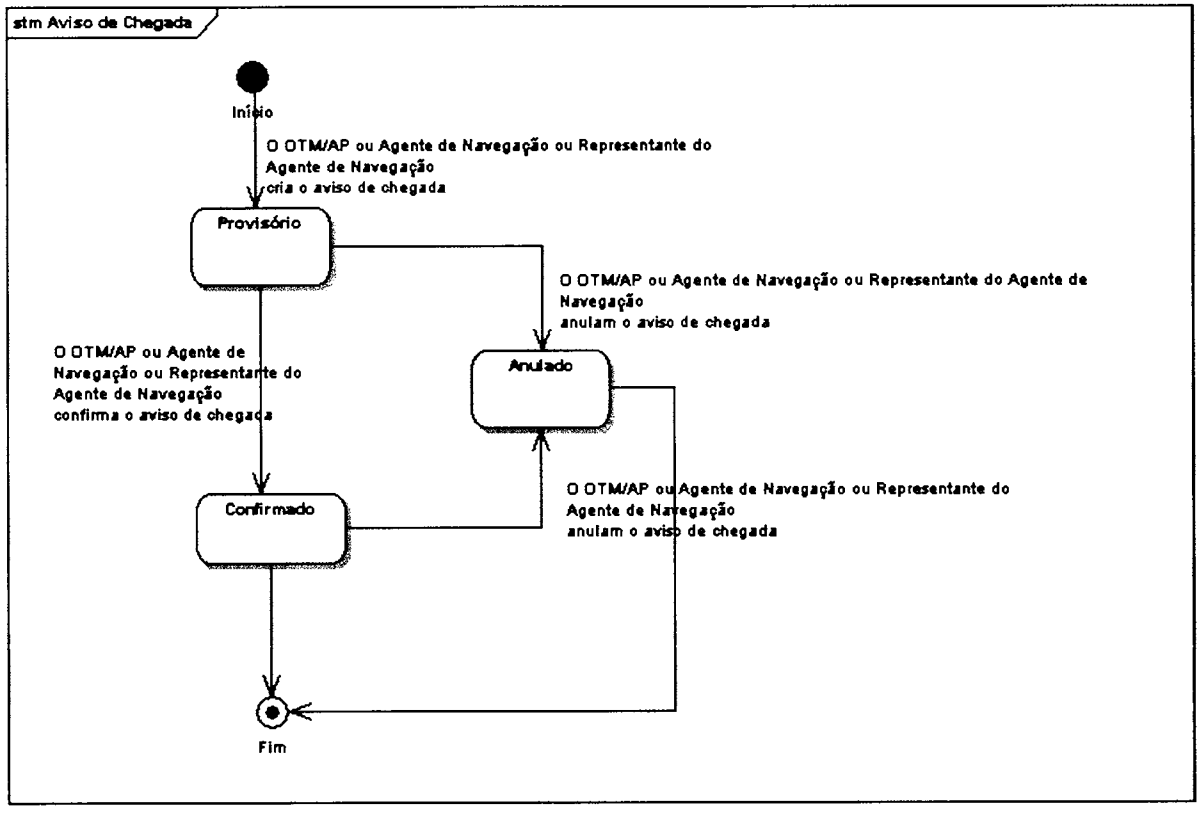


Ilustração 29 – Diagrama de estados do processo Aviso de Chegada.

5.3 Interface

Podemos ver no Anexo B o ecrã correspondente ao processo do Aviso de Chegada. Por detrás desta interface está um *Backing Bean* que faz a ligação com a camada de negócio através de *Web Services*.

5.4 Geração de Tarefas e Notificações

Após a criação de um Aviso de Chegada, várias entidades são notificadas podendo receber tarefas para executar posteriormente.

Vejamos a seguinte imagem que esquematiza esta situação:

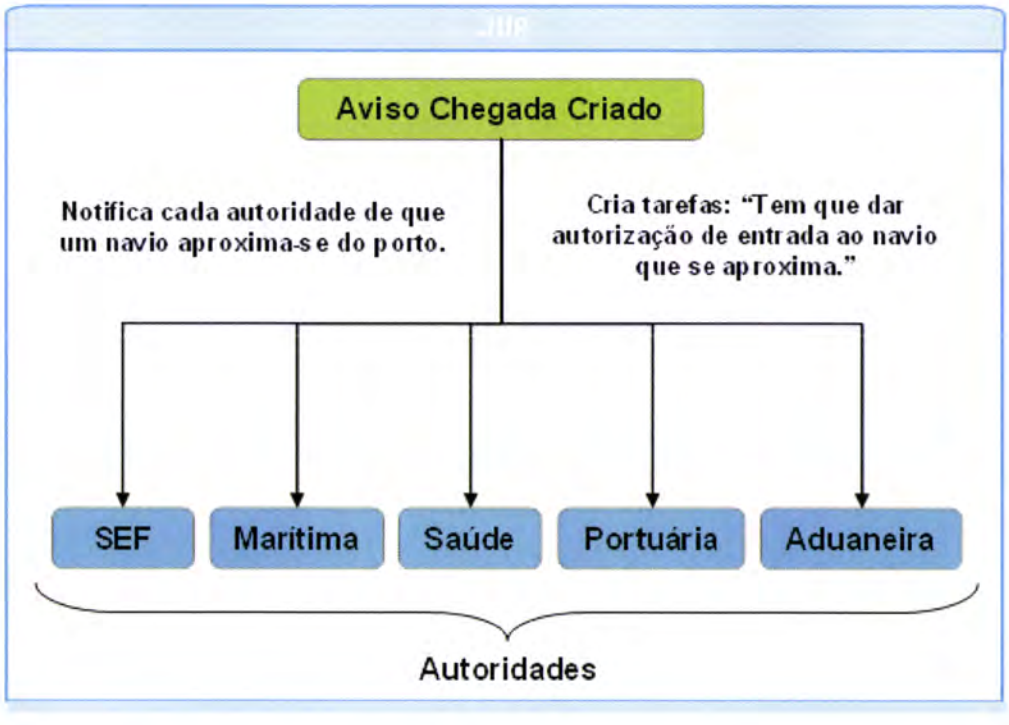


Ilustração 30 – Notificações e tarefas desencadeadas após o anúncio da chegada de um navio ao porto.

O sistema ao registar a BTA de que um Aviso de Chegada foi criado, invoca o serviço de tarefas e notificações para as gerar/enviar às autoridades acima representadas. O navio só pode atracar no porto caso estas autoridades dêem o seu parecer positivo.

5.5 Integração/Orquestração

No momento em que são geradas as notificações e tarefas é executado o serviço de comunicação através da plataforma JCAPS. Como vimos na descrição do caso de uso do processo Criar Aviso de Chegada (na secção 5.2.2), são enviadas mensagens para sistemas externos: a mensagem M1, para a DGAIEC, e Escala_Not para o ACN VTS Costeiro e Port_Not para o ACN SafeSeaNet.

Posteriormente ao envio da mensagem M1 para o sistema SDS da DGAIEC, é recebida a mensagem M10 deste sistema e uma Contramarca. No anexo E podemos ver o BP do envio da mensagem M1, no anexo F a recepção da mensagem M10 e no anexo H a recepção da Contramarca.

5.5.1 Envio da mensagem M1

Previamente ao envio da mensagem M1 é necessário reunir todos os dados que compõem a mensagem. Tudo começa no processamento da BTA Aviso de

Chegada e seguidamente com a pesquisa de dados através de *Web Services*, como está demonstrado na figura abaixo.

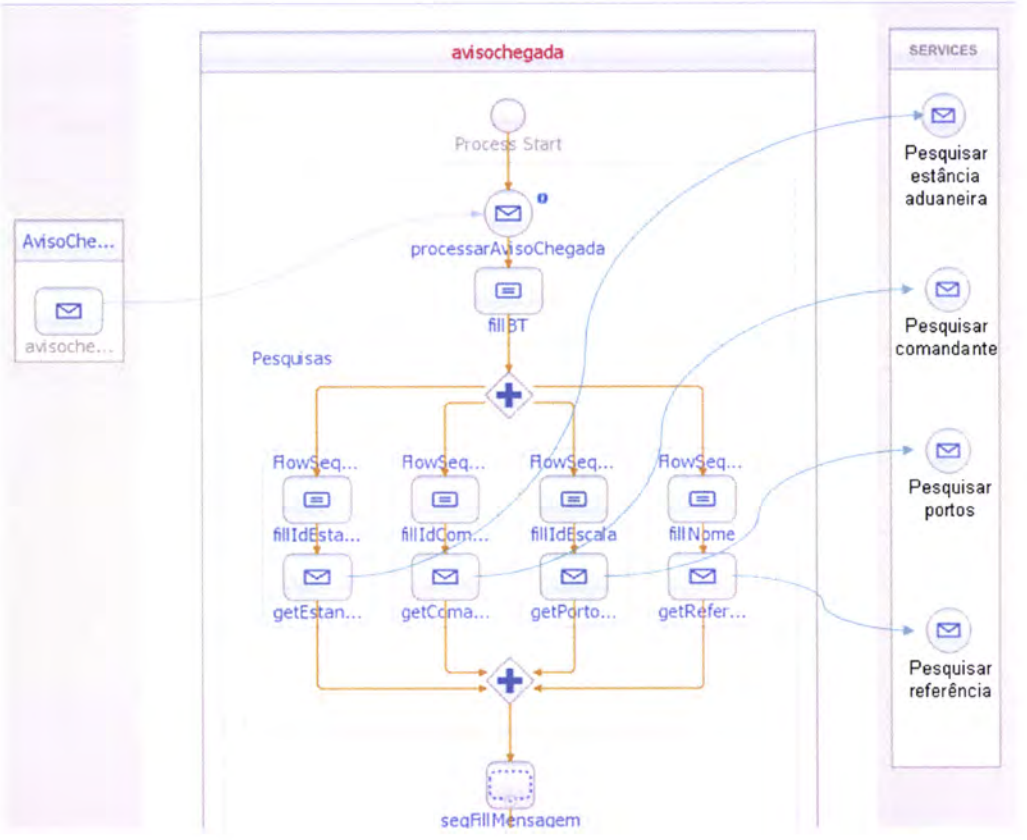


Ilustração 31 – Esquema representativo da fase de preparação da mensagem M1.

Como podemos visualizar, existe o WSDL Aviso de Chegada (do lado esquerdo), um fragmento do *Business Process* e os *Web Services* necessários para executar esta operação.

Após a reunião e confirmação de todos os dados a próxima fase é a construção da mensagem.

Ilustração 32 – Diferentes fases da construção da mensagem M1 (continuação da imagem anterior).

Na figura ao lado podemos ver a continuação do nosso *Business Process*, onde é construída a mensagem M1 com o conteúdo dos dados do formulário preenchido que deu origem ao aviso de chegada de um novo navio ao porto. Como podemos verificar este preenchimento é sequencial, começado por preencher o cabeçalho da mensagem e seguido das entidades, portos, forma de atracação, tipo de tráfego, locais e motivos de entrada e o comandante do navio.

Em cada sequência apresentada são verificados os valores preenchidos (também com a utilização de *Web Services*) e mediante esta verificação é escrita a

mensagem. Podemos ver o exemplo do preenchimento dos três portos anteriores e seguintes:

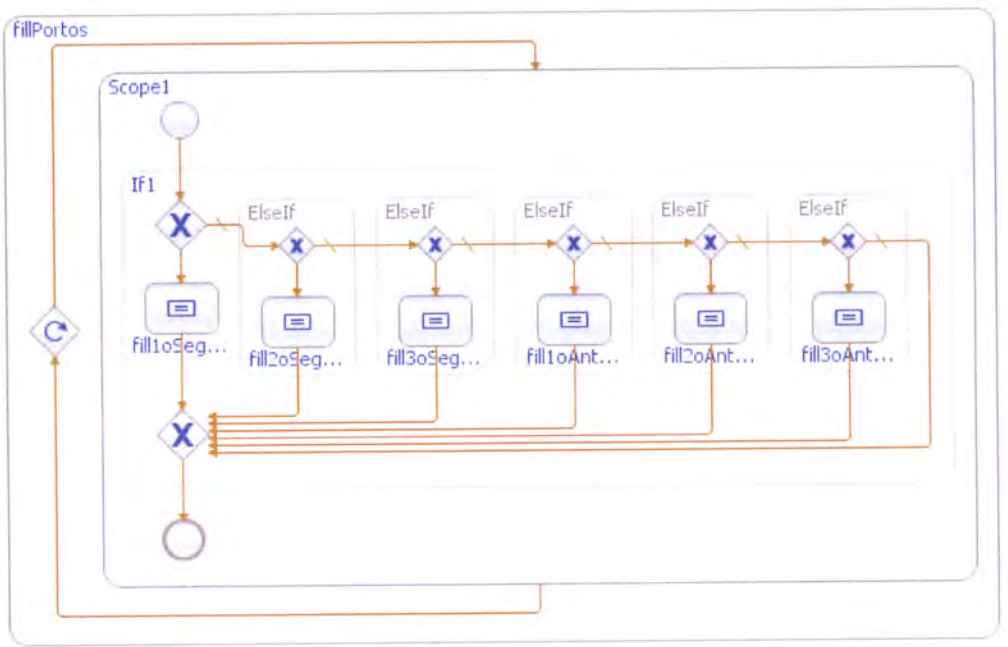


Ilustração 33 – Preenchimento dos portos anteriores e seguintes (campos da mensagem M1).

Observando a figura facilmente percebemos que está representado um ciclo com uma condição *if-then-else*.

Após o preenchimento da mensagem o próximo passo é traduzir a mesma para o formato EDIFACT e posteriormente codificá-la para base 64.

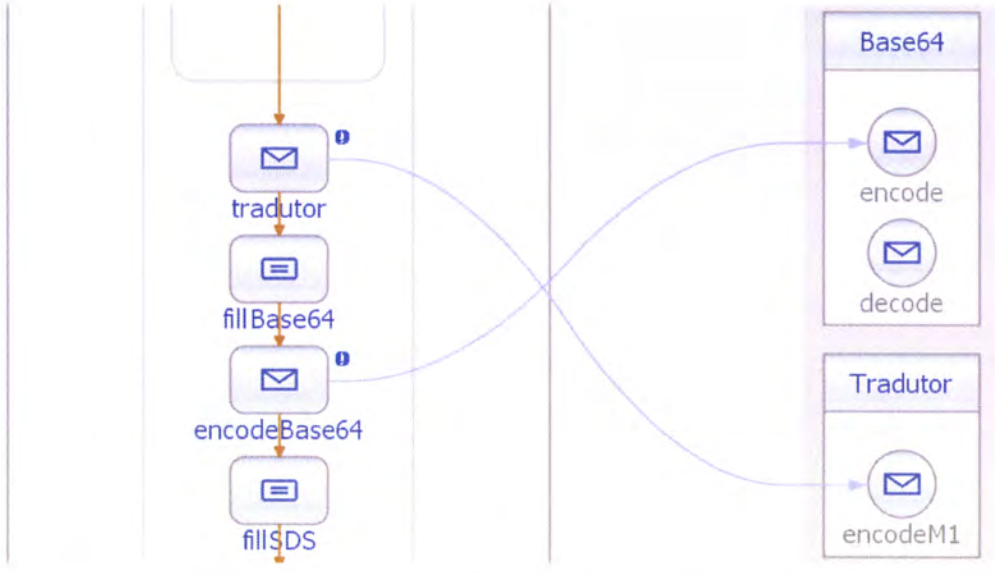


Ilustração 34 – Tradução e codificação da mensagem M1.

Finalmente, a mensagem está pronta a ser enviada para o sistema SDS. A figura abaixo representa esta fase do processo, onde podemos verificar a invocação do serviço que envia a mensagem M1. Posteriormente, é verificado se a mensagem foi enviada com sucesso e enviada a resposta para o WSDL, com esse resultado.

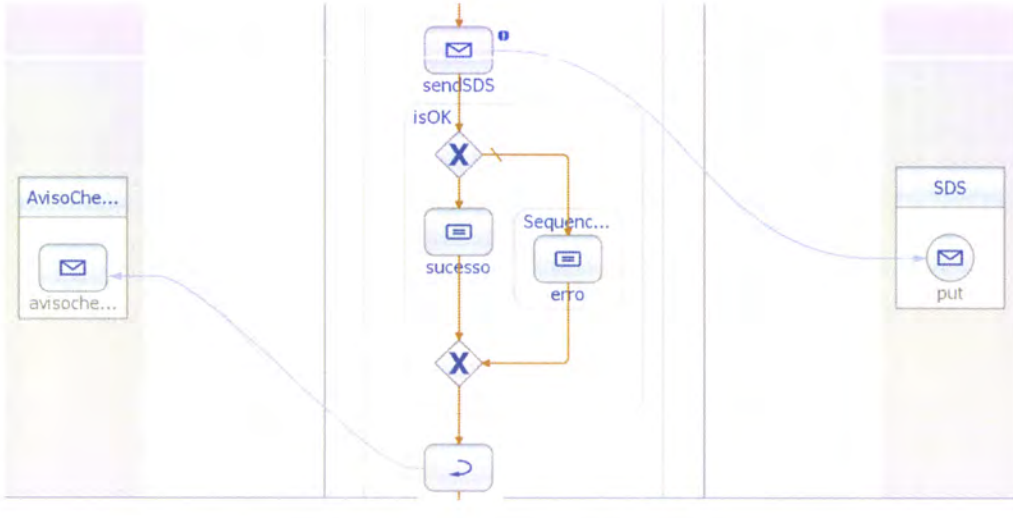


Ilustração 35 – Envio da mensagem M1 para o sistema SDS.

Após o envio da mensagem com sucesso, esta é associada a uma escala e gravada no sistema JUP pela invocação do *Web Service* de mensagens, como podemos verificar de seguida.

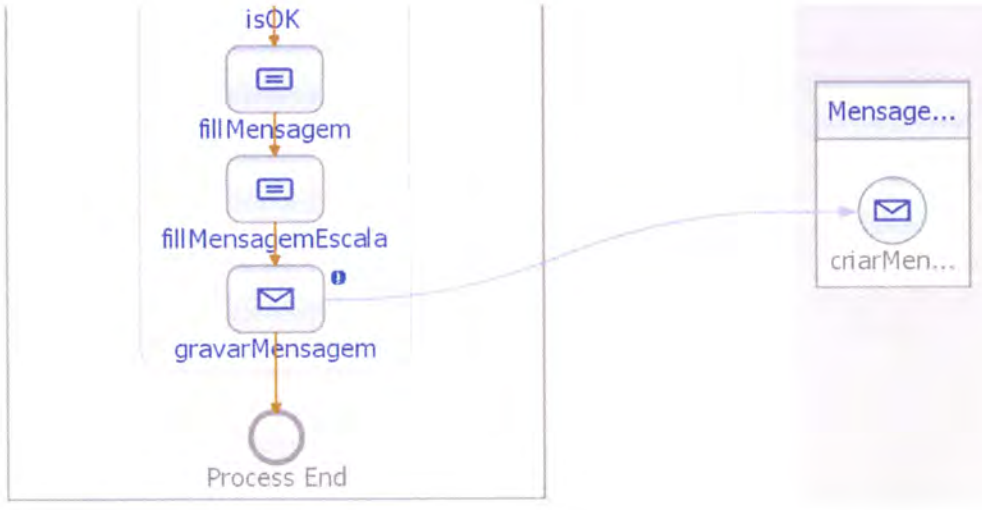


Ilustração 36 – Gravação da mensagem M1 no sistema.

E assim concretizamos o processo do envio da mensagem M1 do sistema JUP para o sistema SDS. No anexo E podemos ver na íntegra o *Business Process* descrito acima.

5.6 Síntese

O presente capítulo mostrou-nos as principais fazes de implementação do processo Aviso de Chega, um dos processos da JUP. Pormenorizámos como é

processado o envio de mensagens para o exterior, através do motor JCAPS, uma das características mais importantes deste projecto.

6 Conclusões [e perspectivas futuras]

Neste capítulo iremos analisar os principais objectivos do projecto e concluir acerca da concretização dos mesmos e de todo o trabalho realizado.

Também neste capítulo, iremos apresentar o que consideramos relevante para um desenvolvimento futuro.

6.1 Conclusões

O principal objectivo deste projecto concentrava-se na apresentação de uma abordagem SOA sobre o sistema de gestão portuária JUP. De realçar que as conclusões que se seguem têm em conta o facto do contexto empresarial em que o projecto estava envolvido.

Optámos por utilizar a tecnologia J2EE, seguindo o modelo multi-camada que esta oferece. Para aplicações de dimensões semelhantes à JUP, esta tecnologia revela grandes vantagens para os programadores, uma vez que, como existem camadas bem definidas na aplicação, cada elemento da equipa de desenvolvimento pode trabalhar em diferentes níveis, separadamente. Uma vez que é baseada em Java, adquire todas as vantagens desta linguagem, relevando-se a característica de multi-plataforma, que era um dos requisitos deste projecto. Outro pormenor vantajoso está relacionado com os baixos custos envolvidos, dado que se trata de uma tecnologia *open source* permitindo a utilização, por

exemplo, de ambientes de desenvolvimento gratuitos, como o Eclipse ou de servidores aplicacionais como o Glassfish e o acesso a APIs que ajudam o utilizador a entender os mecanismos internos da tecnologia. No que se refere a interoperabilidade com outros sistemas, J2EE revelou-se uma excelente proposta para plataforma de integração.

Por outro lado, existem algumas restrições para o programador, como a interacção com o servidor aplicacional, onde este está impossibilitado de ter um controlo absoluto sobre as *threads* de controlo deste. Outra desvantagem identificada foi a portabilidade ineficaz entre servidores aplicacionais, quando se tentou executar a aplicação no JBoss, onde o grau de complexidade desta operação mostrou-se elevado e optou-se por descartá-la.

Relativamente à camada de apresentação, a abolição do ICEFaces da aplicação e a opção de estender componentes trouxe várias vantagens a nível do comportamento da interface da aplicação. Esta alternativa permitiu-nos ter um controlo absoluto de cada componente, nomeadamente na resolução de validações mais alto nível, sem a necessidade de as propagar para outras camadas da aplicação (validações de formato, por exemplo) e, também, na apresentação de dados mais complexos (como o caso da *lookup* e *data table*). Outra vantagem, com efeito superior à anterior, refere-se ao *request scope* utilizado no projecto ICEFaces que não se comporta como o *request scope* standard para uma aplicação web Java. O comportamento do primeiro é semelhante a um *session scope*, uma vez que a cada novo pedido, os dados

ficam em sessão até que o tempo de expiração desta termine. Este facto deve-se a componentes do ICEFaces que necessitam desta propriedade para funcionar correctamente. Numa aplicação web Java, a cada novo pedido, os dados ficam apenas no *garbage collection.*

Ainda em relação à camada Web, o facto de com o JSF toda a lógica de navegação da aplicação e a configuração de algumas validações ser escrita apenas no ficheiro *faces-config.xml*, revelou-se uma vantagem face a outras tecnologias onde é necessário existirem ficheiros diferentes para o fazer.

Continuando no tema das validações na aplicação, uma vez que se trata de uma arquitectura SOA e que ideologicamente a aplicação poderá vir a ser acedida a partir de outras aplicações pela disponibilização de serviços, as validações essenciais à concretização do negócio não podem ser feitas apenas na camada web, mas sim também na camada de negócio. Desta forma foi criado um projecto de validação que é acedido por ambas as camadas.

A aplicação possuí processos de negócio com uma complexidade bastante acentuada e, à medida que esta crescia, verificou-se que a cada transacção de negócio, existiam várias dependências entre objectos, por exemplo, no caso do processo de criação/edição da Declaração Marítima de Saúde, onde é necessário carregar variada informação de outros objectos. Quando era executado o carregamento dos dados deste processo, a aplicação estava bastante lenta (cerca de 10 a 30 segundos a carregar o ecrã). Isto ocorre uma vez que, conforme a arquitectura apresentada anteriormente, para além do

acesso à base de dados ocorrer através da camada de negócio, por EJB's, as *queries* geradas pelo Hibernate são bastante complexas, dado que quando tentamos obter um objecto com outros objectos agregados, a performance ficava aquém do esperado.

A escolha desta ferramenta de persistência foi tomada com base no facto desta estar integrada com a versão EJB 3.0, havendo por isso compatibilidade com as outras tecnologias utilizadas e pelo facto do desenvolvimento da aplicação se tornar mais rápido. O programador deixa de ter que implementar o código SQL/JDBC, uma vez que este está encapsulado por detrás da implementação do Hibernate, ficando transparente para ele.

No entanto, consideramos que para aplicações com modelos de dados com uma complexidade acentuada o Hibernate apresenta algumas lacunas a nível de performance. E, se queremos construir *queries* mais complexas, é essencial conhecer muito bem a sua implementação para obter bons resultados, uma vez que este tipo de operação não é tão transparente para o utilizador.

Para combater o problema da performance, além de construirmos as nossas *queries* independentemente e de criarmos índices nas tabelas da base de dados, a utilização das *Business Transactions* também se revelou um contributo importante para a obtenção de resultados proeminentes. Ainda assim, concordamos que é algo a melhorar e um próximo passo seria construirmos a nossa própria camada de persistência e deixar de utilizar o Hibernate, tema que iremos abordar com mais profundidade no próximo subcapítulo.

Este projecto também permitiu-nos conhecer o ambiente de integração JCAPS e através deste ganhar sensibilidade suficiente para tomar decisões sobre este tipo de ferramentas. Esta ferramenta revelou-se um excelente suporte a todo o ciclo de vida do SOA, com um repositório próprio, assim como um ambiente de desenvolvimento próprio.

Uma funcionalidade que não foi aprofundada no desenvolvimento do nosso projecto e que nos parece interessante explorar é a segurança integrada no JCAPS, uma vez que permite a criação de perfis dentro do orquestrador e definir permissões para aceder às aplicações, algo que iremos explicar melhor no próximo subcapítulo.

Uma vez que é baseado em J2EE e utiliza o GlassFish com servidor aplicacional e suporte para o IDE NetBeans, não houve dificuldade em introduzir esta ferramenta na nossa aplicação. O JCAPS permite-nos implementar processos de negócio de uma forma rápida através da sua interface gráfica. A ferramenta é vantajosa para quem desenvolve, pois não precisa de entender a linguagem que é gerada e de uma forma intuitiva consegue criar os seus BPELs, garantindo uma arquitectura flexível, distribuída, confiável e compatível com os serviços de negócio (SOA).

No entanto, ao longo do desenvolvimento da parte de comunicação com a DGITA, isto é, o desenvolvimento de envio/recepção/retransmissão de mensagens, surgiram alguns problemas que consideramos ser uma desvantagem desta ferramenta. Quando testámos esta componente, tivemos

dificuldades em descortinar onde residiam os erros que ocorriam. Não ficámos satisfeitos com o sistema de *logs* do JCAPS, assim como o sistema de *debug* da ferramenta. Concluímos que para casos mais complexos, como o caso das mensagens, onde são obtidos os dados que queremos enviar na mensagem dos nossos objectos, feito o tratamento dos mesmos, criada a mensagem com estes, traduzida e codificada a mensagem e enviada via *Web service* para o sistema externo SDS, é-nos difícil ter um controlo absoluto sobre a forma como a ferramenta processa cada uma destas fases e perde-se muito tempo na tentativa de descortinar erros. Este pormenor obriga a que haja uma formação adequada para a utilização da ferramenta, o que não foi o caso para a maioria dos colaboradores da equipa de desenvolvimento, incluindo o autor deste documento. No seguimento do problema da falta de controlo sobre a ferramenta JCAPS, melhorias sobre a performance também foi uma tarefa que nos deixou limitações.

Desta forma, podemos concluir que os objectivos a que nos propusemos foram atingidos. Apresentámos uma abordagem SOA sobre um projecto de grande impacto. Expusemos uma arquitectura que respeita este conceito. Mostrámos e apresentámos as tecnologias e metodologias adoptadas durante a sua implementação. Realçámos a componente de integração, estudando-a mais em pormenor de forma a enfatizar a arquitectura distribuída e de integração escolhida.

Além do conhecimento técnico e funcional adquirido, este projecto permitiu conhecer uma realidade para além do contexto académico, trabalhar numa equipa com mais de 10 colaboradores com graus de experiência diferentes, ter contacto com assuntos de gestão de projectos e ter contacto com o cliente. Todos estes factos foram um contributo excepcional para o crescimento profissional e pessoal das pessoas que participaram na realização deste projecto.

6.2 Trabalho Futuro

Este subcapítulo, para além de enunciar os pontos que suscitaram interesse para um desenvolvimento futuro irá também reflectir acerca dos pontos que consideramos serem alvo de alteração.

6.2.1 Geração de Código

Como foi referido anteriormente, alguns problemas foram surgindo à medida que a aplicação crescia e as ferramentas escolhidas mostraram algumas lacunas, as quais incidiam principalmente na falta de controlo sobre as mesmas por parte do utilizador. Tendo em conta que um dos objectivos a atingir com a escolha destas era optimizar o tempo de desenvolvimento, cuidamos que seria interessante num próximo projecto apostar na geração de código, criando um gerador MDA (*Model Driven Arquitecture*) que permita a partir de ficheiros XML gerar cerca de 70% do código (modelo de dados/camada de persistência e camada de negócio).

No contexto empresarial seria uma vantagem, pois seria uma ferramenta adaptável a qualquer projecto, na qual teríamos controlo absoluto sobre esta e que optimizaria os tempos de desenvolvimento dos projectos.

6.2.2 Descartar Hibernate e criar camada de persistência própria

Tendo em conta todos os problemas identificados com a utilização do Hibernate, seria interessante e vantajoso criar a nossa própria ferramenta de persistência, de forma a obter um melhor desempenho no acesso à Base de Dados. Uma ferramenta onde pudéssemos ter controlo sobre as *queries* geradas, mantendo a transparência para o utilizador.

6.2.3 Aprofundar as potencialidades do JCAPS

A ferramenta JCAPS mostra-se ser uma grande potência no que toca ao desenvolvimento de aplicações distribuídas/integradas. Sentimos que é merecedora de um melhor conhecimento e que poderia ser melhor aproveitada para a componente de integração neste projecto. Como foi referido anteriormente, a componente de segurança seria uma boa característica a implementar com base na ferramenta integrada no JCAPS para acesso a aplicações externas.

6.2.4 Melhorar mecanismo de comunicação de mensagens com SDS

Uma vez que o sistema de mensagens é algo muito importante na nossa aplicação, seria interessante desenvolver uma componente administrativa para o controlo deste sistema. Uma pequena aplicação integrada que permitisse ligar/desligar o sistema de comunicação, saber tempos de execução, apresentar registos sobre o processamento de cada mensagem ao utilizador, configurar as mensagens, entre outros pormenores que facilitariam quer quem desenvolve quer o próprio utilizador da aplicação.

6.2.5 Ponto de situação do projecto

Como é sabido, este projecto foi desenvolvido no contexto de um estágio desenvolvido na Indra Sistemas, onde o autor deste documento fez parte da equipa integrante do mesmo. Aquando a realização deste documento, o projecto ainda estava a decorrer, estando cerca de 70% completo. Os próximos passos a efectuar será a realização de uma aplicação de autenticação, o sistema de gestão de acessos e segurança, o sistema de facturação e tarifas, completar o sistema de administração da aplicação e tratar problemas de performance, concorrência, entre outros.

7 Anexos

7.1 Anexo A – Ecrã de Autenticação.

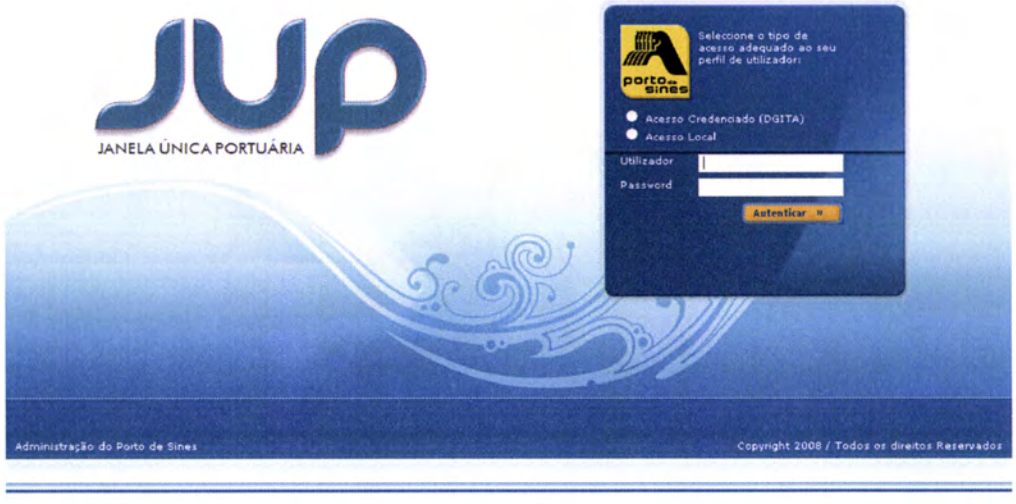


Ilustração 37 – Ecrã de autenticação.

7.2 Anexo B – Ecrã do formulário da criação de um Aviso de Chegada do Navio

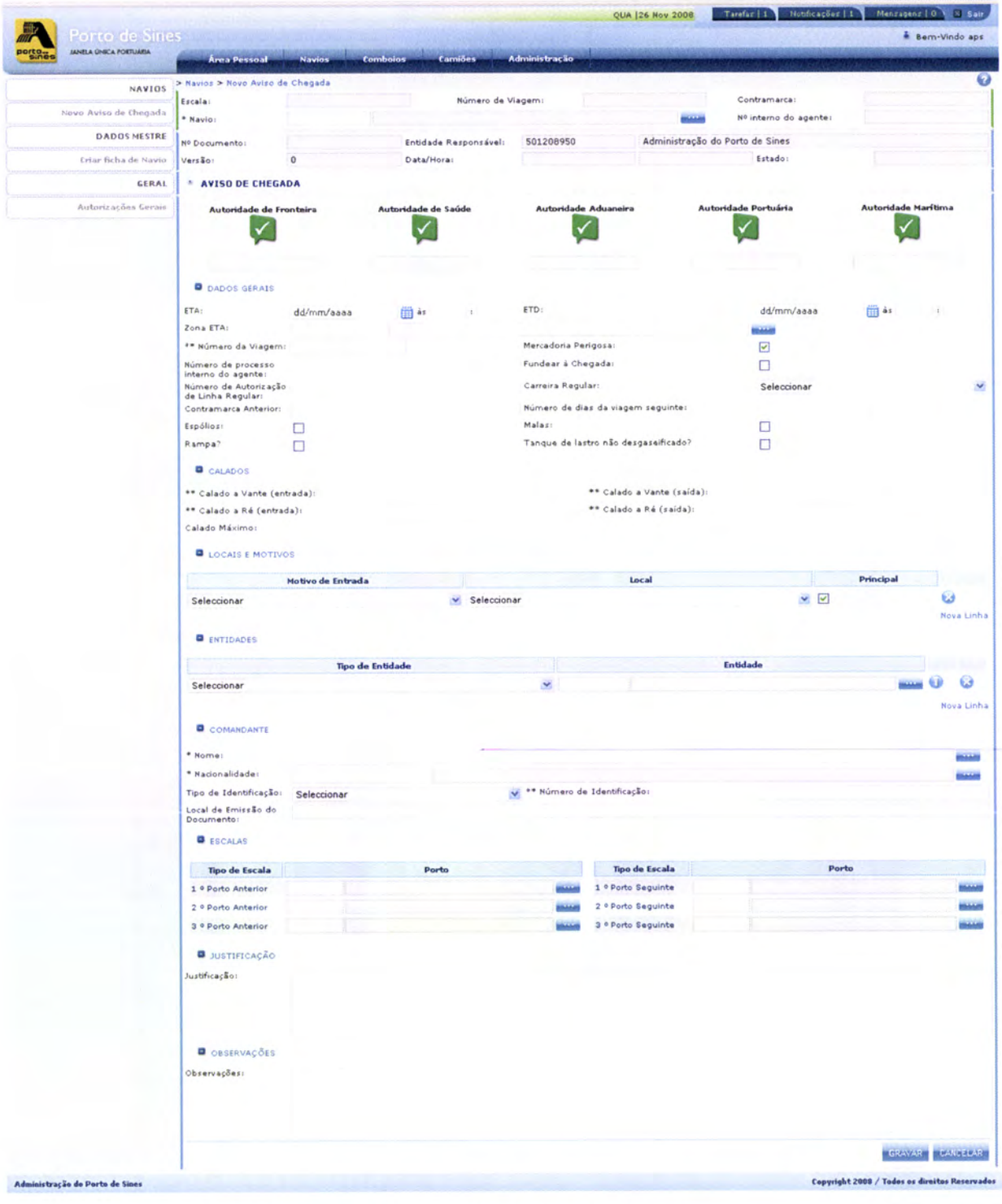


Ilustração 38 – Ecrã do Aviso de Chegada de um Navio.

7.3 Anexo C – Caso de uso do Aviso de Chegada.

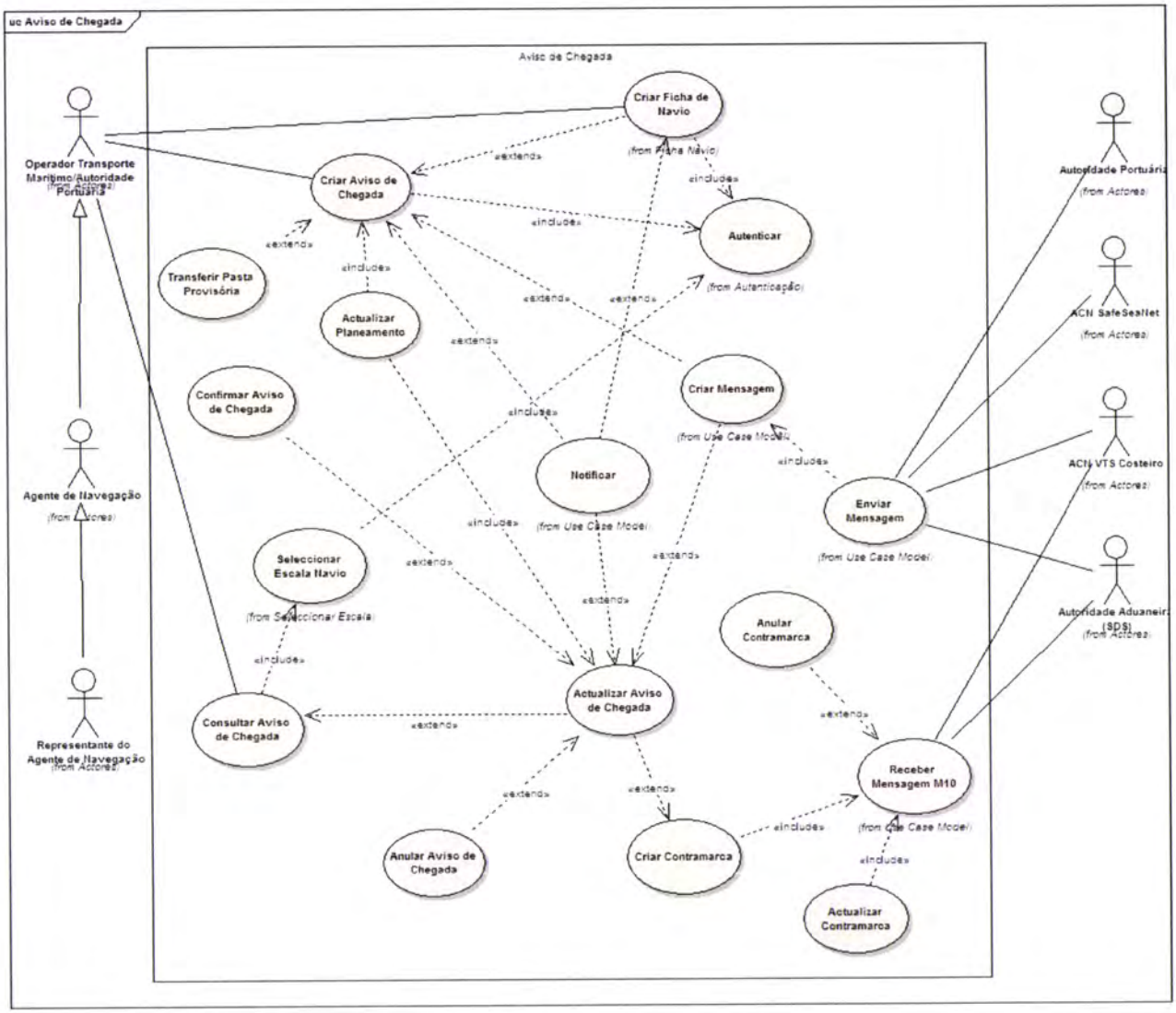


Ilustração 39 – Caso de uso do processo Aviso de Chegada.

7.4 Anexo D – Diagrama de actividades do processo Aviso de Chegada.

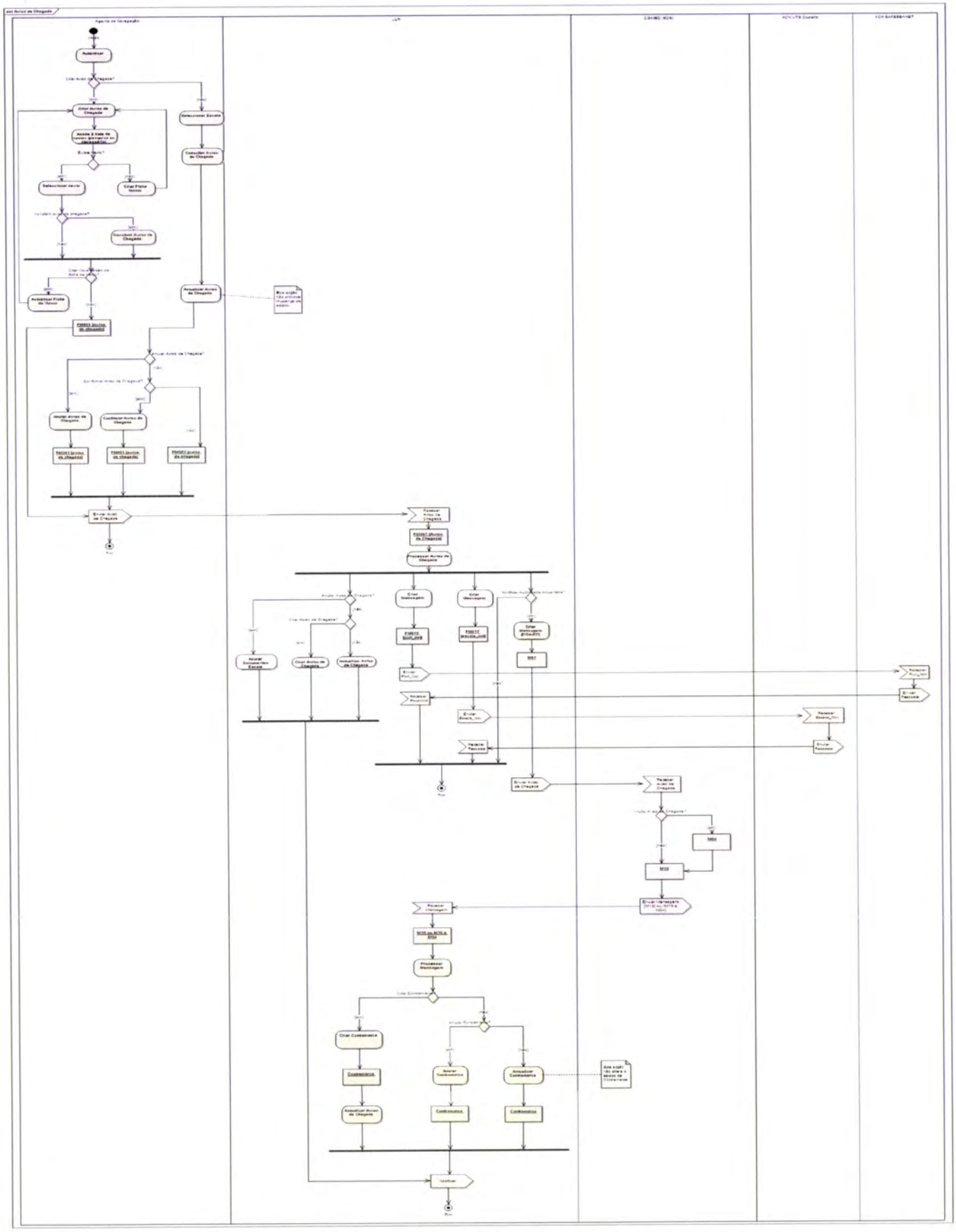


Ilustração 40 – Diagrama de actividades do processo Aviso de Chegada.

7.5 Anexo E – BP do envio da mensagem M1

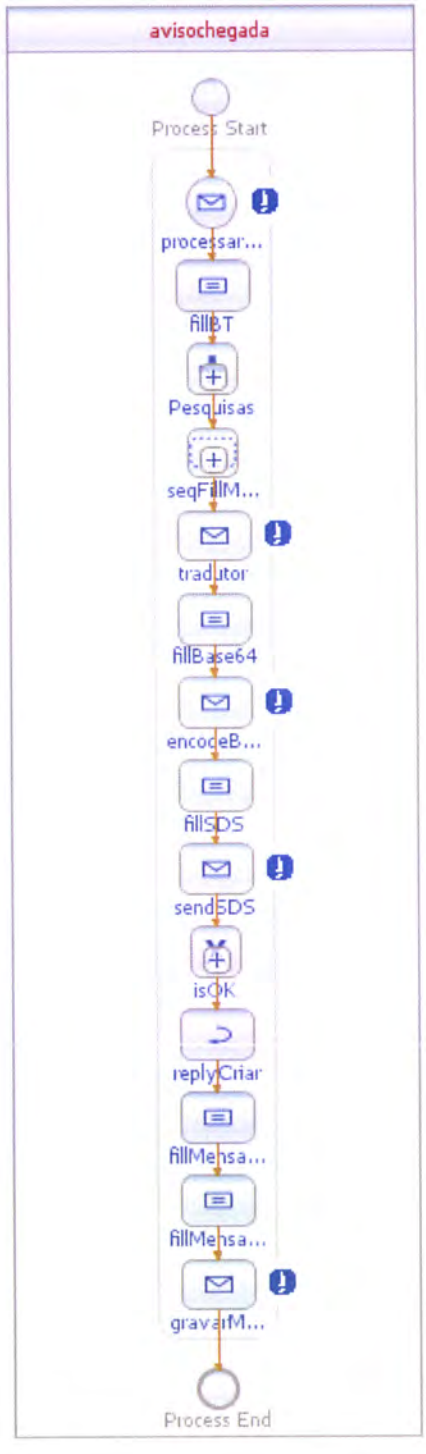


Ilustração 41 – *Business Process* do envio da mensagem M1.

7.6 Anexo F – BP de recepção de mensagens do SDS

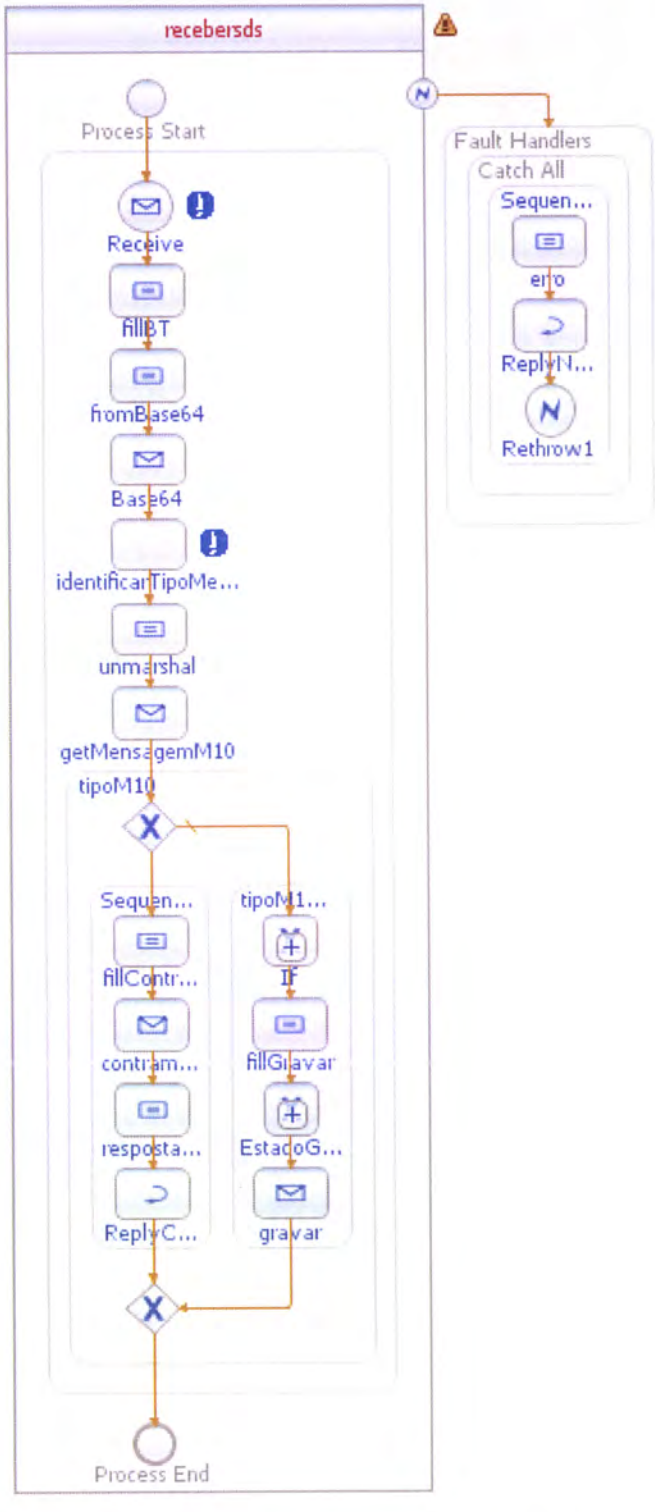


Ilustração 42 – *Business Process* de recepção de mensagens do SDS.

7.7 Anexo G – BP de recepção da Contramarca

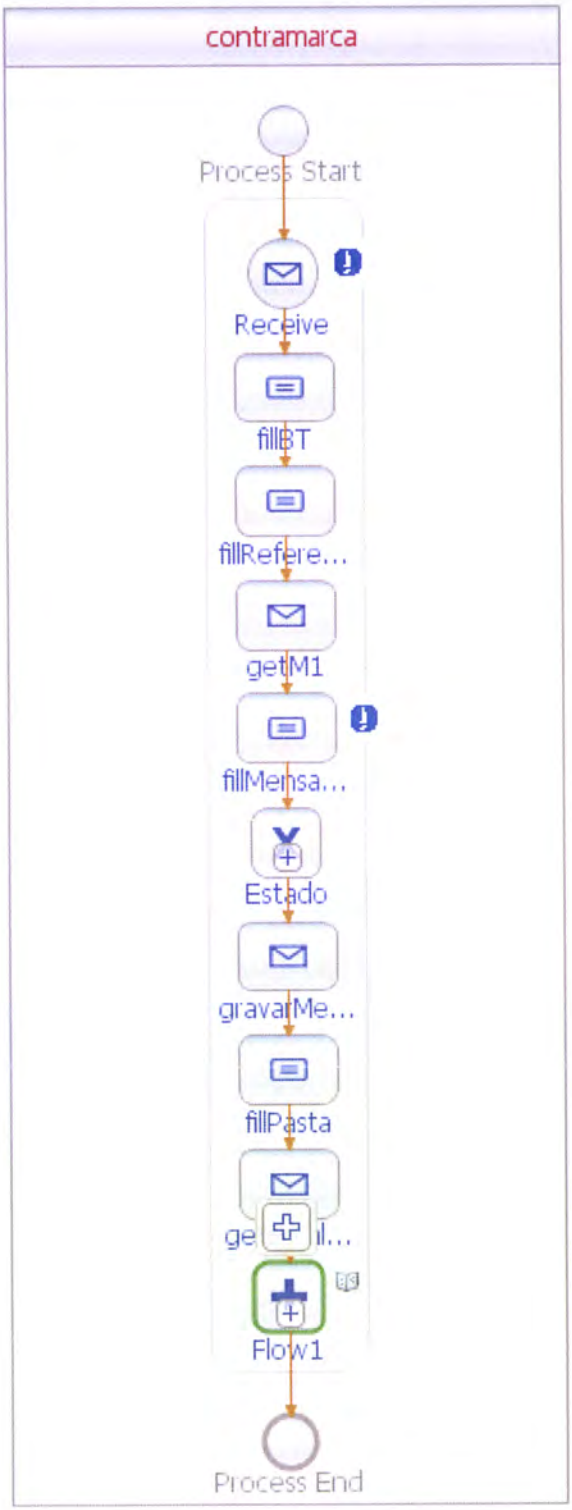


Ilustração 43 – *Business Process* de recepção da contramarca.

8 Referências

Bailliez, S. e outros. 2004. *Apache Ant 1.7.1 Manual,* Apache Ant. [Em linha]. [Acedido: 25 de Novembro de 2008]. Disponível em: http://ant.apache.org/manual/index.html.

Ball, J. Carson, D. Evans, I. Fordin, S. Haase, K. e Jendrock, E. 2006. *The Java™ EE 5 Tutorial - For Sun Java System Application Server Platform Edition 9,* Sun Microsystems, California.

Box, D. Ehnebuske, D. Kakivaya, G. Layman, A. Mendelsohn, M. Nielsen, H. Thatte, S. e Winer, D. 2000. *Simple Object Access Protocol (SOAP) 1.1.* W3C Note 08 May 2000. [Em linha]. [Acedido: 22 de Novembro de 2008]. Disponível em: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

Chappel, D. 2004. *Enterprise Service Bus.* O'Reilly Media, Sebastopol.

Cunha, L. 2007. *Notas Sobre Interoperabilidade & Integração*, Universidade de Évora.

Erl, T. 2004. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services,* Prentice Hall, Upper Saddle River, New Jersey.

Erl, T. 2005. *Service-Oriented Architecture: Concepts, Technology and Design.* Prentice Hall, Saddle River.

Gartner. Natis, Y. 2003. *Service-Oriented Architecture Scenario* [Em linha]. [Acedido: 22 de Julho de 2008]. Disponível em: http://www.gartner.com/DisplayDocument?doc_cd=114358.

Gévaudan, C. 2005. *Service Oriented Architecture*. Tekzoneforum, Zurich.

Gudgin, M. Hadley, M. Mendelsohn, N. Moreau, J. Nielsen, F. Karmarkar, A. e Lafon, Y. 2007. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).* WC3 Recommendation. [Em linha]. [Acedido: 22 de Novembro de 2008]. Disponível em: http://www.w3.org/TR/soap12-part1/.

Hansen, M. 2007. *SOA Using Java Web Services*, Prentice Hall, Indiana.

Iyrngar, A. Jessani, V. e Chilanti, M. 2008. *WebSphere Business Integration Primer – Process Server, BPEL, SCA and SOA*, IBM Press, California.

Jordan, D. e Evdemon, J. 2006. *Web Services Business Process Execution Language Version 2.0*, OASIS. [Em linha]. [Acedido: 23 de Novembro de 2008]. Disponível em: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html.

Josuttis, N. 2007. *SOA in Practice*, 1st Edition, O'Reilly Media, Sebastopol.

Kinnumpurath, M. 2005. JBI – *A Standard-based approach for SOA in Java*, The Server Side. [Em linha]. [Acedido: 15 de Novembro de 2008]. Disponível em: http://www.theserverside.com/tt/articles/article.tss?l=JBIforSOA.

Larman, C. 2002. *Applying UML and Patterns - An Introduction to Object-oriented Analysis and Design*, Prentice-Hall, New Jersey.

Linthicum, D. 2004. *Next Generation Application Integration*, Addison Wesley, Pearson Education, Massachusetts, USA.

Linthicum, D. 2001. *B2B Application Integration*, Addison-Wesley, Massachusetts, USA.

Mann, K. 2005. *Java Server Faces in Action*, Manning, Greenwich.

Newcomer, E. e Lomow, G. 2005. *Understanding SOA with Web Services*, Addison Wesley Professional, Hagerstown, Maryland.

OMG. 2007. *Unified Modeling Language (OMG UML)*, Superstructure. [Em linha]. [Acedido: 6 de Dezembro de 2008]. Disponível em:

http://www.omg.org/docs/formal/07-11-02.pdf

OpenESB. 2008. *About OpenESB*. [Em linha]. [Acedido: 18 de Novembro de 2008]. Disponível em:

https://open-esb.dev.java.net/AboutOpenEsb.html.

Panda, D. Rahman, R. e Lane, D. 2007. *EJB3 in Action*, Manning, Greenwich.

Pelegri-Llopart, E. Yoshida, Y. Moussine-Pouchkine, A. 2007. *The GlassFish Community Delivering a Java EE Application Server,* The GlassFish Community.

Raj, G. PG, B. Babo, K. e Palkovic, R. 2006. *Implementing Service-Oriented Architectures (SOA) with the Java EE 5 SDK*, Sun Microsystems, California.

Salter, D. e Jennings, F. 2008. *Building SOA-Based Composite Applications Using NetBeans IDE 6*, Packt Publishing, Birmingham.

Sun Microsystems. 2008. *Sun Java Composite Application Platform Suite - Flexible, modular, open source service-oriented architecture integration*, Sun Microsystems, Santa Clara.

Sun Microsystems. 2008a. *Sun Java Composite Application Platform Suite (Java CAPS)* [Em linha]. [Acedido: 16 de Outubro de 2008]. Disponível em: http://www.sun.com/software/javaenterprisesystem/javacaps/index.jsp

Sun Microsystems. 2008b. *Service-Oriented Architecture (SOA)* [Em linha]. [Acedido: 5 de Novembro de 2008]. Disponível em:

http://www.sun.com/products/soa/index.jsp

Tanenbaum, A. 1994. *Distributed Operating Systems (Paperback)*, US Ed edition, Prentice Hall.

Ten-Hove, R. 2006. *Using JBI for Service-Oriented Integration (SOI)*, Sun Microsystems.