Universidade do Porto
Faculdade de Engenharia
FEUP



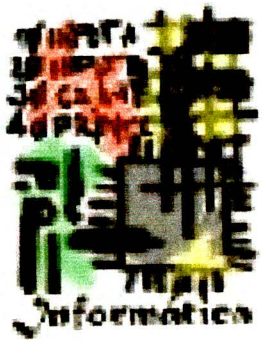
Rúben Joás Pereira

Contact Center - Linha Branca na CPC - Tecnologias Avançadas

LEIC
/PERr

LEIC
Porto, 2002

CONFIDENCIAL



Contact Center – Linha Branca na CPC Tecnologias Avançadas

Rúben Joás da Silva Pereira

Relatório do Estágio Curricular da LEIC 2001/02

Orientador na FEUP: Prof. António Carvalho Brito

Orientador na CPC TA: Dr. Jorge Alves



Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

2002-09-05

UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA
N.º 6532 1
CDU 004(047.3)
Data 14 / 3 / 2003

Resumo

O presente relatório documenta a especificação e implementação de um sistema de Contact Center genérico, denominado Linha Branca. O projecto em questão decorreu na Companhia Portuguesa de Computadores – Tecnologias Avançadas (CPC TA), no âmbito do estágio curricular da Licenciatura em Engenharia Informática e Computação, da Faculdade de Engenharia da Universidade do Porto.

Os Contact Centers da CPC TA disponibilizam várias aplicações para diversos níveis de gestão de contactos multi-canal efectuados com clientes de uma organização. A Linha Branca corresponde a um Contact Center com as funcionalidades mais comuns, usada como base de desenvolvimento de Contact Centers nas instituições, e também como sistema de demonstrações para potenciais clientes. Assim como grande parte das aplicações da CPC TA, o Contact Center é suportado pela plataforma WIS (*Web Information System*).

A WIS é uma plataforma *middleware* multi-camada criada para permitir um desenvolvimento mais rápido e estruturado de aplicações de grande dimensão, garantindo segurança, performance e integração de serviços de *Internet* e *Intranet*. De modo a acompanhar as mais recentes evoluções tecnológicas, a plataforma WIS está a ser remodelada, dando origem à WIS2.

Dada a necessidade de adaptar as aplicações como o Contact Center à nova arquitectura WIS2, foi decidido efectuar uma também um remodelação do Contact Center – Linha Branca para incluir novas funcionalidades e uma melhor usabilidade e estruturação geral. Contudo, dado ainda não existir uma versão completa da WIS2, a nova Linha Branca teve como base uma versão melhorada da plataforma WIS original, de modo a facilitar a posterior migração para a WIS2. Para o efeito foram especificados os requisitos funcionais e não funcionais para o Contact Center a implementar.

O desenvolvimento da Linha Branca incluiu uma fase inicial de análise e definição das metodologias a utilizar, seguindo-se a produção de estruturas de suporte ao desenvolvimento, e finalmente a implementação de funcionalidades requisitadas.

A nova Linha Branca produzida contem uma melhor estruturação, e está mais congruente com os standards tecnológicos da actualidade, facilitando a eficácia do desenvolvimento. A nível funcional são disponibilizadas novas opções, oferecendo uma melhor usabilidade e interfaces apropriados.

Agradecimentos

Antes de mais, gostaria de agradecer à CPC TA pela oportunidade concedida, aos meus colegas de trabalho, pelo bom ambiente que criaram, e em especial a Rui Rocha e ao Dr. Jorge Alves, por todo o apoio e orientação.

Os meus agradecimentos a todos os meus colegas e professores da LEIC, que me acompanharam e contribuíram para a minha formação em Engenharia Informática. Agradeço também o tempo e apoio dispensado pelo Prof. António Carvalho Brito, orientador do estágio por parte da FEUP.

Um agradecimento especial para os meus amigos e família por todo o apoio que me deram.

Índice de Conteúdos

1 Introdução 1

1.1 Apresentação da CPC Tecnologias Avançadas *1*

1.2 O Projecto Contact Center – Linha Branca *2*

1.3 Enquadramento do estágio no Projecto Linha Branca *3*

1.4 Organização e Estrutura do Relatório *3*

2 Contact Center 5

2.1 Contextualização *5*

2.2 Modelo Funcional *6*

2.3 Arquitectura *9*

2.4 A Plataforma WIS *11*

3 Apresentação do Problema 12

3.1 Contextualização *12*

3.2 Especificação de Requisitos *13*

3.2.1 Requisitos Funcionais 13

3.2.2 Requisitos Não Funcionais 30

4 Desenvolvimento do Contact Center – Linha Branca 34

4.1 Modelo de Desenvolvimento *34*

4.2 Documentação *35*

4.3 Arquitectura *35*

4.3.1 Sistema de Gestão de Base de Dados 36

4.3.2 Camada Transaccional 42

4.3.3 Camada de Apresentação 42

4.3.4 Estrutura de Directórios 44

4.4 Implementação *46*

4.4.1 Ambiente de Desenvolvimento 46

4.4.2 Aplication Programming Interface 46

4.4.3 Funcionalidades 48

5 Conclusões e Perspectivas de Trabalho Futuro 49

5.1 Conclusões do Trabalho Realizado *49*

5.2 Perspectivas de Trabalho Futuro *49*

Referências e Bibliografia 51

ANEXO A: Glossário 52

ANEXO B: Plataforma WIS 54

B.1 Introdução *54*

B.2 Arquitectura *55*

B.2.1 Sistema de Gestão de Base de Dados 55

B.2.2 Camada Transaccional 56

B.2.3 Camada de Apresentação – Session Manager 57

B.2.4 Camada de Apresentação – Presentation Manager 58

B.3 Evolução para a WIS2 ..... *58*

ANEXO C: Nova Linha Branca Vs Antiga Linha Branca 59
C.1 Camada Transaccional *59*
C.2 Session Manager *59*
C.2.1 Exemplo de Código Fonte da Antiga Linha Branca 60
C.2.2 Exemplo de Código Fonte da Nova Linha Branca 61
C.3 Presentation Manager *63*
C.3.1 Exemplo de Código Fonte da Antiga Linha Branca 65
C.3.2 Exemplo de Código Fonte da Nova Linha Branca 67
C.4 Interface com o Utilizador *68*
ANEXO D: Guia do Programador 72

Índice de Imagens

Figura 1.1 - Organigrama da CPC IS / SGPS em 2001 1
Figura 1.2 – Arquitectura da Plataforma WIS 3
Figura 2.1 – Fluxos de Informação entre Cliente, Agente e *BackOffice.* 6
Figura 2.2 – Arquitectura Contact Center 10
Figura 3.1 – Caso de Uso: Contact Center 14
Figura 3.2 – Caso de Uso: *Info Center* 16
Figura 3.3 – Caso de Uso: *Agent Desktop* (visão global) 18
Figura 3.4 – Caso de Uso: *Agent Desktop* – Comum 19
Figura 3.5 – Caso de Uso: *Agent Desktop* – Cliente Identificado 21
Figura 3.6 – Caso de Uso: *BackOffice* 22
Figura 3.7 – Caso de Uso: *Supervisor Desktop* 24
Figura 3.8 – Caso de Uso: *Administration Center* 26
Figura 3.9 – Caso de Uso: Simulação de Telefonia 28
Figura 3.10 – Esboço do *Info Desk* 31
Figura 3.11 – Esboço Geral das aplicações 32
Figura 3.12 – Esboço do *Agent Desktop* (com cliente identificado) 32
Figura 3.13 – *Applet* para o controlo de telefonia 33
Figura 4.1 – Modelo de Desenvolvimento Evolutivo 34
Figura 4.2 – Diagrama ER: Base de Dados WIS 36
Figura 4.3 – Diagrama ER: Base de Dados CONTACTOS 38
Figura 4.4 – Diagrama ER: Base de Dados PUBLISHING 39
Figura 4.5 – Diagrama ER: Base de Dados DOCUMENTOS 39
Figura 4.6 – Diagrama ER: Base de Dados CONTACTCENTREEMARKETING - SEGMENTOS 40
Figura 4.7 – Diagrama ER: Base de Dados CONTACTCENTREEMARKETING - CAMPANHAS 41
Figura 4.8 – Diagrama ER: Base de Dados CONTACTCENTRELINHABRANCA 42
Figura 4.9 – Diagrama de Sequenciação: Invocação de um Serviço 44
Figura B.1 – Aplicações da Plataforma WIS 54
Figura B.2 – Arquitectura Base da Plataforma WIS 55
Figura C.1 – Diagrama de Sequenciação: Invocação de um Serviço 64
Figura C.2 – Antiga Linha Branca: *Info Desk* 69
Figura C.3 – Nova Linha Branca: *Info Desk* 69
Figura C.4 – Antiga Linha Branca: *Agent Desktop* 70
Figura C.5 – Nova Linha Branca: *Agent Desktop* 70
Figura C.6 – Antiga Linha Branca: *Supervisor Desktop* 71
Figura C.7 – Nova Linha Branca: *Supervisor Desktop* 71

1 Introdução

Nesta secção é descrito o contexto do estágio, englobando uma apresentação da empresa onde este decorreu, é introduzido o projecto, os seus objectivos e enquadramento no estágio. Na parte final é dada uma visão geral da organização e estrutura do relatório.

1.1 Apresentação da CPC Tecnologias Avançadas

A CPC TA nasceu da restruturação da CPC IS (Companhia Portuguesa de Computadores, Informática e Sistemas). A CPC IS actua no mercado Português desde 1989, possuindo um volume de negócios acima dos 50 milhões de Euros. No início de 2001 a CPC IS subdividiu-se em várias empresas, ficando com a estrutura representada na *Figura 1.1*.

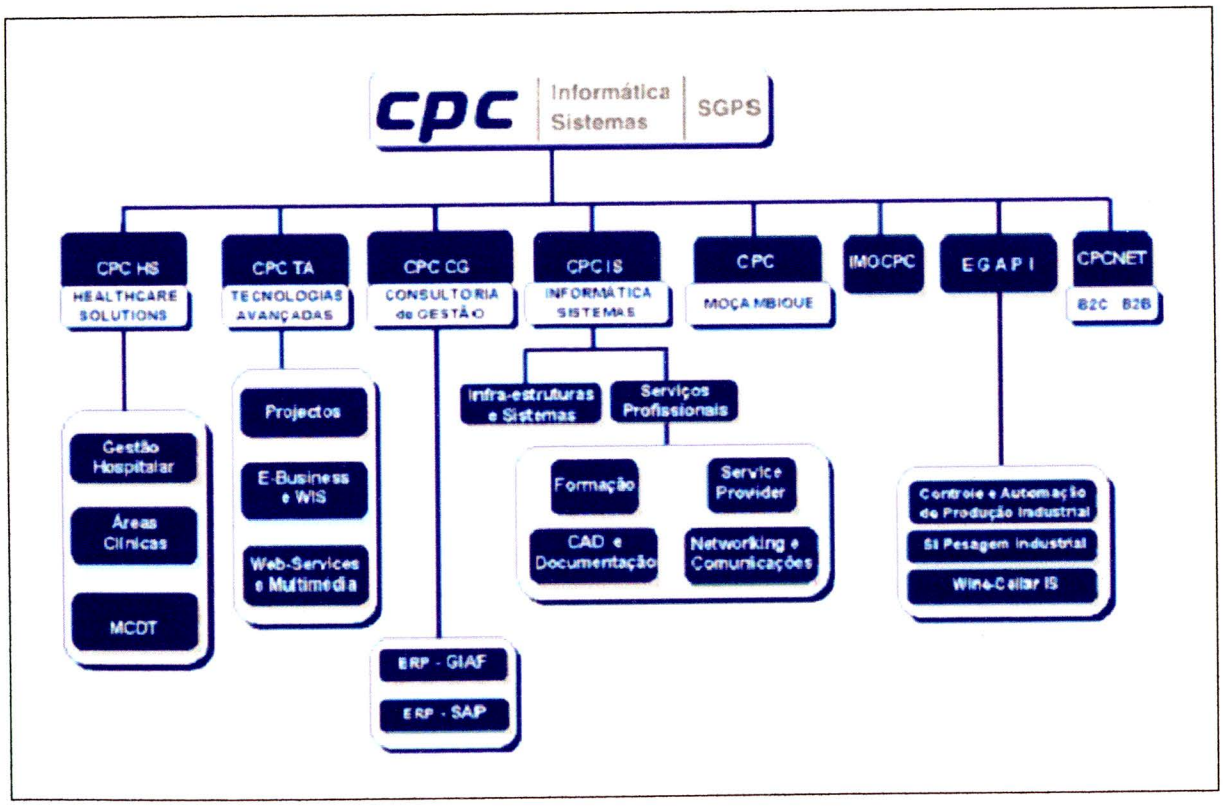


Figura 1.1 - Organigrama da CPC IS / SGPS em 2001

O grupo CPC tem mais de 500 colaboradores em escritórios no Porto, Lisboa, Braga, Coimbra e Maputo. As áreas de negócio são bastante diversificadas no que diz respeito a Tecnologias de Informação (TIs), desde comercialização de *hardware* e infra-estruturas, desenvolvimento e integração de *software* 'à medida', serviços de consultoria, formação e apoio ao cliente. A CPC tem parcerias com os líderes mundiais de soluções em informática, possuindo contratos com Hewlett Packard, Compaq, IBM, Oracle, Microsoft, SAP e Autodesk CAD. Estando ainda credenciada na Central de Compras do Estado para venda de equipamento Hewlett Packard e Compaq. Deste modo a CPC pode oferecer um leque alargado de soluções integradas, satisfazendo praticamente todos os requisitos do cliente, desde as aplicações necessárias até aos servidores, *hardware* e infra-estruturas.

Na área de *software*, o principal objectivo é o desenvolvimento de soluções em plataformas abertas, podendo ser implementadas em diversos sistemas operativos, usando vários sistemas de gestão de Bases de Dados (SGBD). Existe ainda uma vasta gama de soluções de gestão, incluindo um ERP (*Enterprise Resource Planning*) desenvolvido internamente, implementado em mais de uma centena de médias e grandes empresas.

A CPC TA enquadra-se no desenvolvimento de sistemas de informação, predominantemente com orientação *Web*. Para tal é normalmente utilizada a plataforma WIS (*Web Information System*). A WIS é uma plataforma *Middleware*, criada pela CPC, que nos últimos anos tem sido a principal base de desenvolvimento de sistemas 'à medida' do cliente.

No rol de clientes e projectos da CPC TA destaca-se: Banco7, ActivoBanco7, CidadeBCP, NovaRede, Atlântico, BCP, Montepio Geral (Phone24), Finibanco, TAP, PT, BPI, Interbanco, etc.

Mais recentemente, as empresas CPC TA e CPC CG participaram numa *joint venture* com a Indra, líder espanhol em Tecnologias de Informação, presente em mais de 40 países. Prevê-se que a nova empresa Indra-CPC, seja um dos maiores fornecedores de serviços de TI em Portugal, com um volume de facturação previsto acima dos 30 milhões de Euros.

1.2 O Projecto Contact Center – Linha Branca

O projecto descrito neste relatório diz respeito ao desenvolvimento de uma nova Linha Branca do Contact Center.

O Contact Center é um sistema para gerir os contactos (telefone, fax, correio, *email*, etc) efectuados com clientes, por parte de uma organização. Este sistema é um integrado de aplicações com interface *Web* para Agentes (que contactam com os clientes), *BackOffice*, *Help Desk*, Supervisores, Administração, entre outras. Para permitir uma mais fácil e rápida implementação do sistema nas instituições, foi desenvolvida uma Linha Branca do Contact Center, contendo as funcionalidades mais usuais, de modo a ser apenas necessário adaptar o *design* e outras funcionalidades extras requeridas pela organização. A Linha Branca é também usada para demonstrar as funcionalidades do Contact Center da CPC TA a potenciais clientes.

O Contact Center, assim como grande parte das aplicações da CPC TA, tem como base a plataforma WIS. Esta plataforma, também desenvolvida na CPC TA, disponibiliza uma arquitectura multi-camada para o desenvolvimento de aplicações distribuídas de grande porte, permitindo balanceamento de cargas, *caching* de ligações a Bases de Dados, sistemas de *logs*, etc.

Dada a necessidade de reestruturar o actual Contact Center, de modo a acrescentar novas funcionalidades e melhorar a usabilidade geral, tornou-se necessário a implementação de uma nova Linha Branca. Devido à prevista reestruturação da plataforma WIS (que irá dar origem à WIS2), a nova Linha Branca deverá estar preparada para as novas metodologias e APIs (*Application Programming Interface*) da futura WIS2.

Este novo Contact Center será baseado na antiga Linha Branca e no Contact Center recentemente implementado no Interbanco, contendo contudo novas funcionalidades, melhor usabilidade, mais facilidades de adaptação e desenvolvimento, garantindo a compatibilidade com as especificações da actual plataforma WIS, mas preparando desde já a migração do Contact Center para a WIS2.

1.3 Enquadramento do estágio no Projecto Linha Branca

O objectivo principal do estágio curricular visa a participação na análise, definição de arquitectura e desenvolvimento da nova Linha Branca do Contact Center.

A arquitectura geral do Contact Center é definida pela plataforma WIS. Contudo, existem várias possibilidades de estruturação que devem ser devidamente analisadas de modo a optimizar a eficiência do sistema e, sendo a Linha Branca um sistema base para o desenvolvimento de futuros Contact Centers, é crucial a uma boa estruturação de modo a facilitar a implementação, re-implementação e expansão de novos Contact Centers.

A fase de implementação deve ser efectuada em conjunto com a equipa de desenvolvimento do Contact Center na CPC TA, tendo a plataforma WIS como suporte. A arquitectura da WIS (representada na *Figura 1.2*) contem por três camadas principais (*3-tier*), sendo as tecnologias base a utilizar: *Java Server Pages*, *Java / JavaBeans*, *Corba* e Base de Dados *Microsoft SQL Server 2000*. Os interfaces com o utilizador devem ter orientação *Web*, optimizados para o *Microsoft Internet Explorer*.

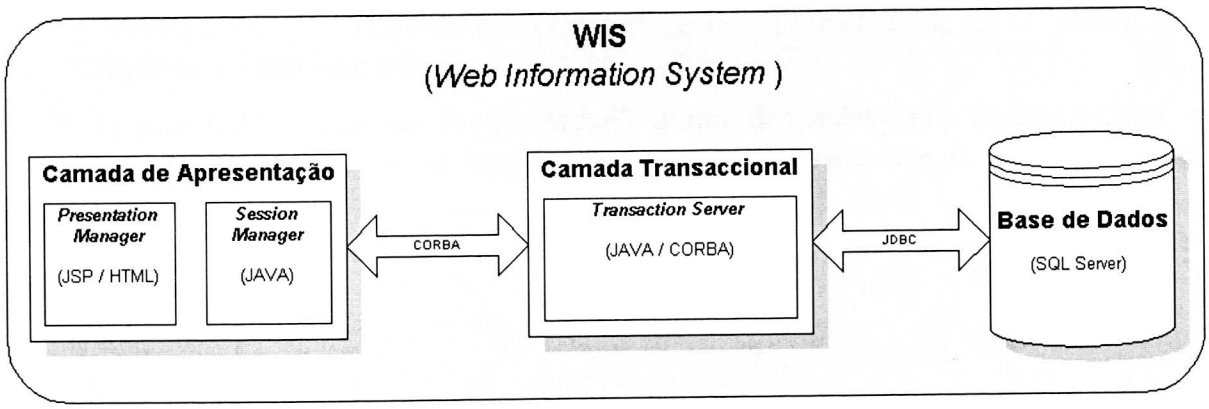


Figura 1.2 – Arquitectura da Plataforma WIS

Dada a dimensão do Contact Center, o seu desenvolvimento está previsto ultrapassar o tempo de realização do estágio. Contudo, no âmbito do estágio curricular, deverá ser implementado todo o sistema de base para desenvolvimento, assim como as principais funcionalidades requisitadas.

1.4 Organização e Estrutura do Relatório

O presente documento encontra-se dividido em 5 capítulos, complementados por 4 anexos:

- No Capítulo 1, "*Introdução*", é apresentada a empresa onde decorreu o estágio, e são dadas umas noções introdutórias sobre o Contact Center e os objectivos e enquadramento do projecto de estágio. Na parte final é dada uma visão geral da organização e estrutura do relatório.
- O Capítulo 2, "*Contact Center*", pretende familiarizar o leitor com os conceitos, funcionalidades e estruturação do Contact Center.
- No Capítulo 3, "*Apresentação do Problema*", é descrito em pormenor o problema proposto, referindo os motivos que levaram à necessidade da criação de uma nova

Linha Branca do Contact Center. São especificados os principais requisitos funcionais e não funcionais do sistema.

- O Capítulo 4, "*Desenvolvimento do Contact Center – Linha Branca*", retracta as fases mais relevantes do processo de desenvolvimento da Linha Branca, descrevendo a arquitectura, opções, algoritmos e métodos utilizados.
- No Capítulo 5, "*Conclusões e Perspectivas de Trabalho Futuro*", são mencionadas algumas considerações sobre o grau de conformidade entre os objectivos iniciais e os resultados obtidos, apresentando também algumas possibilidades de melhoramentos e expansões que o projecto pode tomar no futuro.
- O Anexo A, "*Glossário*", contém a definição de alguns dos termos e siglas utilizados ao longo do relatório e que não foram devidamente explicados na altura do seu uso.
- No Anexo B, "*Plataforma WIS*", são explicados os conceitos, arquitectura e características principais do *middleware* utilizado para o desenvolvimento de aplicações na CPC TA.
- O Anexo C, "*Nova Linha Branca Vs Antiga Linha Branca*", exemplifica as principais diferenças entre as duas versões do Contact Center – Linha Branca, em diversos níveis funcionais e não funcionais.
- O Anexo D, "*Guia do Programador*", é um documento que define normas e metodologias a utilizar na implementação da Linha Branca e de futuros Contact Centers.

2 Contact Center

Este capitulo são apresentados os principais conceitos do Contact Center da CPC TA. O objectivo é apresentar as funcionalidades mais comuns de um Contact Center genérico, e explicar a sua arquitectura habitual.

2.1 Contextualização

Actualmente caminhamos para uma sociedade onde a tecnologia consegue encurtar a distância entre as organizações e o consumidor. De tal modo que o cliente praticamente não necessita de se deslocar às empresas, podendo efectuar as operações que deseja com um simples telefonema, *email*, ou através de um acesso pela *internet*. Desta forma as organizações estão acessíveis ao cliente onde quer que ele esteja, podendo satisfazer as necessidades do cliente de forma mais rápida e eficaz, traduzindo-se numa vantagem competitiva para as empresas. Esta filosofia tem sido largamente aproveitada pelas instituições financeiras, bancos, seguradoras, etc, que têm vindo a evoluir do atendimento pessoal com as longas filas de espera, para novos canais de atendimento. Este movimento foi iniciado com o aparecimento de máquinas de atendimento automático (Multibanco/ATM), evoluindo para soluções de *Home* e *Corporate Banking*.

Contudo nenhum destes sistemas oferece as mesmas funcionalidades do balcão de um banco. Devido à complexidade de certas operações, muitos clientes necessitam de apoio personalizado por parte de profissionais do sector. Inicialmente foram criadas linhas de apoio ao cliente, contudo estas são limitadas pois os assistentes das linhas de apoio tradicionais não possuem infra-estruturas para executar as operações que o cliente pretenda de forma eficiente. Surge então a necessidade de criar um sistema capaz de disponibilizar informação e funcionalidades para satisfazer os requisitos do cliente e gerir os contactos realizados através dos mais diversos canais – um Contact Center.

Dada a necessidade, especialmente no sector financeiro, para a implementação de Contact Centers, empresas de TI como a CPC resolveram aproveitar a abertura do mercado para este tipo de aplicações e dedicaram alguns dos seus recursos à produção e desenvolvimento de Contact Centers.

A CPC TA tinha já uma vasta gama de aplicações (Portais, *Home Banking*, Corretoras *On-Line*, etc) desenvolvidas principalmente para o sector financeiro, possuindo um rol considerável de clientes nesta área. O impulso foi dado com o Contact Center desenvolvido em 2000 para o Montepio Geral (*Phone24*), onde se encontra em pleno funcionamento. Apostando na produção de sistemas deste género, o grupo de Contact Centers da CPC TA construiu um Contact Center - Linha Branca. Esta Linha Branca consiste num Contact Center genérico com o objectivo de se poder adaptar facilmente aos requisitos dos clientes, diminuindo consideravelmente o tempo de implementação do sistema. A Linha Branca serve ainda para demonstrar as principais funcionalidades de um Contact Center a potenciais compradores.

2.2 Modelo Funcional

Um sistema de Contact Center numa instituição implica a alocação de vários recursos para a utilização e administração do sistema. A estruturação e recursos alocados ao Contact Center são adaptados às necessidades e possibilidades de cada empresa, contudo, os intervenientes usuais num Contact Center são:

Agentes – geralmente operadores telefónicos que comunicam directamente com os clientes.

Operadores de *BackOffice* – analisam e executam pedidos e operações.

Supervisores – definem parâmetros relativos ao funcionamento das operações do negócio, e supervisionam operações de Agentes e *BackOffice*.

Administradores – responsáveis pela manutenção de aspectos gerais da aplicação, não relacionados directamente com a lógica de negócio.

No caso de uso mais comum, um Cliente telefona para uma linha da instituição. Um operador telefónico (Agente) atende a chamada e fornece ao Cliente informações gerais que tenha disponíveis. Caso o Cliente deseje que sejam executadas operações que não possam ser executadas em tempo útil ou que não estejam disponíveis para o Agente, este regista o pedido e envia-o para *BackOffice*. Os operadores de *BackOffice* analisam os pedidos recebidos e executam as operações pretendidas. Se necessário os operadores de *BackOffice* podem indicar ao Agente para recontactar o Cliente para fornecer ou requisitar informações adicionais sobre os pedidos efectuados. Este é a utilização mais comum de um Contact Center e está representada na *Figura 2.1*.



Figura 2.1 – Fluxos de Informação entre Cliente, Agente e *BackOffice*.

Um exemplo possível numa instituição bancária poderia ser algo do género:

- *O Cliente liga para a instituição bancária e pede informações sobre o saldo e movimentos da sua conta.*
- *O Agente fornece as informações da conta do Cliente (após um processo de autenticação do Cliente).*
- *De seguida o cliente pede para abrir uma nova conta.*
- *Após obter os dados necessários o Agente envia o pedido para BackOffice.*
- *Um operador de BackOffice recebe o pedido e detecta discrepâncias nos dados. Envia ao Agente uma indicação para recontactar o Cliente.*
- *O Agente liga para o Cliente para o inquirir sobre as duvidas levantadas. Após receber o esclarecimento do Cliente, o Agente volta a enviar os dados para BackOffice.*
- *O operador de BackOffice abre a nova conta do Cliente.*

Existem ainda outras funcionalidades para a interacção com o cliente disponíveis para o Agente e *BackOffice*. O Agente pode ainda tentar interessar o Cliente em campanhas para aquisição de produtos ou serviços da instituição. Pode também agendar um contacto para outra altura mais conveniente. No *BackOffice*, os operadores podem também registar o envio de documentos para o Cliente (contratos de adesão, extractos de conta, etc). No Contact Center da CPC TA são também suportados outros canais de comunicação para além da telefonia, como *email*, correio e fax.

Os Supervisores, para além de monitorizarem as operações dos Agentes, são também responsáveis pela definição de certos parâmetros relativos à lógica do negócio (por exemplo, definição de campanhas activas, criação de *templates*, alteração de agendas de Agentes, etc). Ocasionalmente os Supervisores também podem entrar em contacto com clientes, mas não directamente. Caso se justifique, um Agente pode transferir uma chamada para um dos seus Supervisores.

O Administrador do Contact Center é responsável pela manutenção dos aspectos gerais da aplicação que não estejam directamente ligados com as operações do negócio. O Administrador gere conteúdos disponíveis para os utilizadores das aplicações relativos notícias, *links*, publicidade, etc. É também responsável por outros aspectos cruciais para o funcionamento do sistema, como a manutenção utilizadores, tabelas, segmentos, etc.

O Contact Center da CPC TA é constituído por várias aplicações, integradas entre si, orientadas para os diferentes perfis de utilizadores nas instituições. Seguidamente são descritas as principais funcionalidades de cada aplicação do Contact Center genérico (Linha Branca):

- **Info Center**

O *Info Center* funciona como um portal de entrada para todos os utilizadores do Contact Center. Este portal de informação dispõe vários conteúdos como notícias, apontadores (*links*), publicidade, etc. A manutenção destes conteúdos é efectuado na aplicação de Administração.

A partir do *Info Center* um utilizador pode aceder à aplicação apropriada para a sua função, passando por um processo de autenticação do utilizador.

- **Agent Desktop**

Esta é a aplicação base para os Agentes, disponibilizando os serviços necessários para a comunicação com o cliente. O *Agent Desktop* está integrado com o sistema de telefonia, permitindo ao operador o controlo de várias funções da telefonia (atender chamadas, marcar números, transferir chamadas, etc) através duma *applet* integrada no sistema. A aplicação tem dois modos de operação.

No primeiro modo o Agente não está em contacto com nenhum cliente, ou o cliente não é conhecido. Neste modo o Agente pode, entre outras coisas, verificar a sua agenda, consultar os pedidos efectuados ou registar pedidos de clientes que ainda não existam na Base de Dados.

Ao autenticar um cliente, o Agente dispõe de novas opções relativas ao cliente com que está em contacto. Neste modo de operação, a aplicação disponibiliza vários dados do cliente relativos ao negócio da instituição (por exemplo: dados de contas, saldos, informações do cliente, etc). O Agente pode também consultar dados do cliente relativos a contactos anteriores, assuntos agendados e pedidos efectuados. Outra funcionalidade integrada no *Agent Desktop* é o *CrossSelling*. Certos clientes podem ser alvos de campanhas para a aquisição de produtos ou serviços da instituição. A aplicação indica ao Agente quais as campanhas (*InBound*) destinadas ao cliente identificado, para que o Agente possa tentar aliciar o cliente a aderir aos produtos ou serviços fornecidos (através da aplicação de *eMarketing*).

- **BackOffice**

O *BackOffice* tem como objectivo o tratamento de pedidos que não possam ser executados de forma automática sem a supervisão de um profissional qualificado. Para cada pedido, a aplicação permite que o operador de *BackOffice* o aceite, rejeite ou adie. Pode ainda inserir na agenda do Agente uma indicação para voltar a contactar o cliente de modo a resolver qualquer problema encontrado no pedido.

Esta aplicação também disponibiliza o envio de documentos por correio ou fax.

- **Supervisor Desktop**

A aplicação destinada aos Supervisores permite a monitorização e intervenção na gestão de clientes. É possível visualizar diversos dados relativos às operações efectuadas nas aplicações de interacção com o cliente (*Agent Desktop*, *eMarketing* e *Help Desk*). Está também presente nesta aplicação a *applet* de controlo de telefonia, podendo os Supervisores receber chamadas eventualmente transferidas pelos Agentes.

Para o tratamento de informação de apoio ao negócio, o *Supervisor Desktop* disponibiliza potentes interfaces para manutenção de ajudas, criação de mapas de perguntas/respostas (para *eMarketing*), etc.

- **Administration Center**

O Contact Center oferece aos administradores do sistema meios para uma fácil manutenção de Bases de Dados e Aplicações. No *Administration Center* são disponibilizadas várias funcionalidades como: gestão os conteúdos do portal de entrada (*Info Center*), criação de segmentos e campanhas de *eMarketing*, manutenção de Bases de Dados e utilizadores, e diversas informações estatísticas de gestão.

- **eMarketing**

Esta aplicação é um instrumento valioso para os departamentos de marketing e comercias. Inclui um componente de CRM (*Customer Relation Management*), que é responsável por analisar o perfil dos clientes, de modo a seleccionar clientes alvo das campanhas definidas no *Administration Center*.

A aplicação de *eMarketing* é normalmente utilizada pelos Agentes, podendo estes contactar clientes (ou potenciais clientes) para promover produtos e serviços (campanhas *OutBound*) ou aproveitar um contacto efectuado pelo cliente para publicitar campanhas das quais o cliente seja alvo (campanhas *InBound*). A aplicação guia o Agente na promoção da campanha, indicando os dados disponíveis do cliente, que perguntas deverá efectuar, que

argumentos deverá utilizar, que produtos existem na concorrência, etc. Todas estas indicações são parametrizáveis no *Supervisor Desktop*. Está também disponível na aplicação de *eMarketing* um sistema de ajuda *on-line* com o objectivo de familiarizar o Agente com os aspectos de cada campanha.

➢ **Help Desk**

O *Help Desk* é uma aplicação standard para o atendimento de clientes. Os utilizadores são tipicamente Agentes, estando os componentes de telefonia integrados nesta aplicação. Contudo, ao contrário do *Agent Desktop*, o *Help Desk* é mais orientado a problemas de resolução imediata: fornecimento de informações gerais, esclarecimento de duvidas relativas a produtos e serviços, etc. Para além da telefonia, serviços de *email*, fax e *web-chat* são também suportados pela aplicação.

2.3 Arquitectura

Um Contact Center de grande porte implica a existência de uma infra-estrutura que responda aos requisitos de processamento, tráfico e segurança da instituição. A *Figura 2.2* representa a arquitectura habitual de um Contact Center.

Quando um cliente contacta a instituição através da telefonia, a chamada é tratada através de uma central telefónica (PABX). O cliente poderá optar por um atendimento automático, acedendo às principais operações através do teclado do telefone. Este serviço é disponibilizado através do IVR (*Interactive Voice Response System*). Se o cliente preferir um atendimento personalizado, o PABX reencaminha a chamada para o CTI (*Computer Telephony Integration*) do modo a estabelecer a ligação com um Agente. O CTI é um sistema de central telefónica que permite o controlo das chamadas através de *software*. Isto permite ao Agente controlar a telefonia (atender, marcar, transferir chamadas, etc) através de um interface fornecido pela aplicação ligada ao *CTI Server*.

Muitas organizações fornecem também vários serviços para os clientes através da *Internet*: *email*, *web-chat*, *Home Banking*, etc. Estes serviços são disponibilizados através de servidores *web* colocados numa zona desmilitarizada (DMZ). A DMZ é protegida contra possíveis ataques do exterior através de uma *firewall*.

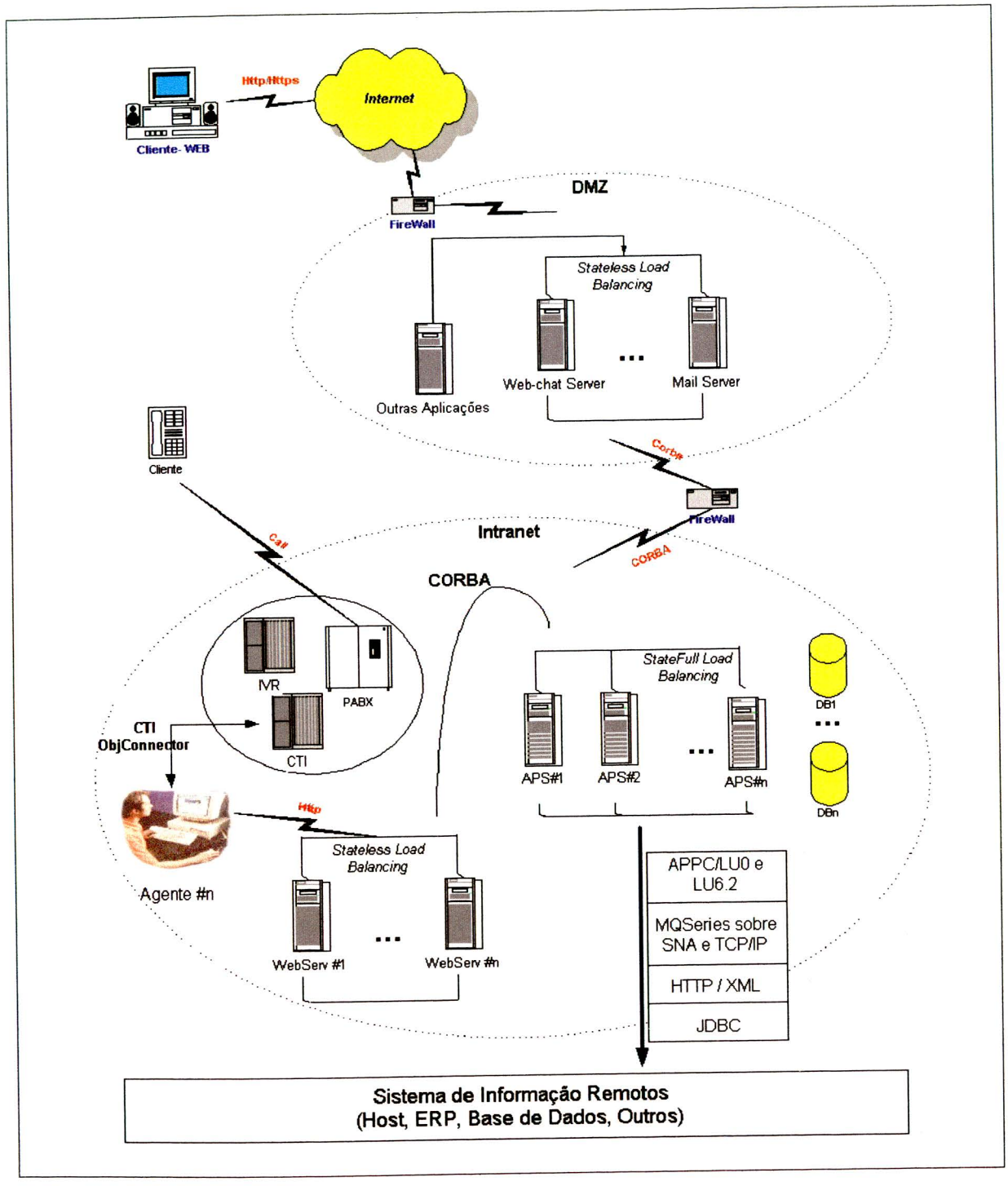


Figura 2.2 – Arquitectura Contact Center

Na rede interna da instituição, o interface para os utilizadores é também gerado a partir de servidores *web*, de forma que as aplicações possam correr em *web browsers* comuns, independentes da plataforma cliente. A lógica de negócio é implementada através de uma camada transaccional assente em servidores aplicacionais. A comunicação entre esta camada e a camada de apresentação e efectuada através de *Corba*. Para garantir a segurança nos acessos do exterior para a rede interna, a ligação para a DMZ contém ainda uma segunda *firewall*.

Habitualmente, as instituições possuem grandes quantidades de informação nas suas Bases de Dados e sistemas. Não seria prático a integração directa das Bases de Dados do Contact Center nos sistemas das instituições, pois para além dos custos acrescidos desta integração, o volume de tráfego produzido pelo Contact Center poderia diminuir a performance dos sistemas remotos da instituição. Deste modo toda a informação relativa ao funcionamento do Contact Center fica guardado em Bases de Dados locais com acesso directo a partir da camada transaccional. Os dados da organização que sejam usados com mais frequência no Contact Center (tipicamente informações gerais dos clientes) são replicados nas Bases de Dados locais, de modo a aumentar a performance do Contact Center e diminuir a carga nos outros sistemas da instituição. Os restantes dados normalmente só são necessários a pedido do cliente, não se justificando a replicação dos dados nas Bases de Dados do Contact Center. A informação necessária das Bases de Dados da instituição pode ser obtida através de vários protocolos de comunicação e sincronização, dependo dos sistemas e plataformas existentes na instituição.

Todos os servidores do Contact Center permitem o balanceamento de carga, podendo existir vários servidores *web*, aplicacionais e de Bases de Dados, dependendo dos requisitos de tráfego da instituição.

2.4 A Plataforma WIS

Para dar resposta a aplicações de grande dimensão com um elevado número de requisitos tecnológicos e de segurança, a CPC TA desenvolveu um *middleware* para a implementação deste tipo de soluções. Denominada *Web Information System* (WIS), esta poderosa plataforma oferece suporte para a maior parte das aplicações da CPC TA para *Internet* e *Intranet*, entre as quais o Contact Center.

A plataforma WIS dispõe de elevados níveis de performance e segurança, capacidade de balanceamento de carga, fácil gestão e administração. Oferece também grandes facilidades para a integração entre aplicações WIS e conexão a sistemas de informação remotos. O tempo de desenvolvimento de aplicações sobre a plataforma é diminuído devido a vários métodos, APIs, componentes e módulos reutilizáveis incluídos na WIS.

Este *middleware* é explicado em maior pormenor no Anexo B – "*Plataforma WIS*".

3 Apresentação do Problema

Neste capítulo é descrito em pormenor o problema em questão. São introduzidas as circunstancias que levaram à implementação de um novo Contact Center – Linha Branca, seguindo-se a especificação formal dos requisitos funcionais e não funcionais.

3.1 Contextualização

Dado os constantes avanços tecnológicos que as TIs sofrem, facilmente um sistema recente se torna desactualizado. Num mercado cada vez mais competitivo, as empresas têm que acompanhar as novas tecnologias para manterem a sua competitividade. Estas evoluções frequentes envolvem custos muitas vezes elevados, uma vez que a actualização de um sistema pode implicar uma reestruturação significativa de outros sistemas dependentes.

Quando uma instituição de grande dimensão adquire um sistema informático, tende a optar pela solução tecnologicamente mais avançada, mesmo que os custos associados sejam mais elevados. É portanto crucial que as empresas que fornecem este tipo de soluções estejam preparadas para os exigentes requisitos das instituições. Neste momento a procura soluções de grande porte divide-se em dois segmentos principais:

- Instituições com preferência por soluções *Microsoft* devido às facilidades na interacção com outros sistemas e aplicações *Microsoft*, cada vez mais usados ao nível empresarial. A mais recente estratégia da *Microsoft* para o desenvolvimento de soluções empresariais multi-camada é a plataforma .NET, que apesar da sua recente existência, traz perspectivas muito promissoras.
- Outras instituições optam por soluções não proprietárias, independentes do sistema operativo, com exigências de custos e *hardware* geralmente mais baixas, seguindo standards conceituados, utilizando sistemas baseados em *Java* da *Sun Microsystems*.

Com o objectivo de dar resposta às actuais necessidades das grandes e médias empresas, a CPC TA resolveu evoluir a plataforma WIS para as novas tecnologias emergentes. Deste modo vão ser desenvolvidas duas versões para nova plataforma: WIS2.NET e WIS2.JAVA.

Com a reestruturação da WIS, as aplicações implementadas na plataforma terão também de ser adaptadas para a WIS2. Este é o caso do Contact Center.

Como já foi referido, existe um Contact Center Linha Branca usado como base de desenvolvimento dos Contact Centers para as instituições. Aproveitando a evolução da plataforma WIS a CPC TA decidiu fazer uma nova Linha Branca, com funcionalidades mais enquadradas nas necessidades dos clientes, melhor usabilidade, melhor estruturação, e um *design* remodelado. Contudo, a conclusão da WIS2 (versão *Java*) só está prevista para Setembro/Outubro do presente ano. Se o desenvolvimento da Linha Branca fosse iniciado apenas nessa altura, a sua conclusão arrastar-se-ia até 2003, e outros projectos de Contact Center que entretanto surgissem não poderiam beneficiar da nova Linha Branca como base. Deste modo optou-se por desenvolver a Linha Branca numa versão melhorada da plataforma WIS original, utilizando já certas normas e metodologias da futura WIS2. As alterações mais significativas na construção do Contact Center Linha Branca dar-se-ão nas camadas de

Apresentação, mantendo-se para já as mesmas regras gerais da camada Transaccional. Prevê-se assim que a nova Linha Branca esteja pronta a quando a conclusão da WIS2, bastando então migrar o motor transaccional para as novas definições da plataforma.

3.2 Especificação de Requisitos

Nesta secção são especificados os requisitos funcionais e não funcionais para a fase inicial do Contact Center Linha Branca a implementar. Os requisitos funcionais visam captar a forma como o sistema é visto externamente, ou seja, correspondem ao conjunto de funcionalidades que a aplicação deve disponibilizar aos seus diferentes tipos de utilizadores. Os requisitos não funcionais são elementos que influenciam o modo como as referidas funcionalidades vão ser implementadas. Estes requisitos podem ir desde restrições impostas pelo ambiente em que o sistema vai operar até, por exemplo, necessidades especiais ao nível do interface gráfico com o utilizador.

O desenvolvimento da Linha Branca segue um modelo evolutivo, consistente em várias iterações de fases de especificação, desenvolvimento e validação. Deste modo os requisitos aqui apresentados foram sujeitos a várias revisões no decorrer do projecto, tanto a nível funcional como não funcional.

3.2.1 Requisitos Funcionais

Com as experiências anteriores no desenvolvimento de Contact Centers para as instituições, observou-se quais as funcionalidades com maior aceitação por parte dos clientes. Foram ainda projectadas novas funcionalidades que trarão mais valias a futuros Contact Centers.

De modo a ilustrar formalmente os requisitos funcionais, são apresentados seguidamente um conjunto de diagramas de casos de uso modelados através da *Unified Modelling Language* (UML)[1]. Dada a dimensão do sistema, os diagramas apresentados não entrarão no nível máximo de detalhe. Apenas é pretendido especificar todas as funcionalidades de um modo geral, detalhando apenas as que se considerem mais significantes.

Sempre que se mostre relevante, serão também indicadas as alterações funcionais relativamente ao antigo Contact Center – Linha Branca.

3.2.1.1 Visão Geral

Na Figura 3.1 está representado o diagrama de caso de uso geral do Contact Center – Linha Branca. São visíveis todos os utilizadores do sistema e os pacotes aplicacionais disponíveis para cada tipo de utilizador.

[1] Esta linguagem é especificada no documento *Unified Modelling Language Specification* [1].

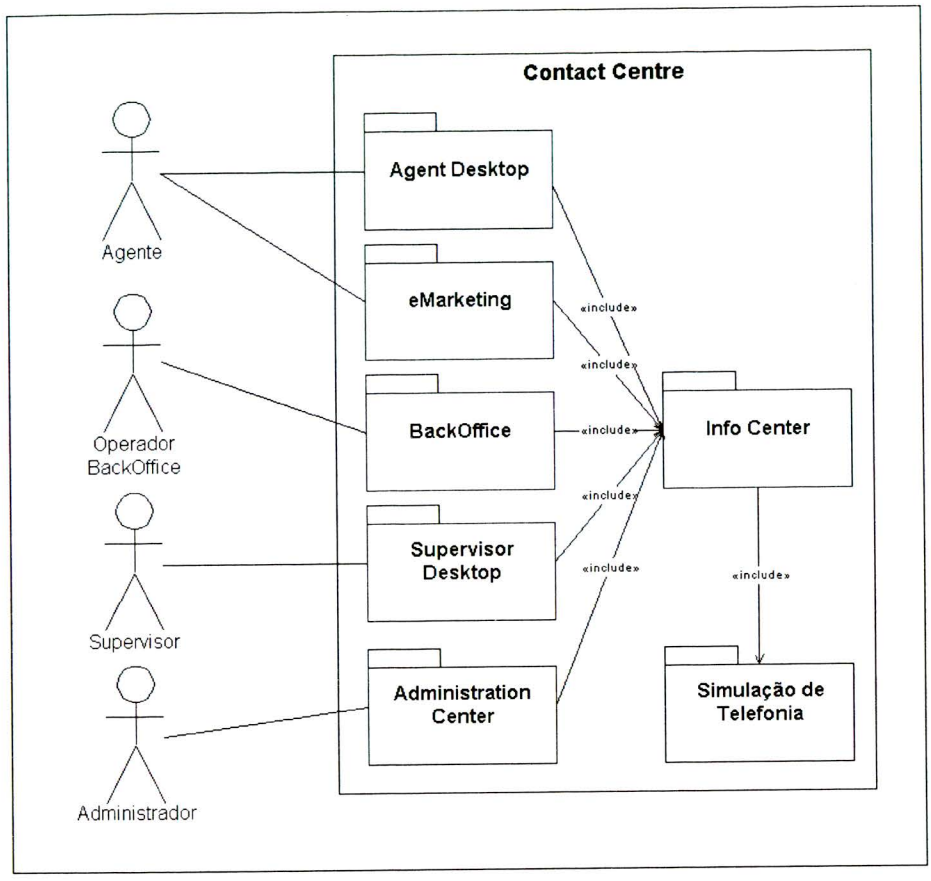


Figura 3.1 – Caso de Uso: Contact Center

Actores

- *Agente* – operador telefónico que utiliza as aplicações destinadas ao contacto directo com clientes.
- *Operador de BackOffice* – tem como função principal a análise e execução de pedidos e operações.
- *Supervisor* – define parâmetros relativos ao funcionamento das operações do negócio e supervisiona operações de *Agentes* e *Operadores de BackOffice*.
- *Administrador* – responsável pelas actividades relacionadas com a gestão dos aspectos gerais da aplicação.

Nota: *A plataforma WIS possui um elevado nível de construção de perfis de utilizadores. Deste modo um Actor pode desempenhar vários papeis em simultâneo, tendo assim acesso a vários pacotes aplicacionais. É também possível restringir o acesso base a funcionalidades especificas a utilizadores individuais, ou seja, o facto de um utilizador ter acesso geral a uma aplicação, não implica que tenha disponível todas as funcionalidades da aplicação.*

Pacotes

No diagrama anterior, cada pacote representa um módulo aplicacional (excepto a *Simulação de Telefonia*). O *Info Center* é o portal de entrada para todas as outras aplicações. Deste modo o uso de qualquer aplicação implica a passagem pelo *Info Center*.

- *Info Center* – é o portal de entrada para todos os utilizadores do Contact Center. Este portal de informação dispõe vários conteúdos como notícias, apontadores (*links*), publicidade, etc.
- *Agent Desktop* – esta aplicação disponibiliza os serviços necessários para a comunicação com o cliente. Inclui, entre outras funcionalidades, a gestão de pedidos, agenda, contactos, pesquisa e dados de clientes.
- *eMarketing* – aplicação que fornece informações sobre campanhas e clientes alvo, com o objectivo de promoção produtos e serviços.
- *BackOffice* – tem como objectivo o tratamento de pedidos que não possam ser executados de forma automática sem a supervisão de um profissional qualificado.
- *Supervisor Desktop* – permite a monitorização, intervenção na gestão de clientes, e manutenção de informação de apoio ao negócio.
- *Administration Center* – disponibiliza meios para a manutenção das restantes aplicações: gestão de conteúdos, utilizadores, Bases de Dados, etc.
- *Simulação de Telefonia* – O sistema para controlo de telefonia, integrado nos Contact Centers da CPC TA, é normalmente subcontratado a uma terceira entidade. Na Linha Branca apenas se requer um sistema capaz de simular o comportamento básico da *applet* de controlo de telefonia.

Comparação com o sistema anterior

A este nível, a alteração mais significativa em comparação com a anterior Linha Branca é a eliminação da aplicação de *Help Desk*. Considerou-se preferível desenvolver esta aplicação num módulo separado, sem integração directa na base do Contact Center. Deste modo o *Help Desk* pode ser vendido como um componente separado, podendo eventualmente ser integrado num Contact Center específico, se a instituição o requisitar.

3.2.1.2 Info Center

O *Info Center* funciona como um portal de entrada para todos os utilizadores do Contact Center. Esta aplicação disponibiliza diversos conteúdos que os utilizadores podem visualizar antes de entrarem na aplicação apropriada ao seu cargo na instituição. A *Figura 3.2* contém o caso de uso representativo das principais funcionalidades requisitadas para o *Info Center*.

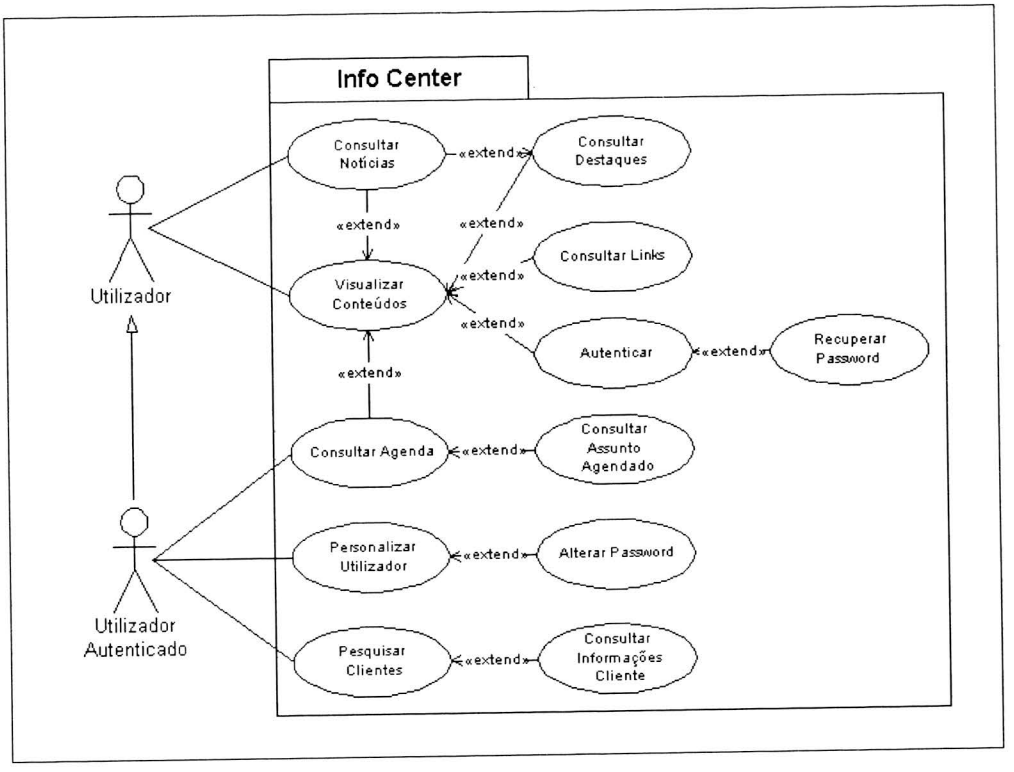


Figura 3.2 – Caso de Uso: *Info Center*

Actores

- *Utilizador* – corresponde a todos os tipos de utilizador do Contact Center, uma vez que o *Info Center* é destinado a todos os utilizadores, independentemente da sua função no negócio.
- *Utilizador Autenticado* – equivale a um *Utilizador* que passou o processo de autenticação. Herda todas as funcionalidades acessíveis ao *Utilizador*, para além de ter também disponíveis outras funcionalidades de acesso restrito.

Casos de Uso

- *Visualizar Conteúdos* – representa a entrada do *Info Center*, com a disponibilização de conteúdos gerais é como objectivo principal. Os conteúdos disponíveis na entrada são notícias, destaques (de notícias), apontadores (*links*), *banners* publicitários, informações gerais e a agenda do utilizador. Esta ultima só está disponível para o *Utilizador Autenticado*, sendo a *consulta da agenda* substituída por uma opção de *autenticação* para utilizadores não autenticados. Para o *Utilizador Autenticado*, a visibilidade de cada conteúdo na entrada do portal pode ser *personalizada*.
- *Consultar Notícias* – disponível para todos os utilizadores, este caso de uso inclui a listagem das notícias mais recentes e a possibilidade de visualizar os detalhes de cada notícia.
- *Consultar Destaques* – listagem das notícias que sejam alvo de destaque.
- *Consultar Links* – lista de apontadores (*links*) agrupados por tema.

- *Autenticar* – permite a autenticação do *Utilizador* no sistema, obtendo acesso às opções do *Utilizador Autenticado*.
- *Recuperar Password* – implica uma autenticação através de um processo de pergunta/resposta (previamente personalizado), e no caso de sucesso, será gerada automaticamente uma nova *password* para o *Utilizador*.
- *Consultar Agenda* – corresponde à consulta da agenda de um *Utilizador Autenticado*, havendo a possibilidade de ver os detalhes de cada assunto agendado.
- *Consultar Assunto Agendado* – representa a visualização dos detalhes de um assunto em agenda.
- *Personalizar Utilizador* – um *Utilizador Autenticado* pode editar os seus dados pessoais e quais os conteúdos que pretende ver à sua entrada no sistema.
- *Alterar Password* – extensão da *personalização do utilizador* que permite a alteração da *password* do *Utilizador Autenticado*.
- *Pesquisar Clientes* – permite a pesquisa de clientes através de vários parâmetros.
- *Consultar Informações Cliente* – representa a visualização de dados gerais de um cliente, seleccionado após uma acção de pesquisa.

Comparação com o sistema anterior

A inovação principal no *Info Center* da nova Linha Branca é a existência de funcionalidades personalizáveis para *Utilizadores Autenticados*. Anteriormente esta aplicação era indiferente ao utilizador, sendo a autenticação efectuada à entrada de cada aplicação. Deste modo os conteúdos do portal não eram personalizáveis.

Em suma, as novas funcionalidades são nomeadamente, autenticação, recuperação de *password*, consulta de agendas, personalização e pesquisa de clientes; incluindo todas as funcionalidades consequentes.

3.2.1.3 Agent Desktop

Este pacote representa a aplicação base para os *Agentes*. O objectivo principal é disponibilizar informação de suporte para os contactos com clientes.

O *Agent Desktop* contém modos de operação ligeiramente distintos, dependendo na identificação e autenticação de um cliente. A *Figura 3.3* representa o estado geral da aplicação antes de ocorrer a identificação de um cliente. No diagrama pode-se ainda observar dois sub-pacotes do *Agent Desktop*, englobando os casos de uso comuns (independentemente da identificação do cliente), e as funcionalidades especificas para um cliente seleccionado. Os diagramas de casos de usos representativos destes pacotes encontram-se nas *Figuras 3.4* e *3.5* respectivamente.

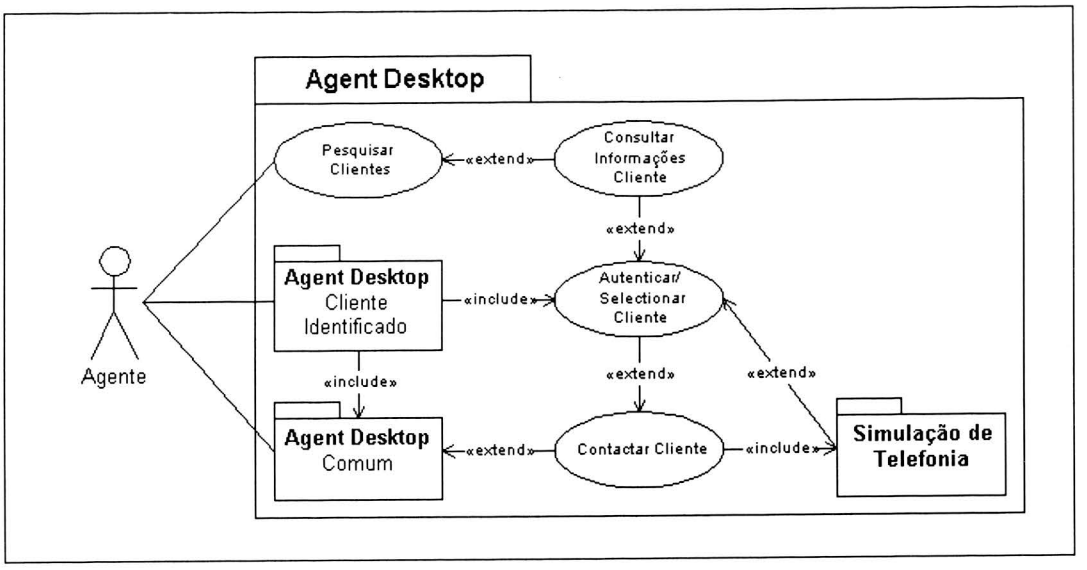


Figura 3.3 – Caso de Uso: *Agent Desktop* (visão global)

Casos de Uso

- *Pesquisar Clientes* – permite a pesquisa de clientes através de vários parâmetros, semelhante à pesquisa no *Info Desk*.
- *Consultar Informações Cliente* – representa a visualização de dados gerais de um cliente, seleccionado após a acção de pesquisa. Adicionalmente possibilita a identificação do cliente na aplicação.
- *Autenticar/Seleccionar Cliente* – este caso de uso representa a identificação de um cliente perante a aplicação, sendo que ao sucesso desta identificação, o modo de funcionamento da aplicação passa a ser definido pelo pacote *Agent Desktop – Cliente Identificado*, incluindo também as funcionalidades comuns da aplicação. Dependendo das políticas de negócio das instituições, o acesso à área de clientes pode ou não estar sujeito a um processo de autenticação. Deste modo a aplicação deverá ser configurável para a inclusão de um processo de autenticação de clientes, ou para a selecção directa do cliente. No sistema de autenticação, cada cliente possuirá uma *password* com 8 caracteres, sendo necessária a introdução de 3 caracteres (aleatórios) da *password* para proceder.
- *Contactar Cliente* – caso um *Agente* tenha um contacto com um cliente na sua agenda, deverá efectuar esse contacto na altura devida, utilizando o sistema (simulado) de controlo de telefonia. Após o contacto com o cliente, o *Agente* poderá autenticar/seleccionar o cliente para tratar os pedidos eventualmente efectuados.

Pacotes

- *Agent Desktop (Comum)* – representado na *Figura 3.4*, este pacote define todas as funcionalidades presentes no *Agent Desktop* em todas as ocasiões, independentemente de um cliente ter sido ou não seleccionado.

- *Agent Desktop (Cliente Identificado)* – contém os casos de uso para a consulta e tratamento de dados específicos de um cliente (representado na *Figura 3.5*).
- *Simulação de Telefonia* – sistema para a simulação do controlo de telefonia, descrito posteriormente.

3.2.1.3.1 ***Agent Desktop*** – *Comum*

Neste sub-pacote estão representadas as funcionalidades gerais presentes em todas as ocasiões na aplicação principal dos *Agentes*. Contudo, alguns dos casos de uso especificados possuem comportamentos que dependem do estado da identificação de um cliente, apesar do objectivo principal ser constante.

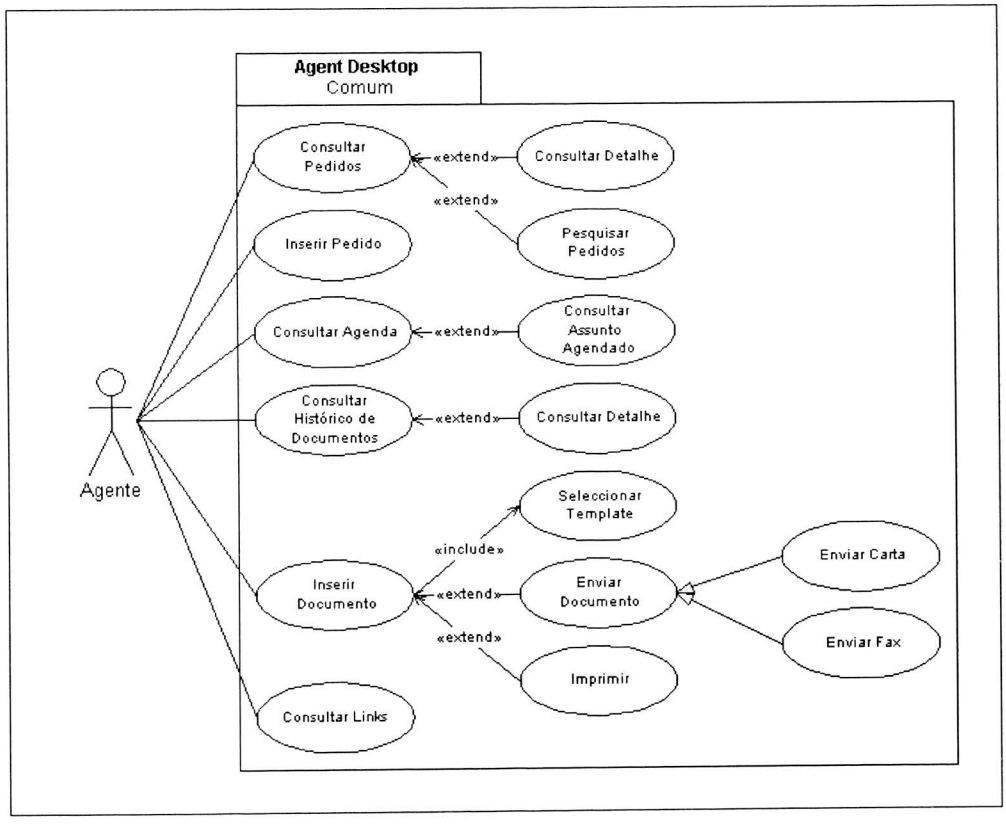


Figura 3.4 – Caso de Uso: *Agent Desktop* – Comum

Casos de Uso

- *Consultar Pedidos* – corresponde à consulta dos pedidos registados no sistema. Caso o agente esteja em contacto com um cliente previamente identificado, devem aparecer apenas os pedidos relativos ao cliente identificado. O *Agente* pode também visualizar informação detalhada de cada pedido e pesquisar pedidos através de vários parâmetros.
- *Inserir Pedido* – um cliente poderá solicitar vários tipos de pedidos, de modo a serem efectuadas as operações pretendidas. O *Agente* limita-se a registar os pedidos

efectuados, sendo estes posteriormente executados (ou rejeitados) em *BackOffice*. Os pedidos são relativos a um cliente, contudo quando o cliente não está identificado, o pedido é inserido como pertencente a um cliente desconhecido. Os pedidos disponíveis poderão ser de vários tipos, cada um com características particulares muitas vezes complexas. No âmbito da Linha Branca apenas devem ser implementados alguns pedidos genéricos (reclamações, pedidos de adesão, etc) a título de exemplo.

- *Consultar Agenda* – corresponde à consulta da agenda de contactos do *Agente*. Para além de atender chamadas de clientes, o *Agente* pode também ter que contactar um cliente em horários específicos, indicados na agenda. Caso exista um cliente identificado no momento, apenas são visíveis os assuntos agendados para o cliente actual.
- *Consultar Assunto Agendado* – representa a visualização dos detalhes de um assunto na agenda.
- *Consultar Histórico de Documentos* – visualização da lista de documentos previamente registados. Seleccionado um documento pode ser visto o respectivo detalhe.
- *Consultar Detalhe (Documento)* – nos detalhes de um documento pode-se alterar o seu estado (recebido, devolvido, etc), podendo também optar-se por reenviar o documento.
- *Inserir Documento* – representa a inserção (e possível envio) de um documento. Um *Agente* pode produzir um documento (seleccionando previamente um *template*), podendo seguidamente imprimir ou registar o seu envio através de carta ou fax.
- *Consultar Links* – Corresponde aos apontadores (*links*) atribuídos para o *Agent Desktop*.

3.2.1.3.2 **Agent Desktop** – *Cliente Identificado*

Ao identificar um cliente o *Agente* tem acesso a novas funcionalidades relativas ao negócio da instituição. Sendo a Linha Branca um Contact Center genérico, a área de negócio do sistema final é uma incógnita. Contudo, a maior parte dos clientes da CPC TA pertencem ao sector financeiro, e a título de exemplo, a Linha Branca deverá implementar algumas funcionalidades típicas em Contact Centers de instituições financeiras.

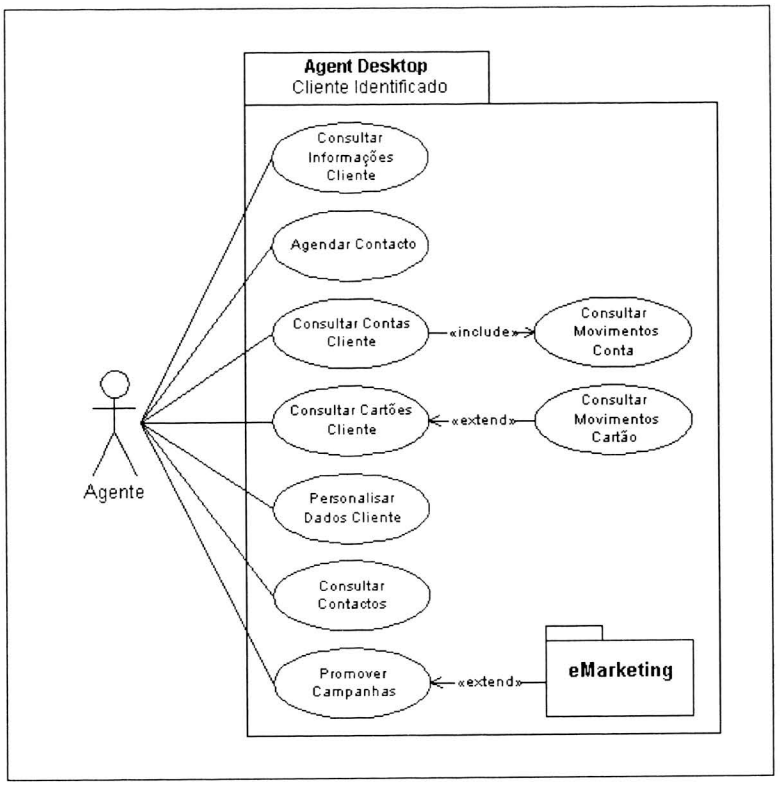


Figura 3.5 – Caso de Uso: *Agent Desktop* – Cliente Identificado

Casos de Uso

- *Consultar Informações Cliente* – o *Agente* pode em qualquer momento ver as informações gerais do cliente com o qual estabeleceu contacto.
- *Agendar Contacto* – na eventual ocorrência de problemas diversos durante o contacto (por ex: falha no sistema, indisponibilidade do cliente, etc), o *Agente* pode agendar o contacto de modo a tratar o assunto pendente numa outra altura mais conveniente.
- *Consultar Contas Cliente* – representa a consulta de uma lista com as contas do cliente, podendo também ser visualizado os detalhes e movimentos de uma conta.
- *Consultar Cartões Cliente* – de modo semelhante à consulta de contas, o *Agente* pode ver a listagem, detalhes e movimentos de contas cartão do cliente identificado.
- *Personalizar Dados Cliente* – consiste na alteração de dados diversos relativos ao cliente, como por exemplo, a alteração da conta *default*.
- *Consultar Contactos* – representa a possibilidade de consultar os contactos efectuados previamente com o cliente actual, incluindo os detalhes e o historial de cada contacto.
- *Promover Campanhas* – durante o contacto, o *Agente* pode tentar aliciar o cliente em campanhas das quais seja alvo. Caso o cliente mostre interesse numa campanha, o *Agente* poderá redireccionar-se para a aplicação de *eMarketing* de modo a obter apoio para a promoção e eventual venda do produto ou serviço.

Comparação com o sistema anterior

Relativamente à antiga versão da Linha Branca, existem várias funcionalidades que foram revistas. A mudança mais significativa é a inclusão de métodos para a consulta e inserção de documentos, permitindo ao *Agente* uma interacção não exclusivamente dependente à telefonia. Outras funcionalidades acrescentadas são a possibilidade de personalização de dados do cliente, consulta de movimentos e contas de cartões.

Quanto a opções não incluídas nesta especificação inicial, destaca-se a possibilidade de visualização de informações detalhadas do *Agente* (considerado não muito significativo para o *Agent Desktop*), e a consulta e introdução de informações sobre assuntos diversos (que teve pouca aderência em Contact Centers anteriores, devido ao afastamento da área de negócio).

Na anterior Linha Branca era ainda permitido ao *Agente* a alteração da *password* (PIN) do cliente. Com o novo sistema de *passwords* de clientes evita-se que o *Agente* conheça a *password* completa do cliente. Deste modo, na nova versão do sistema a alteração da *password* não é permitida, pois iria contra esta filosofia.

3.2.1.4 BackOffice

A aplicação *BackOffice* tem como objectivo o fornecimento de meios para que operadores especializados na área do negócio, possam analisar e tratar os pedidos efectuados por clientes, através dos *Agentes*.

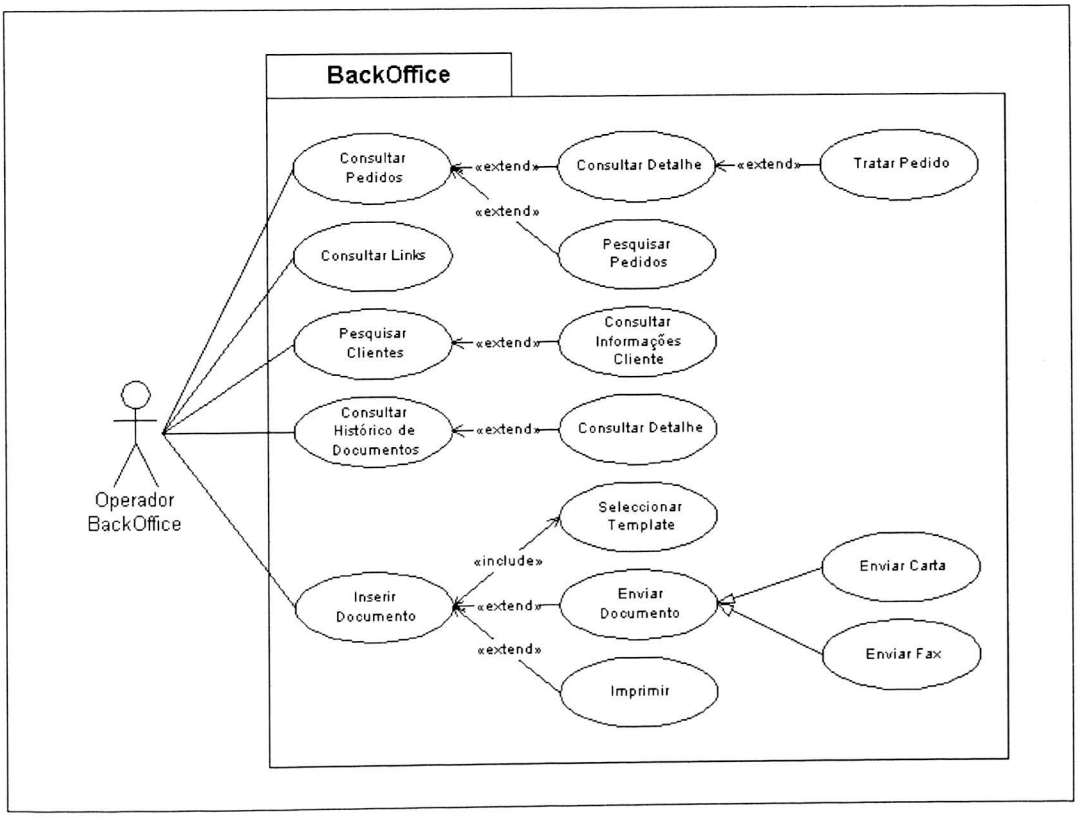


Figura 3.6 – Caso de Uso: *BackOffice*

Casos de Uso

À excepção da possibilidade de *Tratar Pedidos*, os casos de uso representados na *Figura 3.6* já foram descritos em casos de uso equivalentes na aplicação *Agent Desktop*. É de salientar apenas que os documentos a enviar no módulo de documentação são definidos por *templates*, podendo o tipo de documentos enviados no *BackOffice* ser significativamente diferentes dos do *Agent Desktop*.

- *Tratar Pedidos* – o *Operador de BackOffice* ao consultar os detalhes de um pedido pode efectuar a sua execução, seja esta externa ou posteriormente integrada no Contact Center. No âmbito da Linha Branca, o *Operador de BackOffice* poderá avaliar cada pedido como aceite, rejeitado, pendente (agendado em *BackOffice* para posterior tratamento), ou *CallBack* (colocado na agenda do *Agente* que registou o pedido para recontactar o cliente). Em cada um dos casos o utilizador poderá introduzir observações que ache necessárias.

Comparação com o sistema anterior

A principal mudança na evolução do *BackOffice* é o sistema de envio e consulta de documentos. Apesar do módulo de documentação ter sido considerado para o *BackOffice* da Linha Branca original, nunca chegou a ser implementado.

3.2.1.5 Supervisor Desktop

Este pacote aplicacional tem como objectivo fornecer aos supervisores meios para definirem regras do negócio e monitorizarem eventos relacionados com o comportamento dos intervenientes no Contact Center. A *Figura 3.7* contém os casos de uso desta aplicação.

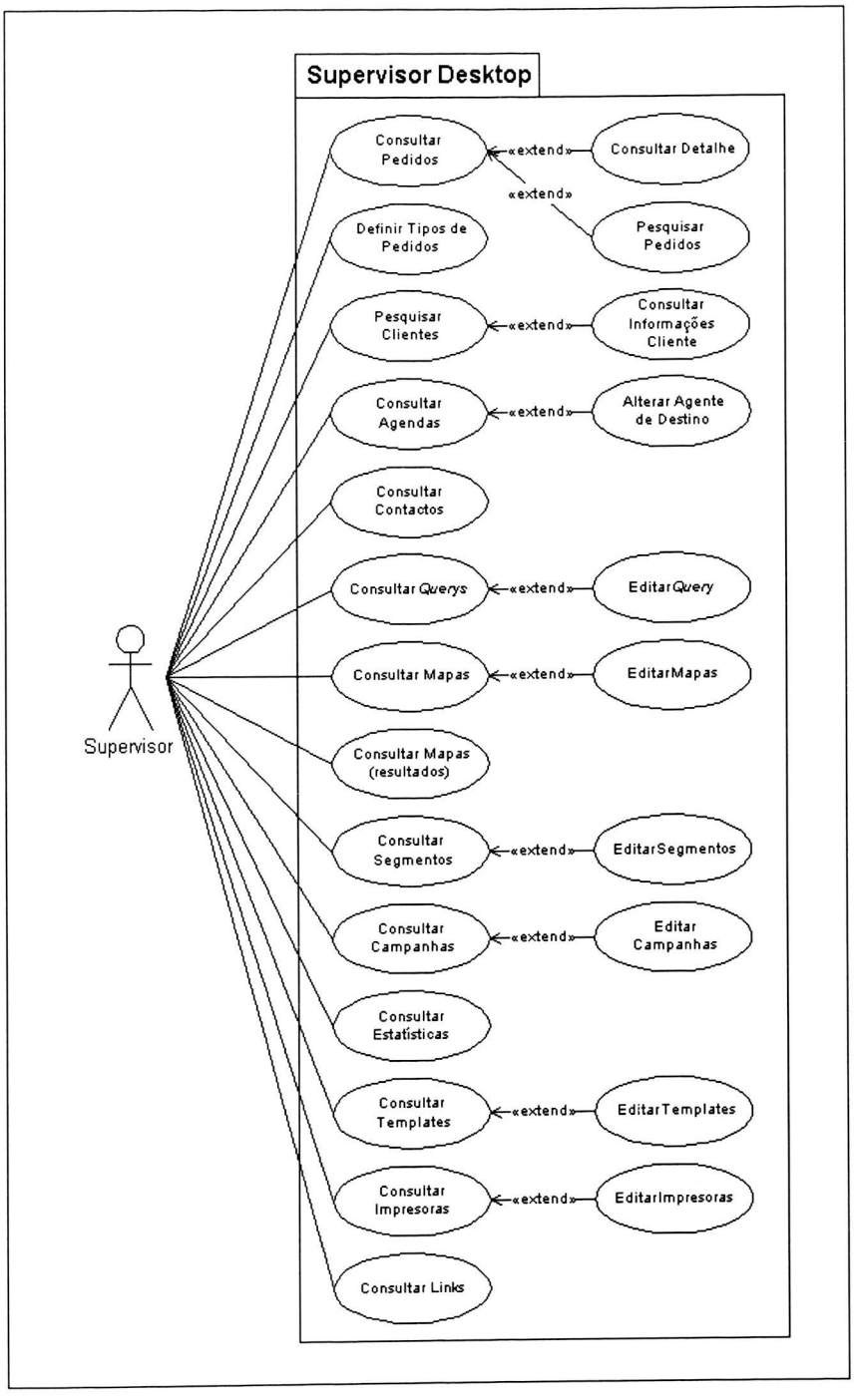


Figura 3.7 – Caso de Uso: *Supervisor Desktop*

Casos de Uso

Os casos de uso representativos das acções de *Consultar Pedidos*, *Pesquisar Clientes*, *Consultar Contactos*, *Consultar Links*, e todas as opções consequentes, são equivalentes a casos de uso já descritos anteriormente.

- *Definir Tipos de Pedidos* – corresponde à definição do tipo e estrutura dos pedidos disponíveis no *Agent Desktop*. A definição dos campos de entrada de um pedido pode

ser feita directamente através da introdução de código XML, ou por intermédio de interfaces mais intuitivos.

- *Consultar/Alterar Agendas* – o *Supervisor* pode consultar todos os assuntos agendados pelos *Agentes*, assim como *Alterar o Agente de Destino* de um agendamento, de modo que o item seja movido para a agenda de outro *Agente*.
- *Consultar/Editar Querys* – representa a consulta, introdução e alteração de *querys* (pesquisas) à Base de Dados. Os *querys* podem ser introduzidos através de código SQL, ou com a ajuda de interfaces fornecidos pela aplicação.
- *Consultar/Editar Mapas* – um mapa é a apresentação formatada de um *query*. Através de *querys* e mapas o *Supervisor* pode apresentar quaisquer dados que ache relevantes, sem ser necessário nenhuma intervenção técnica no Contact Center.
- *Consultar Mapas (resultados)* – permite visualizar o resultado do mapa criado nos pontos anteriores, na forma de uma tabela.
- *Consultar/Editar Segmentos* – semelhante à consulta e edição de *querys* mas com o objectivo específico de criar segmentos de clientes alvos para campanhas. Um exemplo de um segmento poderia ser "todos os clientes do sexo masculino com menos de 25 anos".
- *Consultar/Editar Campanhas* – o *Supervisor* pode consultar, criar e alterar campanhas para a promoção de produtos ou serviços. Para cada campanha podem ser definidos vários parâmetros, entre os quais: segmentos de clientes alvo da campanha; grafos de sequências perguntas/respostas para guiar os *Agentes* na promoção da campanha; argumentos a utilizar para convencer o cliente; etc.
- *Consultar Estatísticas* – corresponde à visualização de diversos dados estatísticos, relativos a pedidos, chamadas, campanhas, etc.
- *Consultar/Editar Templates* – os *templates* disponíveis para os documentos criados no *Agent Desktop* e *BackOffice* são aqui criados pelo *Supervisor*.
- *Consultar/Editar Impressoras* – corresponde à consulta e definição de impressoras a utilizar para a impressão de documentos.

Comparação com o sistema anterior

A antiga versão do *Supervisor Desktop* continha algumas funcionalidades para a gestão de *Help Desk*. Estas opções foram retiradas visto que esta aplicação não é incluída na nova Linha Branca.

Quanto a funcionalidades adicionadas, destaca-se a criação de *querys* e mapas, segmentos e campanhas (anteriormente implementados na aplicação de Administração), definição de *templates* e impressoras.

3.2.1.6 Administration Center

Esta aplicação de administração disponibiliza ferramentas para a gestão geral do Contact Center, incluindo a manutenção de utilizadores, tabelas e conteúdos. A *Figura 3.8* representa os casos de uso deste pacote.

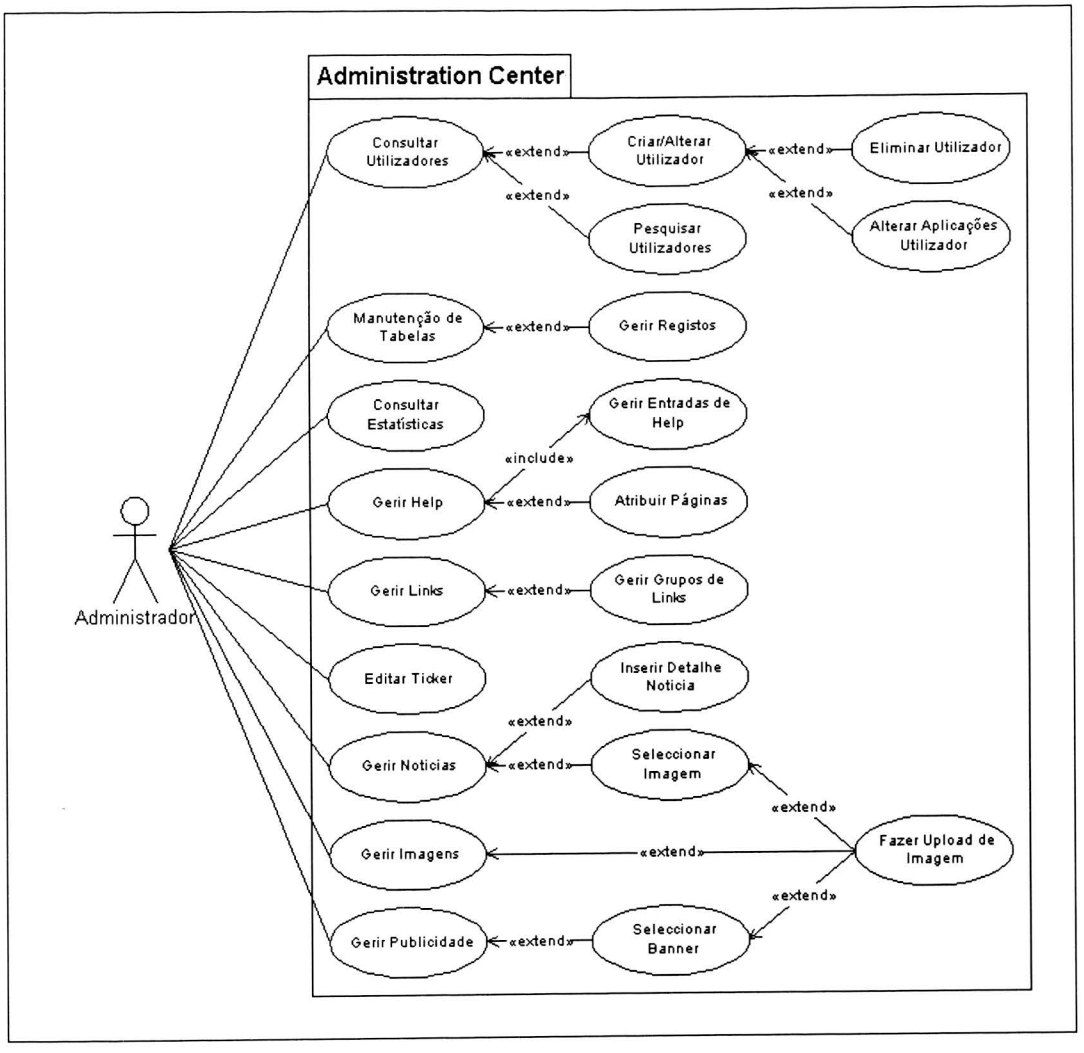


Figura 3.8 – Caso de Uso: *Administration Center*

Casos de Uso

- *Consultar/Pesquisar Utilizadores* – representa a visualização de uma lista de utilizadores, podendo os resultados ser refinados através de uma pesquisa.
- *Criar/Alterar Utilizador* – cabe ao *Administrador* do Contact Center a manutenção dos utilizadores, sendo-lhe permitido a criação, modificação e eliminação dos utilizadores do sistema.
- *Alterar Aplicações Utilizador* – permite a definição das aplicações às quais o utilizador tem acesso, definindo assim a função do utilizador no Contact Center (*Agente*, *Supervisor*, etc).
- *Manutenção de Tabelas* – corresponde à consulta das Bases de Dados, tabelas e registos disponíveis, sendo possível introduzir e alterar os registos de uma tabela através da aplicação.
- *Consultar Estatísticas* – assim como no *Supervisor Desktop,* corresponde à visualização de diversos dados estatísticos.

- *Gerir Help* – representa a gestão de itens do sistema de ajuda *on-line* (*Help*), incluído em todas as aplicações do Contact Center. Os itens podem ser sobre temas genéricos ou relativos a uma funcionalidade específica duma aplicação.
- *Atribuir Páginas* – pode-se definir que uma determinada página de uma aplicação corresponde a uma entrada de ajuda. Isto permite que quando um utilizador acciona o sistema de ajuda a partir da página em questão, seja mostrado o item de ajuda respectivo (ajuda contextual).
- *Gerir Links* – os apontadores (*links*) disponíveis nas restantes aplicações são aqui definidos. Podem também ser geridos grupos de *links*, e cada *link* pode pertencer a uma ou mais aplicações.
- *Editar Ticker* – cada aplicação dispõe ainda de um dispositivo para a passagem de mensagens em rodapé, intitulado de *ticker*. A mensagem que aparece no *ticker* de cada aplicação é aqui definida pelo *Administrador*.
- *Gerir Notícias* – corresponde à inserção, alteração e eliminação de notícias (apresentadas no *Info Desk*). Cada notícia contém vários parâmetros de formatação, podendo também ser declarada como um destaque.
- *Inserir Detalhe Notícia* – cada notícia pode conter vários detalhes que constituem o corpo da notícia. Os detalhes podem ser dispostos seguindo várias possibilidades de formatação.
- *Seleccionar Imagem* – representa a selecção de uma imagem para ilustrar a notícia. A imagem pode ser seleccionada a partir de outras imagens previamente inseridas, ou pode ser efectuado o *upload* de uma nova imagem.
- *Gerir Imagens* – o *Administrador* está também encarregue da gestão de imagens. As imagens inseridas na aplicação podem posteriormente ser utilizadas noutros conteúdos como notícias ou *banners* publicitários.
- *Gerir Publicidade* – em todas as aplicações poderá existir um *banner* para publicidade diversa, mostrado aleatoriamente. Os *links* e conteúdos correspondentes são geridos nesta secção. Cada elemento publicitário poderá ter mais que um *banner*.
- *Seleccionar Banner* – semelhante à selecção de imagens para notícias, mas orientado para imagens do tipo *banner*. Existe também a possibilidade de ser efectuado o *upload* de um novo *banner*.

Comparação com o sistema anterior

Na Linha Branca a desenvolver foi acrescentada a possibilidade de gerir a ajuda *on-line*, funcionalidade que não existia anteriormente. O sistema de publicidade foi também modificado para a inclusão de *banners* standard, em vez de *links* e imagens simples.

Quanto a opções retiradas, destaca-se a criação de segmentos e campanhas (transferidos para o *Supervisor Desktop*), e gestão de notas de clientes. Estas funcionalidades foram retiradas visto não se enquadrarem com as funções do *Administrador*.

3.2.1.7 *eMarketing*

A aplicação de *eMarketing* tem como objectivo guiar e auxiliar o *Agente* na promoção de campanhas. Nas funcionalidades previstas incluem-se a selecção de campanhas/clientes alvo, e o apoio ao *Agente* disponibilizando sequências de perguntas/respostas, argumentos a utilizar, ajuda *on-line*, etc.

A implementação deste módulo só deverá ser iniciada após a conclusão dos respectivos sistemas de suporte, como a criação de segmentos e campanhas no *Supervisor Desktop* e o sistema de apoio ao *Agente* (ainda não definido). Dadas estas circunstâncias, a aplicação de *eMarketing* será especificada numa fase posterior, não sendo abrangida por este documento.

3.2.1.8 *Simulação de Telefonia*

O objectivo deste pacote é a simulação do comportamento base da *applet* de controlo de telefonia, integrada nos Contact Centers da CPC TA através de uma terceira entidade. A simulação de telefonia é útil tanto para melhor demonstrar o funcionamento do controlo de telefonia a potenciais clientes, como para facilitar a eventual integração do sistema de controlo de telefonia nos Contact Centers das instituições. As funcionalidades gerais a simular estão representadas na *Figura 3.9*.

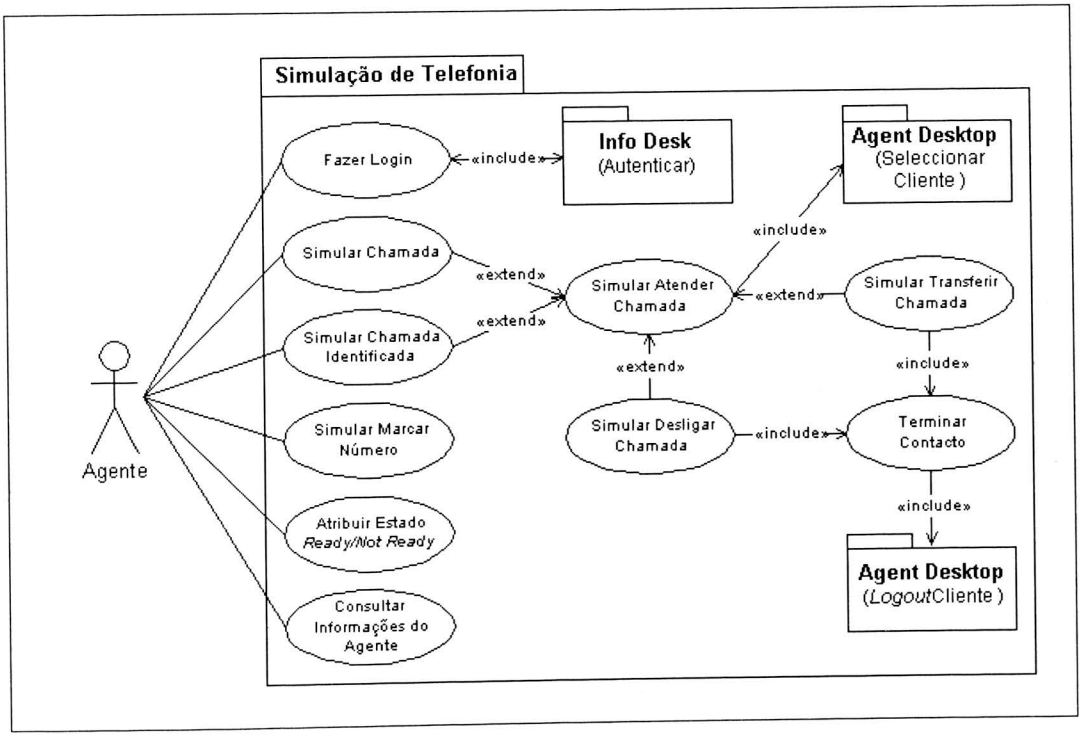


Figura 3.9 – Caso de Uso: Simulação de Telefonia

Casos de Uso

- *Fazer Login* – representa a autenticação de um *Agente* no sistema de controlo de telefonia. Apenas um *Agente* autenticado poderá controlar a telefonia. Ao autenticar-se

no controlo de telefonia, o *Agente* fica automaticamente autenticado no Contact Center, e vice-versa.

- *Simular Chamada* – corresponde à simulação de uma chamada recebida. Se a chamada não for atendida a simulação volta ao estado inicial.
- *Simular Chamada Identificada* – semelhante ao anterior, mas recebe um número de telefone como parâmetro.
- *Simular Atender Chamada* – ao atender uma chamada o *Agente* é redireccionado para a área de autenticação/selecção de clientes no *Agent Desktop*. Se a chamada for identificada com o número de telefone de um cliente registado no sistema, o campo relativo ao *username* é preenchido automaticamente.
- *Simular Transferir Chamada* – o *Agente* pode transferir chamadas para outros *Agentes* ou *Supervisores*.
- *Simular Desligar Chamada* – representa a terminação da ligação telefónica.
- *Terminar Contacto* - depois de desligar ou transferir uma chamada, o *Agente* pode tratar de operações pendentes antes de sair da área do cliente, terminando o contacto. Caso o *Agente* não termine o contacto passado um espaço de tempo, este é terminado automaticamente e a área do cliente é fechada.
- *Simular Marcar Número* – corresponde à marcação de um número de telefone.
- *Atribuir Estado (Ready/Not Ready)* – o estado do *Agente* pode ser *Ready*, ou *Not Ready*. Se o estado for *Ready*, todas as funções do controlo de telefonia devem estar disponíveis. O estado *Not Ready* significa que o *Agente* está indisponível para fazer ou receber chamadas.
- *Consultar Informações do Agente* – permite a visualização de dados gerais do *Agente*.

Comparação com o sistema anterior

Relativamente à antiga Linha Branca, o novo sistema de simulação de telefonia acrescenta a possibilidade recepção de chamadas identificadas (e respectiva interacção com o *Agent Desktop*), e a junção da autenticação da telefonia com a autenticação geral do Contact Center.

3.2.1.9 Outros Requisitos Funcionais

Existem ainda outras funcionalidades, não presentes na versão original da Linha Branca, que devem estar presentes na generalidade do sistema a desenvolver.

- *Help* – Sistema de ajuda *on-line* que o utilizador deverá poder utilizar de modo a receber apoio sobre o funcionamento da aplicação. Ao accionar o sistema de *help* numa determinada página da aplicação, o utilizador deverá receber ajuda sobre o objectivo da página (ajuda contextual). Caso não exista ajuda contextual sobre a página pretendida, deverá ser mostrada ajuda genérica sobre todos os temas disponíveis, com a possibilidade de pesquisa.
- *Paginação* – Sempre que possível, as listagens produzidas devem ser paginadas. Ou seja, deverão apenas ser mostrados um certo número de itens, podendo o utilizador seleccionar a página seguinte para visualizar os próximos itens.

- *Ordenação* – De modo semelhante, as listagens deverão poder ser ordenadas livremente pelo utilizador. Esta ordenação poderá ser feita por uma ou mais colunas, por ordem crescente ou decrescente.

3.2.2 Requisitos Não Funcionais

Existe uma série de requisitos não funcionais inerentes a qualquer projecto de *software*, estes dizem respeito à usabilidade, desempenho, fiabilidade e manutenção do sistema. É tido como um dado adquirido, por exemplo, que qualquer sistema deve ter um interface intuitivo, fácil de aprender e usar, assim como qualquer sistema deve prevenir a ocorrência de falhas graves.

No projecto em questão os requisitos tecnológicos principais prendem-se com o cumprimento das normas e métodos de desenvolvimento da plataforma WIS, preparando sempre que possível a compatibilidade com a WIS2. As características principais da plataforma WIS e as mudanças gerais previstas para a WIS2, estão definidas no Anexo B – "*Plataforma WIS*".

No que diz respeito à usabilidade e interface com o utilizador existem também algumas definições pré-conceptuadas que as aplicações deverão seguir.

3.2.2.1 Restrições Tecnológicas

O sistema a desenvolver terá como base uma versão melhorada da plataforma WIS, devendo ser usadas as tecnologias apropriadas para cada camada do *middleware*. Esta plataforma possui já algumas das funcionalidades que irão constar na futura WIS2, nomeadamente a nível da camada de sessão/apresentação.

Assim sendo, na camada de apresentação, na secção do *Presentation Manager* podem ser utilizadas quaisquer tecnologia orientada para geração de interfaces *web*: HTML, JSP, *JavaScript*, CSS, ASP, etc. Contudo as metodologias de inclusão de classes do *Session Manager* devem ser alteradas de modo a seguir uma filosofia de *JavaBeans*. Deve também ser feito um esforço para diminuir a quantidade de código *Java* incluído em ficheiros JSP, que não seja directamente relacionado com a lógica de apresentação. As páginas produzidas devem estar optimizadas para o *browser Microsoft Internet Explorer*, versão 5 ou superior.

A principal inovação dar-se-á no *Session Manager*, que implementará os *JavaBeans* a declarar nas páginas JSP. Cada *bean* deverá representar um componente de informação, populado com dados obtidos da camada transaccional, e seguindo os standards definidos.

Na camada transaccional não haverá grandes mudanças relativamente aos métodos de implementação dos serviços, constituídos por classes *Java* e estruturas IDL, sendo as ligações com as camadas de apresentação e Bases de Dados feitas através de *Corba* e JDBC respectivamente.

As Bases de Dados deverão receber algumas reestruturações, de modo a suportarem não só os melhoramentos efectuados para a nova Linha Branca, como também para preparar futuras expansões do sistema.

Certas funcionalidades específicas, envolvem também certos requisitos tecnológicos. Este é o caso do módulo de documentação, em que os documentos e *templates* produzidos devem ser editados seguindo um interface *Microsoft Word*, a integrar na aplicação.

3.2.2.2 *Interface com o Utilizador*

A estrutura base do interface com o utilizador deverá seguir um esquema semelhante à anterior Linha Branca. Contudo devem ser efectuadas algumas alterações no visual das aplicações. Seguidamente são apresentados alguns esboços dos interfaces gerais a criar.

O interface principal do portal de entrada, representado na *Figura 3.10*, é composto por vários componentes. Na parte superior existem áreas reservadas para o logotipo da CPC TA, *banners* publicitários e para a *applet* de controlo de telefonia. Imediatamente abaixo deverá constar um menu com todas as opções do *Info Desk*. No lado esquerdo encontra-se um segundo menu para a selecção das aplicações pretendidas. Em baixo existe uma área onde passam mensagens de texto em rodapé (*ticker*), e no canto inferior esquerdo encontram-se uma série de símbolos para efectuar várias operações (*home*, *logout*, sair, etc). À entrada do *Info Desk*, a área de conteúdos deverá conter quatro secções: notícias, destaques, links, e agenda (se o utilizador está autenticado) ou um formulário de *login* (se o utilizador não está autenticado).

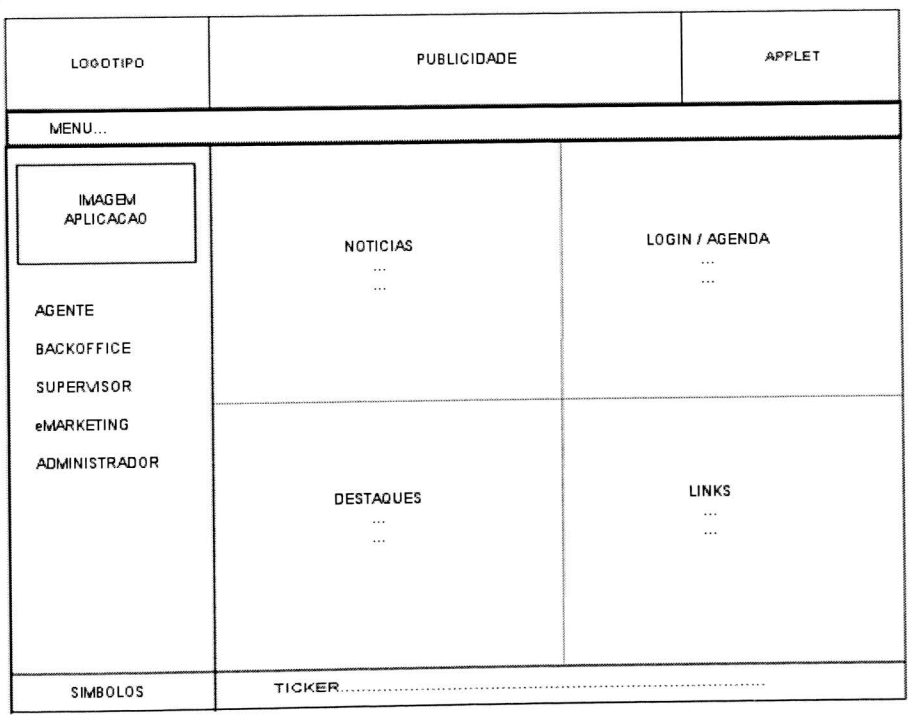


Figura 3.10 – Esboço do *Info Desk*

A parte superior (logotipo, publicidade e *applet*) deverá manter-se na maioria das restantes aplicações, contudo o menu com as opções da aplicação deverá ser colocado na parte esquerda do ecrã. Este *layout* geral das aplicações está representado na *Figura 3.11*. A área anteriormente alocada para o menu no portal deverá ficar reservada para opções gerais não directamente ligadas com a aplicação, como o sistema de ajuda *on-line* e informações do utilizador.

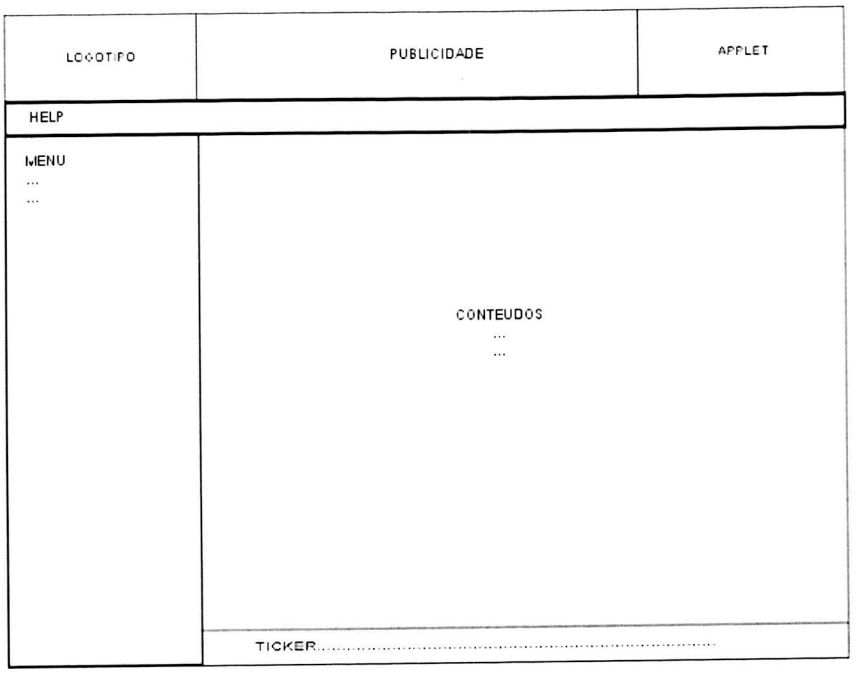


Figura 3.11 – Esboço Geral das aplicações

Um caso excepcional é a aplicação *Agent Desktop*, esboçada na *Figura 3.12*. Tendo em conta os conteúdos a disponibilizar não se considerou viável a utilização do menu lateral, devendo optar-se por uma área de dados alargada, e um menu superior semelhante ao *Info Desk*, com um sistema de *pop-downs* para submenus. No caso de existir um cliente identificado na aplicação, a área de publicidade deve ser substituída com informações do cliente. Na parte inferior deve estar também disponível o histórico de contactos do cliente, notas, e campanhas das quais o cliente seja alvo.

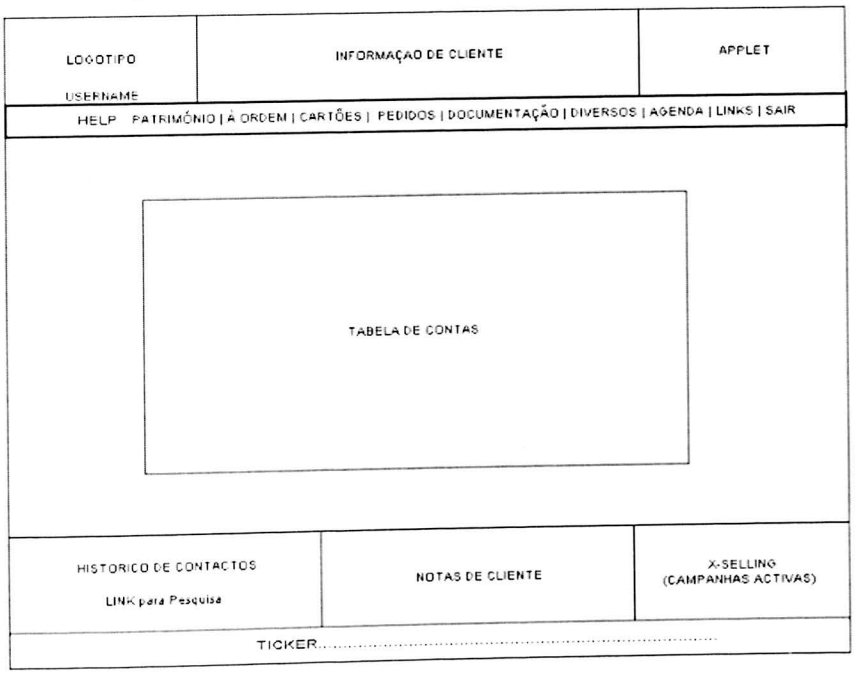


Figura 3.12 – Esboço do *Agent Desktop* (com cliente identificado)

Nos Contact Centers da CPC TA, a *applet* para o controlo da telefonia, faz parte de um sistema subcontratado a uma terceira entidade. Na Linha Branca é pretendido que o

sistema de simulação da *applet* tenha o aspecto o mais próximo possível ao sistema original, representado na *Figura 3.13*.



Figura 3.13 – Applet para o controlo de telefonia

De salientar ainda que o esquema de cores a utilizar deverá ir de encontro com as cores da CPC TA, à base de tons de azul e branco.

3.2.2.3 *Usabilidade*

Na primeira versão da Linha Branca foram detectados alguns problemas de usabilidade, que se espera corrigir no novo sistema.

Uma das deficiências encontradas diz respeito à passagem do portal de entrada (*Info Desk*) para as restantes aplicações. Sempre que o utilizador entrava numa aplicação, era aberta uma nova janela, de modo a ter o *Info Desk* sempre disponível numa janela, e a aplicação pretendida noutra. O facto de existirem duas janelas do Contact Center poderia originar alguma confusão no utilizador. Contudo o problema maior ocorria se o utilizador fechava a janela do portal. Visto que o *Info Desk* guardava alguma informação sobre a sessão do utilizador, o seu encerramento originava um comportamento errático da aplicação, muitas vezes sem que o utilizador se aperceba do motivo. Para resolver este problema na nova Linha Branca, a entrada nas aplicações a partir do *Info Desk* deverá ser efectuada na mesma janela, substituindo parcial, ou completamente a aplicação *Info Desk*.

Outro aspecto pouco "usável" era a criação de segmentos para campanhas. Esta funcionalidade estava implementada no *Administration Center*, e como já foi referido, deverá ser deslocada para o *Supervisor Desktop*, dado que a lógica dos segmentos e campanhas não se enquadra nas funções do Administrador. Na anterior Linha Branca os segmentos eram criados através de vários formulários independentes colocados numa mesma página, o que se poderia tornar bastante confuso. No novo sistema, o Supervisor poderá optar por inserir directamente um *query* SQL, ou, caso não domine esta linguagem, poderá criar o segmento através de um interface mais intuitivo, seguindo uma lógica passo-a-passo do tipo assistente (*wizard*).

Dada a natureza do conteúdo de certas páginas, o seu carregamento pode tornar-se moroso em casos de redes lentas ou sobrelotadas, ou havendo a necessidade de acesso a sistemas remotos. Para informar o utilizador que o sistema está à espera de dados, e não se encontra encravado, deverá existir uma página intermédia que indique ao utilizador que deve aguardar uns momentos.

4 Desenvolvimento do Contact Center – Linha Branca

Neste capítulo são referidas as fases mais relevantes do processo de desenvolvimento da Linha Branca, definindo as metodologias de desenvolvimento, a arquitectura utilizada e algumas das principais funcionalidades implementadas.

4.1 Modelo de Desenvolvimento

O rápido desenvolvimento de aplicações é uma das prioridades na CPC TA. Contudo, o desenvolvimento da Linha Branca está dependente de vários factores externos no que diz respeito à definição de requisitos e arquitectura. Apesar de existir um modelo base para o desenvolvimento, as funcionalidades previstas podem ser alteradas caso se considerem inadequadas. De mesmo modo, podem ser feitas mudanças significativas na arquitectura durante o desenvolvimento, para facilitar a integração com novas tecnologias eventualmente adoptadas para a WIS2.

A adopção de um modelo típico em cascata com fases distintas de análise de requisitos, definição da arquitectura, implementação e testes, não seria viável para este projecto, visto que uma especificação precisa de requisitos e arquitectura poderia tornar-se muito morosa. Deste modo resolveu adoptar-se uma solução de desenvolvimento evolutivo, partindo de uma especificação base inicial, e seguindo-se várias fases iterativas de especificação, implementação e validação (*Figura 4.1*). Este modelo adapta-se bem a muitos projectos na CPC TA, visto já existir uma especificação base de arquitectura (a plataforma WIS), e uma definição funcional genérica (a Linha Branca no caso de Contact Centers).

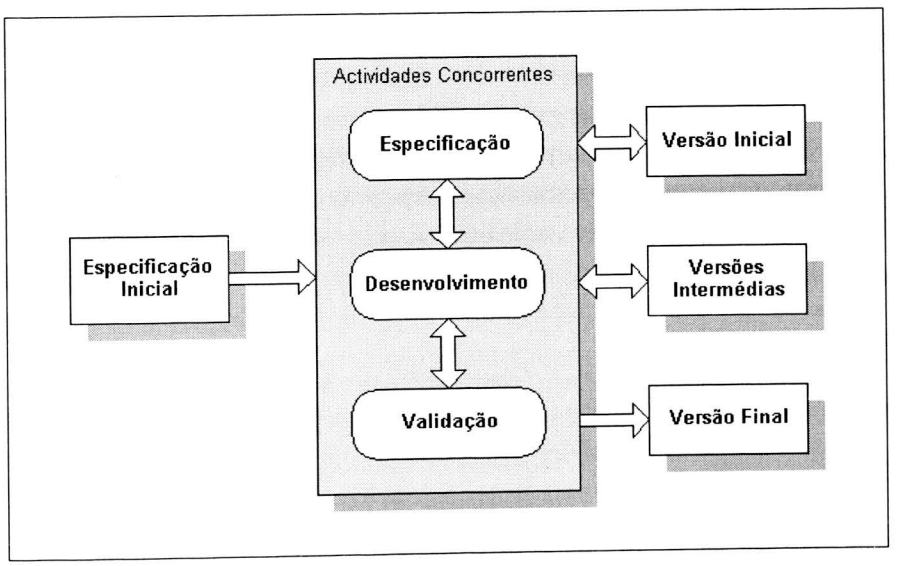


Figura 4.1 – Modelo de Desenvolvimento Evolutivo

Para melhor optimizar o tempo de desenvolvimento da Linha Branca, a implementação e especificação são muitas vezes efectuadas em paralelo, dependendo dos recursos disponíveis. Deste modo, ao mesmo tempo que se desenvolve um módulo do Contact Center, outros módulos podem estar a ser revistos e especificados.

4.2 Documentação

A produção de documentação formal da fase de desenvolvimento não costuma ser uma prioridade nos Contact Centers, optando-se por alocar o tempo disponível para adiantar a produção de código fonte. Contudo, alguma documentação para auxiliar futuras fases de desenvolvimento é um elemento valioso, capaz de poupar tempo gasto na análise o sistema. Em sistemas usados como base de desenvolvimento, como é o caso da Linha Branca, é de extrema importância documentar as normas e metodologias utilizadas.

Na CPC TA, os recursos humanos alocados a um projecto são muitas vezes variáveis, e muitas pessoas são frequentemente transferidas de uns projectos para outros, conforme as necessidades. Diferentes projectos e diferentes equipas de desenvolvimento seguem muitas vezes normas distintas. Para facilitar a integração de novos colaboradores nas normas e métodos utilizados no Contact Center e na Linha Branca, foi criado o documento apresentado no Anexo D – "*Guia do Programador*". Este documento tem o objectivo principal de constituir uma referência útil para programadores já familiarizados com as tecnologias base utilizadas, sendo os conteúdos baseados em exemplos práticos, evitando introduções e texto excessivo que poderiam tornar a consulta menos prática. O documento define normas genéricas de programação e também as normas, APIs e novas tecnologias disponíveis para o projecto específico da nova Linha Branca.

No que diz respeito à documentação da implementação propriamente dita, destaca-se a *standarização* de comentários de código através das normas *JavaDoc*[1]. Apesar da criação de comentários atrasar ligeiramente a produção de código fonte, uma documentação posterior de toda fase de desenvolvimento seria muito mais morosa e complexa. A produção de comentários seguindo as normas *JavaDoc* tem a grande vantagem de permitir a geração de um sistema de documentação formatado, descrevendo todas as classes, métodos e propriedades do código *Java* desenvolvido.

Adicionalmente, todos os ficheiros de código produzidos deverão conter um cabeçalho, indicando o nome do projecto/aplicação, descrição, autor, data de criação e eventuais autores e datas de alterações ao código original. Isto permite que, em análises futuras do código fonte produzido, se descubra facilmente o objectivo de cada ficheiro e quem foi o seu autor, de modo a esclarecer eventuais duvidas e responsabilidades.

4.3 Arquitectura

Nesta secção é descrita a estrutura base implementada para o desenvolvimento geral do sistema. A arquitectura do Contact Center está directamente ligada à plataforma WIS, e o entendimento de certas definições expostas nesta secção requerem alguma familiarização com os principais conceitos da WIS, descritos no Anexo B – "*Plataforma WIS*".

Como foi referido anteriormente, foram efectuadas algumas melhorias na plataforma WIS para garantir um melhor suporte, e acomodar a eventual migração de aplicações para a WIS2. Estas melhorias passam por uma melhor estruturação nas camadas de Apresentação/Sessão, e a utilização de *JavaBeans* para facilitar a ligação entre o *Presentation*

[1] A especificação destas normas estão disponíveis em *http://java.sun.com/j2se/javadoc/* [6].

e o *Session Manager*. Na camada transaccional não foram efectuadas mudanças significativas, esperando-se por uma especificação definitiva desta camada para a WIS2.

O Contact Center é constituído por vários módulos verticais integrados na WIS. Cada um dos módulos poderá ser utilizado por várias outras aplicações, devendo estar estruturado de modo a garantir independência do Contact Center. Nomeadamente, na Linha Branca são usados módulos comuns para o tratamento de contactos, *publishing* (conteúdos gerais: notícias, *links*, etc), e documentos.

4.3.1 Sistema de Gestão de Base de Dados

De modo a melhor estruturar módulos comuns, a Linha Branca utiliza várias Bases de Dados distintas, implementadas num servidor *Microsoft SQL Server 2000*. Visto poderem ser usadas por diversas aplicações, as tabelas foram preparadas para vários requisitos mais exigentes que o requerido pelo Contact Center. Os aspectos mais significativos de cada Base de Dados são descritos de seguida. A estruturação das tabelas mais relevantes estão representadas usando diagramas Entidade/Relação (ER).

4.3.1.1 WIS

A Base de Dados `WIS` suporta a gestão de utilizadores, aplicações, serviços, línguas, entre outras características base da plataforma.

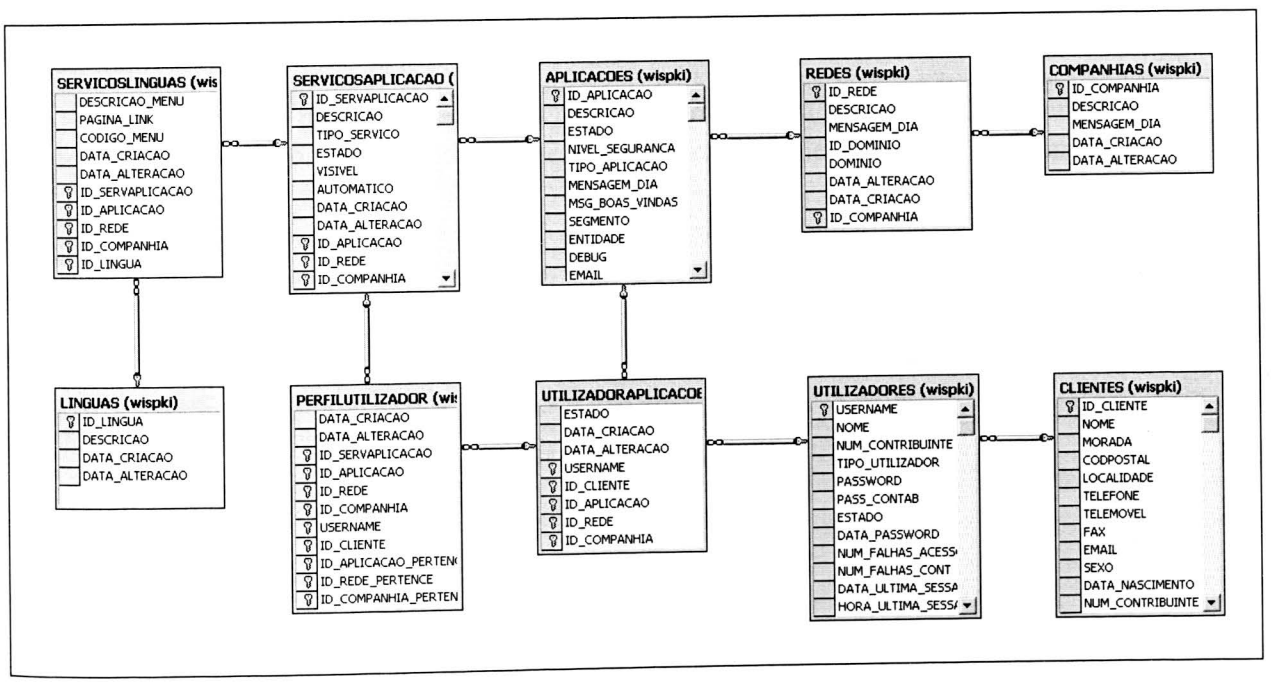


Figura 4.2 – Diagrama ER: Base de Dados WIS

Para que vários projectos possam coabitar na mesma Base de Dados, a cada projecto são atribuídos identificadores de rede e de companhia. Estes identificadores permitem vários projectos, possivelmente de diferentes empresas, possam utilizar Bases de Dados comuns garantindo a independência dos dados.

Cada projecto (identificado pelo par `id_rede` e `id_companhia`), poderá conter várias aplicações, que por sua vez poderão ter vários serviços[1]. Cada um destes serviços, na Linha Branca representa uma opção do menu da aplicação respectiva.

Para gestão de utilizadores, a Base de Dados WIS dispõe de duas tabelas principais, `utilizadores` e `clientes`. Cada utilizador da WIS deverá ter um cliente associado, em que um cliente poderá representar uma instituição ou secção. Por exemplo, os utilizadores dos agentes que utilizam o Contact Center – Linha Branca correspondem todos ao cliente CPC TA. Cada acesso de cada utilizador às aplicações é personalizável através das tabelas `utilizadoraplicacoes` (indica quais as aplicações acessíveis ao utilizador) e `perfilutilizador` (indica quais os serviços de cada aplicação aos quais o utilizador tem acesso). No Contact Center, os clientes propriamente ditos também têm atribuído um utilizador WIS, para que possam ser autenticados no sistema pelo Agente.

4.3.1.2 *WISJTS*

Esta Base de Dados é usada para a gestão de certas características internas do sistema, como contadores, objectos e dados de sessão.

4.3.1.3 *Contactos*

Todas as acções relacionadas com contactos efectuados em aplicações WIS, podem ser registados nesta Base de Dados. Contém suporte para guardar contactos e detalhes de todas as operações efectuadas durante um contacto, assim como as agendas e pedidos efectuados.

Assim como na maior parte dos módulos comuns da plataforma, uma única Base de Dados pode conter contactos efectuados em diversas aplicações, produtos e companhias.

[1] Estes serviços referem-se a serviços funcionais das aplicações, como opções de menus, não estando relacionados com os serviços implementados na camada transaccional.

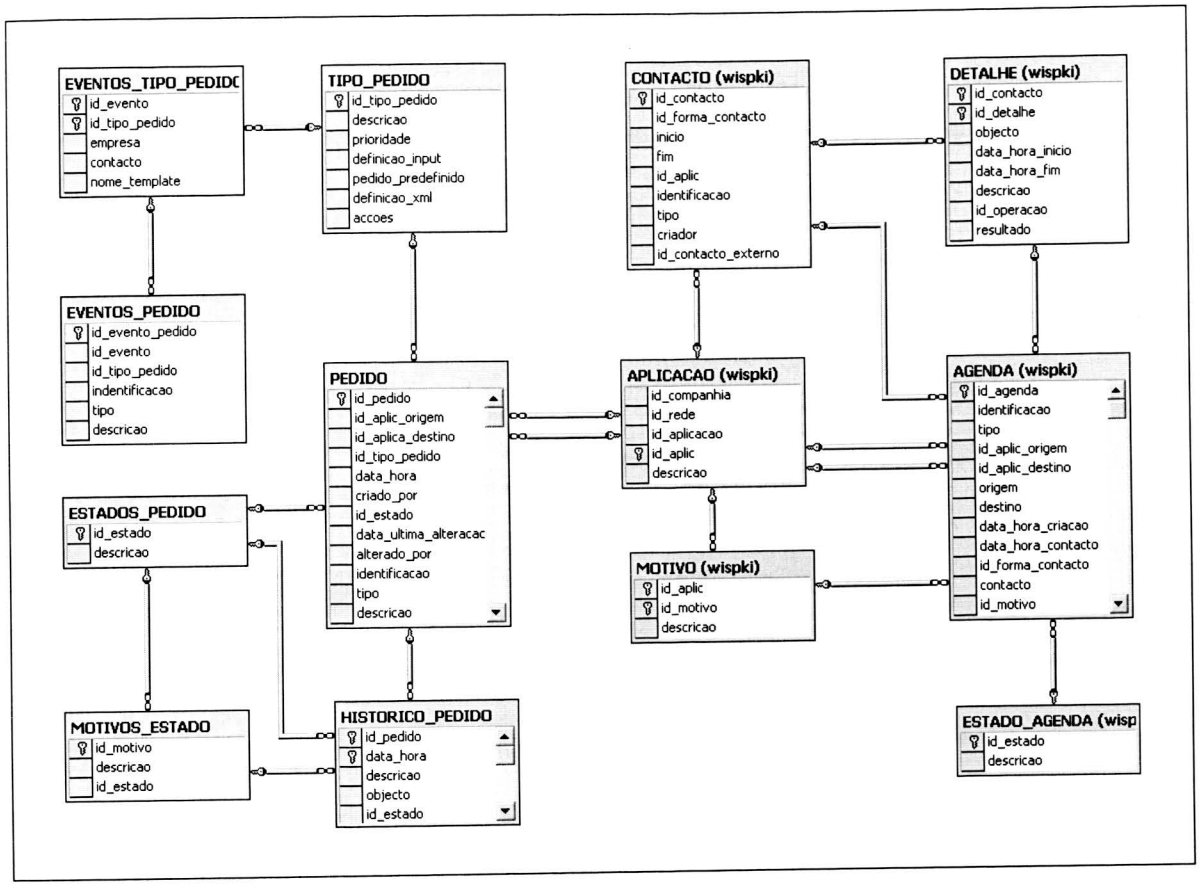


Figura 4.3 – Diagrama ER: Base de Dados CONTACTOS

Em cada contacto, agenda ou pedido efectuado podem ser guardados detalhes ou históricos com as operações efectuadas. Em cada um destes itens existe a possibilidade de guardar um `objecto` do tipo binário, podendo este conter qualquer tipo de dados necessários.

4.3.1.4 *Publishing*

A Base de Dados `Publishing` contém a informação necessária para a publicação de conteúdos como notícias, *links*, publicidade, imagens e *ticker*.

A Base de Dados está preparada para suportar que vários conteúdos possam pertencer a uma ou mais aplicações. Mesmo que esta característica não seja necessária, é incluída para o caso de novos Contact Centers ou outras aplicações eventualmente o exigirem.

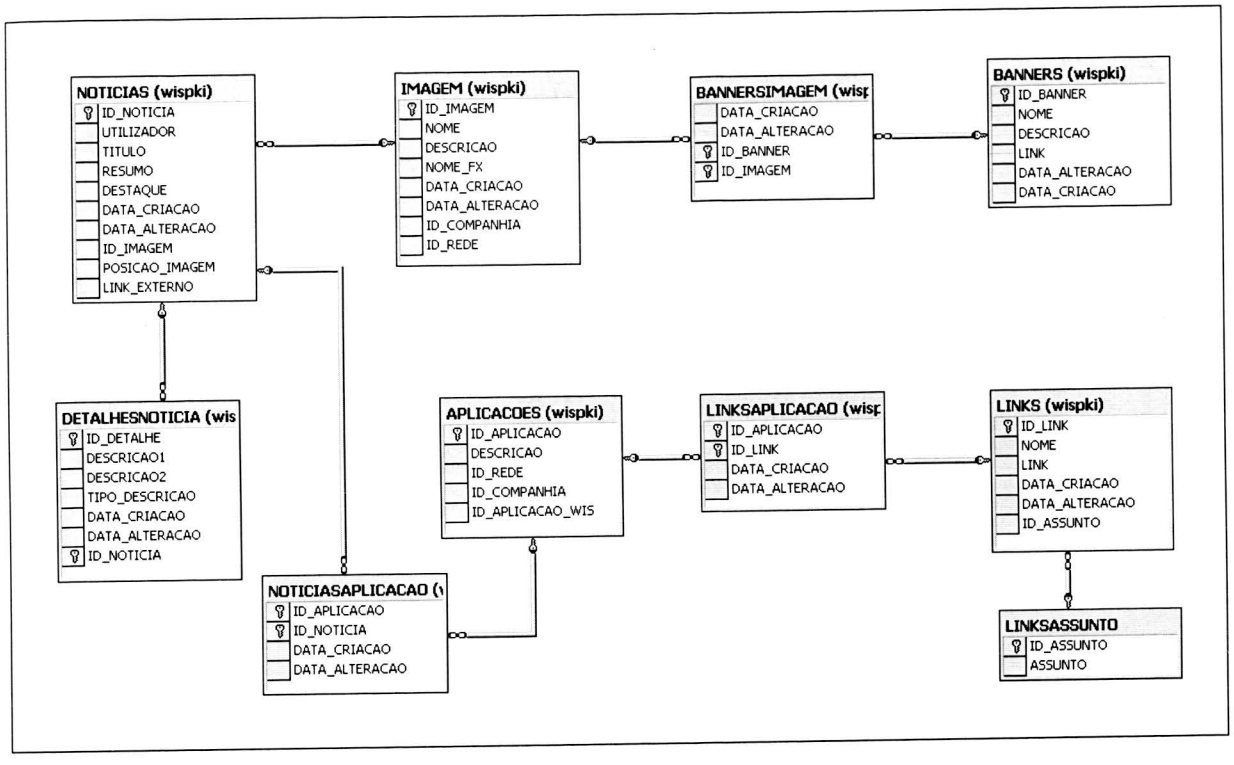


Figura 4.4 – Diagrama ER: Base de Dados PUBLISHING

Existe uma única tabela de imagens, podendo vários conteúdos estar associados a elementos desta tabela.

4.3.1.5 Documentos

O módulo de documentos regista todos os documentos, *templates* e respectivas *bookmarks*.

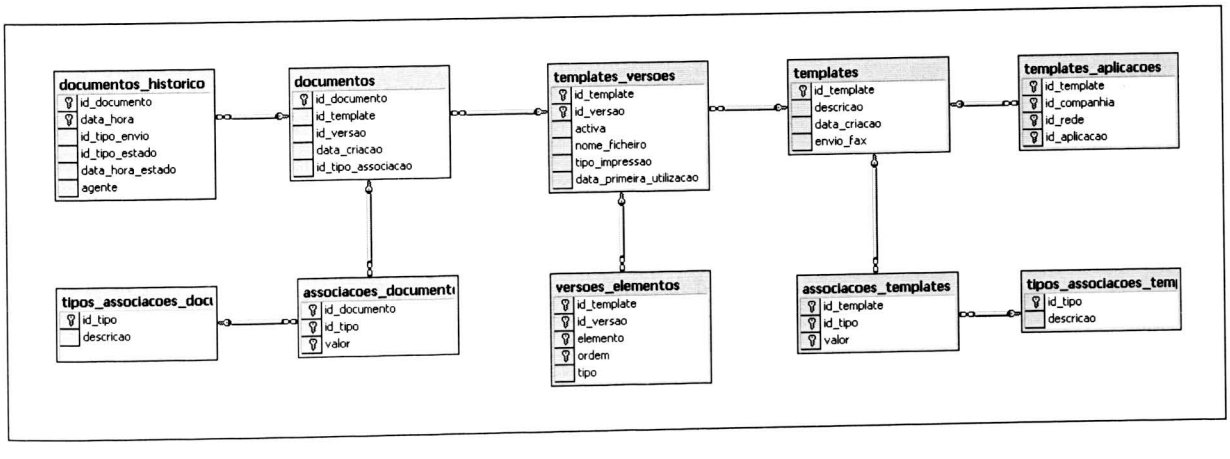


Figura 4.5 – Diagrama ER: Base de Dados DOCUMENTOS

Para cada *template* podem existir várias versões, podendo estas estar ou não activas. Tanto os documentos como os *templates* podem ter associações diversas, normalmente relacionadas com as unidades de negócio.

4.3.1.6 *contactCentreEMarketing*

A Base de Dados `contactCentreEmarketing` suporta todo o sistema de segmentos e campanhas do Contact Center.

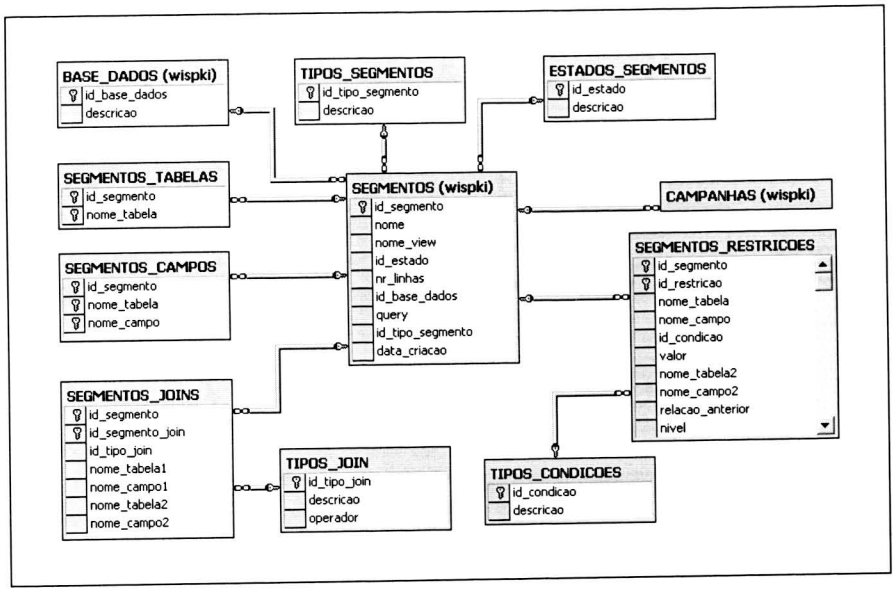


Figura 4.6 – Diagrama ER: Base de Dados CONTACTCENTREEMARKETING - SEGMENTOS

Um segmento pode ser criado directamente através de uma expressão SQL, ou através de um assistente (*wizard*) para a criação do segmento. No primeiro caso, a *query* SQL pode ser guardada directamente na tabela `segmento`, contudo, para permitir a posterior edição de segmentos através do assistente, é necessário guardar todas as operações efectuadas em cada passo do assistente. De modo a dar solução a esta particularidade foram criadas as respectivas tabelas para registar os campos, tabelas, relações (*joins*) e restrições de cada segmento.

A partir de um segmento podem ser criadas várias campanhas. Para o suportar os dados inerentes de uma campanha foram criadas tabelas para armazenar os mapas de perguntas/respostas e o sistema de ajuda *on-line*, para auxiliar o agente na promoção da campanha.

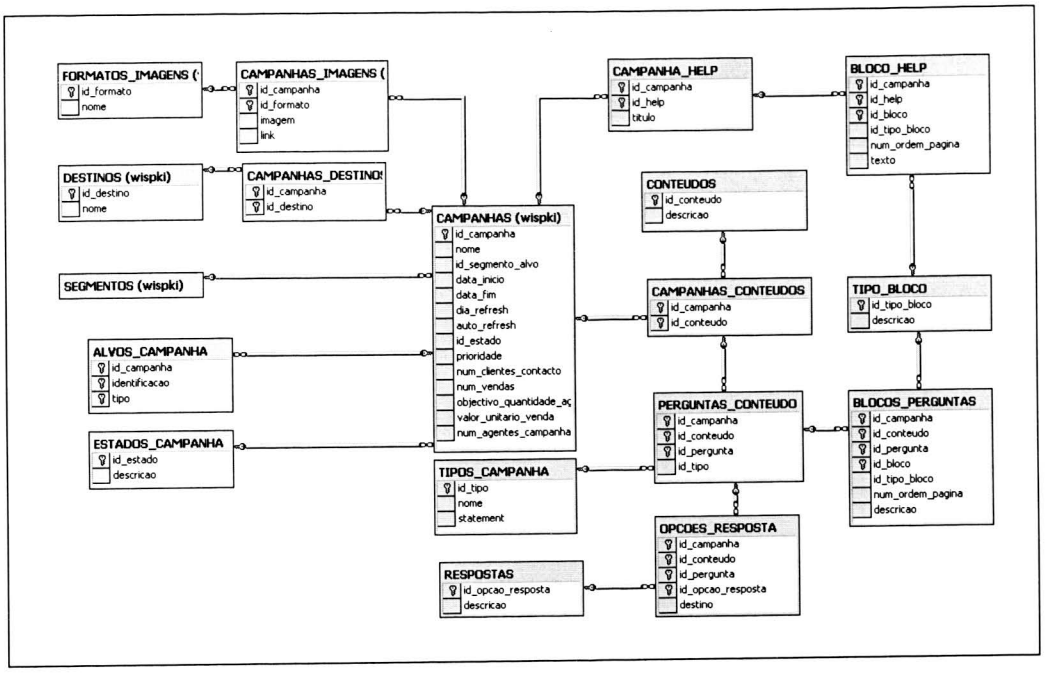


Figura 4.7 – Diagrama ER: Base de Dados CONTACTCENTREEMARKETING - CAMPANHAS

Existe também uma tabela `alvos_campanha`, onde são referidos os clientes potencialmente mais interessados na campanha, ou seja, as entidades abrangidas pelo segmento criado. Apesar desta selecção poder ser determinada em *run-time*, a sua execução frequente poderia sobrecarregar o SGBD, e diminuir a performance. Deste modo optou-se por uma actualização periódica da tabela `alvos_campanha`, permitindo assim uma referência rápida e razoavelmente actualizada dos clientes alvos de cada campanha.

4.3.1.7 *contactCentreLinhaBranca*

Nesta Base de Dados são guardados os dados específicos relativos ao Contact Center – Linha Branca, não sendo prevista a sua reutilização em futuros Contact Centers nem noutros produtos. Numa eventual implementação numa instituição esta Base de Dados poderia ser substituída por Bases de Dados apropriadas ao negócio da instituição, ou como acontece na maioria dos casos, é criada uma ligação a sistemas de informação remotos.

Como componente não reutilizável, a Base de Dados `contactCentreLinhaBranca` contém tabelas relativas a características típicas de uma instituição financeira genérica, com o objectivo de suportar a demonstração de operações financeiras no Contact Center. As tabelas criadas correspondem a contas à ordem e cartões de crédito ou débito.

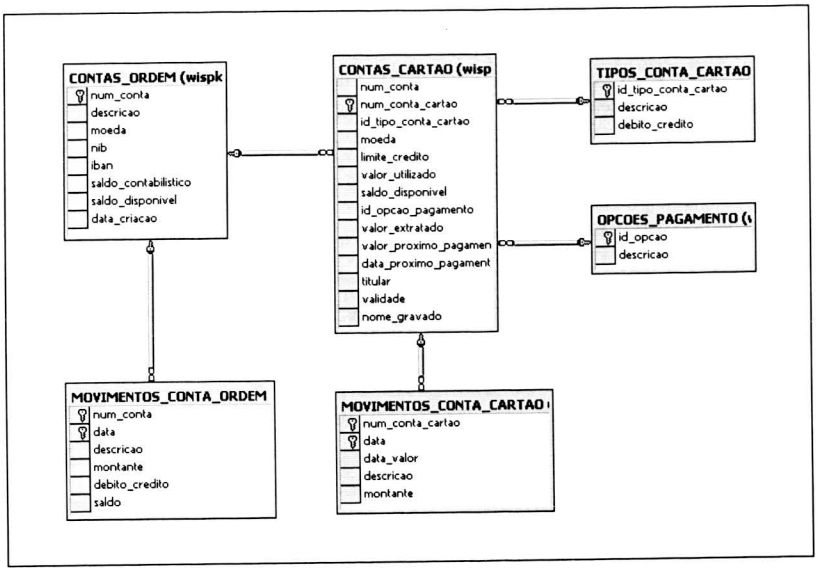


Figura 4.8 – Diagrama ER: Base de Dados CONTACTCENTRELINHABRANCA

4.3.2 Camada Transaccional

Ao nível da arquitectura transaccional, a WIS melhorada não trouxe grandes mudanças relativamente à versão utilizada anteriormente.

A camada transaccional é composta por serviços (implementados por métodos *Java*) que transportam e traduzem dados entre as camadas de apresentação e os SGBD, consoante a lógica de negócio. O motor transaccional (*Transaction Server*) permite a comunicação com os SGBD através de JDBC, e com a camada de apresentação através de *Corba*, utilizando estruturas IDL. As características desta camada são definidas em mais pormenor no Anexo B: "*Plataforma WIS*".

Dado que a Linha Branca é um projecto interno, não existem sistemas de informação remotos. Não foi portanto necessário a implementação de nenhum motor de dados para a ligação a sistemas externos.

4.3.3 Camada de Apresentação

A camada de apresentação está dividida em duas sub-camadas, *Session Manager* e *Presentation Manager*. Esta separação em sub-camadas ocorre apenas ao nível lógico, sendo ambas as camadas suportadas pelos mesmos servidores *web*, existindo assim uma partilha directa de informação entre sub-camadas.

Na versão melhorada da WIS, a interligação entre *Session Manager* e *Presentation Manager* é implementada através de *JavaBeans*, de modo a permitir uma melhor portabilidade e estruturação de dados.

Devido a restrições tecnológicas da WIS, nomeadamente no balanceamento de carga entre servidores *web*, a persistência de *beans* é comprometida. Ao ser instanciado num servidor *web*, um *JavaBean* pode abranger vários níveis de persistência, nomeadamente, pode

ter um espectro de página, pedido (*request*), sessão[1] ou aplicação. Ao declarar um *bean* com um espectro de página ou pedido, os dados contidos no *bean* apenas são mantidos para a página actual, ou enquanto não for efectuado um novo pedido. Contudo, um *bean* com espectro de sessão já seria persistente para a duração da sessão, até que fosse eliminado. Isto seria muito vantajoso, permitindo que as informações do *JavaBean* fossem guardadas como variáveis de sessão do *browser*, reduzindo a carga transaccional do sistema. Contudo, o espectro de sessão não pode ser utilizado na WIS pois com o balanceamento de carga, pode ocorrer que um utilizador seja servido por mais que um *web server*, em que as sessões não são partilhadas entre servidores. Espera-se que esta limitação seja superada com o lançamento da WIS2.

4.3.3.1 Session Manager

O *Session Manager* é composto por *JavaBeans*, permitindo uma mais fácil instanciação e representação de dados nos JSPs no *Presentation Manager*. O interface para a invocação de serviços da camada transaccional foi também melhorado, proporcionando uma implementação mais simples e estruturada.

Um *JavaBean* é implementado através de uma classe *Java*, de modo a criar um componente reutilizável, suportando persistência de dados. No âmbito do desenvolvimento assente na plataforma WIS, os *JavaBeans* que constituem o *Session Manager* são compostos por propriedades privadas com respectivos métodos (públicos) `set` e `get` para obter e atribuir valores. Estas propriedades podem conter diversos tipos de dados enviados ou recebidos do *Presentation Manager* ou da camada transaccional. Os *beans* podem também conter métodos que se considerem necessários para a pré-formatação de dados antes de serem encaminhados para a camada respectiva. A instanciação de um *bean*, é efectuada através de páginas JSP no *Presentation Manager*, existindo uma ligação directa com o *Session Manager* inerente ao servidor *web*.

Todos os *JavaBeans* do *Session Manager* devem estender a classe base `ServicoPS`, de modo a obter acesso a métodos e propriedades da sessão, herdando vários métodos e variáveis. Esta nova característica da WIS tem objectivo de tornar mais transparentes vários aspectos no que diz respeito à invocação de serviços e propriedades gerais de aplicações.

A invocação de um serviço deverá também seguir algumas normas definidas para o efeito. Para cada serviço invocado, um *bean* deverá ter um método, que poderá receber como parâmetro uma instância de uma ligação de sessão WIS (`LigacaoSessaoTS`), para a comunicação por *Corba* com a camada transaccional. Este objecto de ligação é obtido da *pool* de ligações ao *Transaction Server* através classe pai `ServicoPS`. Para este efeito a classe `SerivcoPS` disponibiliza os métodos `invocaMetodoTS()` e `invocaMetodoLSTS()`, respectivamente para invocar serviços utilizando uma nova sessão ou usando a sessão do utilizador WIS autenticado no sistema. Estes métodos vão por sua vez chamar o método criado para a invocação do serviço na classe que implementa o *bean*, com a respectiva instância da `LigacaoSessaoTS` como parâmetro. A *Figura 4.9* ilustra a sequência de procedimentos efectuados para a invocação de um serviço transaccional. Exemplos práticos

[1] Esta sessão refere-se à sessão criada em cada conexão entre um servidor *web* e um *web browser*, não estando relacionada com a sessão interna de cada utilizador WIS.

da utilização desta metodologia estão demonstrados no Anexo D – "*Guia do Programador*", na secção dedicada ao *Java Session Manager* (JSM).

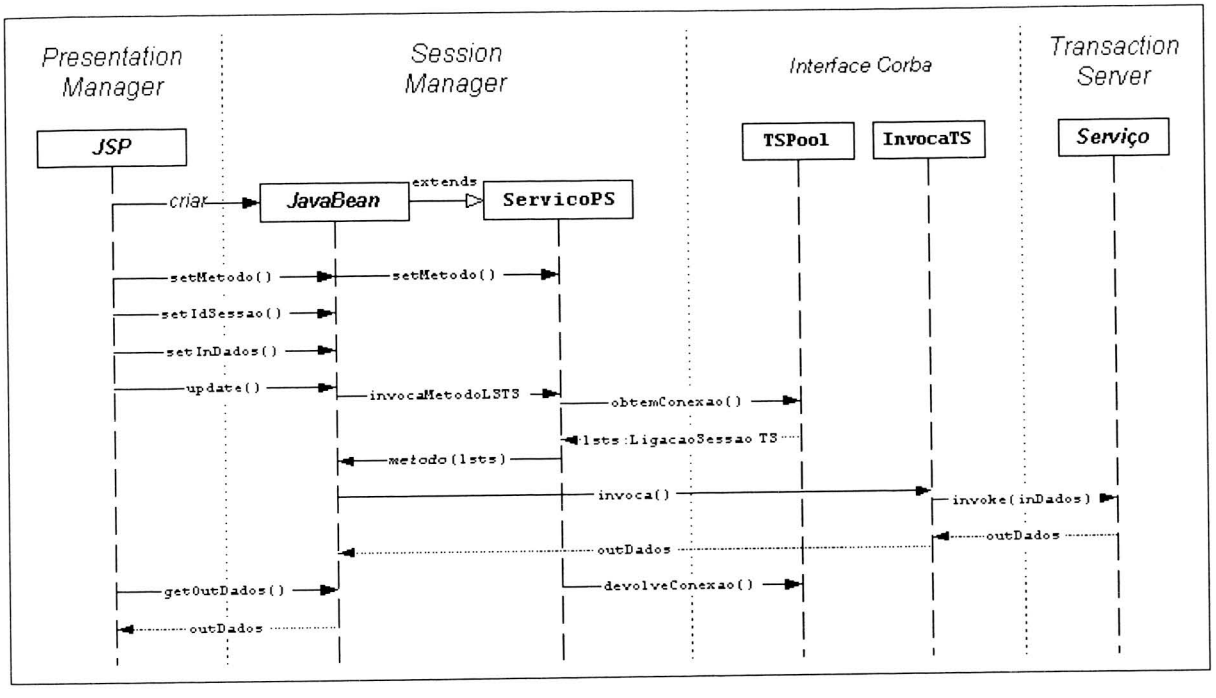


Figura 4.9 – Diagrama de Sequenciação: Invocação de um Serviço

4.3.3.2 *Presentation Manager*

No *Presentation Manager* são colocados todos os conteúdos relacionados directamente com o interface com o utilizador. Nomeadamente, páginas e *scriplets web* (HTML, JSP, ASP, etc), imagens, CSSs e ficheiros *JavaScript*. Contudo, o mais utilizado para a definição dos interfaces com o utilizador são as *scriplets* JSP.

O uso de *JavaBeans* permite uma considerável diminuição do código *Java* embutido em páginas JSP, fornecendo também um interface mais normalizado ao *Session Manager*. Com as novas normas para a invocação de serviços, o acesso a dados obtidos na camada transaccional é mais simples e transparente, permitindo uma melhor separação entre as sub-camadas de Apresentação/Sessão. A secção do *Java Presentation Manager* (JPM) do "*Guia do Programador*" (disponível em anexo), apresenta as metodologias utilizadas na instanciação de *beans* e invocação de serviços, assim como outras normas utilizadas para o desenvolvimento de páginas na Linha Branca.

4.3.4 Estrutura de Directórios

Para uma melhor organização do código produzido, foi criada uma estrutura de directórios, seguindo normas standard, também previstas para a WIS2. O projecto foi desenvolvido no servidor `Pandora` na CPC TA, sendo o directório base `\apps\wis`.

Devido à divisão da CPC IS em sub-empresas (entre as quais a CPC TA), os nomes dos *packages* base das classes *Java* foram alterados de `cpcis.wis` para `pt.cpcta.wis`, de modo a melhor seguir as normas internacionais de nomeação de *packages*.

Dentro do directório base existem diversos directórios cada um com uma função específica na plataforma. Nomeadamente:

- **etc** – Contém os ficheiros para a configuração de vários parâmetros relativos ao *Session Manager* (`etc\sm`) e ao *Transaction Server* (`etc\ts`). Estas propriedades são configuráveis aos níveis da plataforma, projecto e aplicação, tendo a estrutura de directórios uma forma do tipo `etc\camada\projecto\aplicação`. Por exemplo, as propriedades do *Session Manager* características de todo o projecto Linha Branca estão colocadas no directório `etc\sm\linhaBranca`; As propriedades e configuração dos serviços ao nível da aplicação *BackOffice*, por exemplo encontram-se no directório `etc\ts\linhaBranca\backOffice`.
- **var\logs** – Neste directório são colocados os *logs* produzidos pelos servidores do Contact Center. Assim como no caso anterior, os subdirectórios seguem uma estrutura do tipo `var\logs\camada\projecto\aplicação`. Contudo, esta estrutura depende do número de instâncias de servidores utilizados em cada caso. No desenvolvimento da Linha Branca foi utilizado apenas uma instância do servidor *web* para todo o projecto, e uma instância do *Java Transaction Server* para cada módulo aplicacional. Ou seja, os *logs* da camada de apresentação são colocados em `var\logs\sm\linhaBranca`, enquanto que para a camada transaccional os *logs* encontram-se em directórios do tipo `var\logs\ts\linhaBranca\nomeAplicação`.
- **ts** – Contém as classes *Java* que implementam o *Transaction Server*. De modo a gerar os *packages* de forma apropriada o subdirectório base para as classes é `ts\pt\cpcta\wis\jts`.
- **ts\pt\cpcta\wis\jts\pki** – As classes neste directório implementam parte do *core* do *Transaction Server*, sobretudo ao nível da ligação ao *Session Manager* através de *Corba*.
- **ts\pt\cpcta\wis\jts\sql** – Contém APIs para o tratamento de conexões a Bases de Dados, implementando também a *pool* de conexões.
- **ts\pt\cpcta\wis\jts\util** – Este *package* disponibiliza algumas classes úteis para a camada transaccional.
- **ts\pt\cpcta\wis\servicos** – Este é o directório base para os serviços implementados. Cada serviço deverá estar organizado numa estrutura de subdirectórios com a forma `servicos\projecto\aplicacao\nomeServico`. Certos serviços podem ser comuns a várias aplicações, e mesmo a vários projectos, sendo estes colocados em subdirectórios com o nome "`comum`", dentro do nível respectivo. Por exemplo os serviços para o tratamento de *links*, fazem parte do módulo de *publishing*, comum a vários projectos. De modo que estes serviços estão colocados no directório `ts\pt\cpcta\wis\servicos\comum\publishing\trataLinks`.
- **idl** – Contém os ficheiros IDL que definem as estruturas utilizadas na comunicação entre as camadas transaccional e de apresentação. Contém também um subdirectório "`base`" com estruturas e definições gerais. Existe ainda o subdirectório "`servicos`", que deverá conter os IDLs respectivos a cada serviço criado, utilizando uma estrutura de directórios semelhante à da classe que implementa o serviço (`servicos\projecto\aplicacao\nomeServico`).

- **gerado** – Contém as classes *Java* produzidas na compilação de um IDL, incluindo a estrutura de subdirectórios correspondente ao *package* da classe.
- **sm** – Este é o directório base para os *packages* das classes do *Session Manager*. Os nomes dos *packages* seguem uma estrutura do tipo `sm\pt\cpcta\wis\jsm\projecto\aplicação`. Assim como nos *packages* dos serviços transaccionais, existem subdirectórios para classes comuns a várias aplicações ou a vários projectos.
- **pm** – Contém todos os ficheiros que implementam o *Presentation Manager*. Os ficheiros correspondentes às páginas de interface com o utilizador (JSP, HTML, ASP, etc) são divididos por projecto e aplicação, na forma `pm\projecto\aplicacao\jsp`. De igual forma as imagens de cada aplicação foram colocadas em subdirectórios do tipo `pm\projecto\aplicação\imagens`. Ficheiros de estilos e de *JavaScript* comuns a todas as aplicações são colocados nos directórios `pm\linhaBranca\css` e `pm\linhaBranca\js`, respectivamente. Ficheiros obtidos através de *uploads*, como imagens e *templates*, são colocados em `pm\linhaBranca\repositorio`.

4.4 *Implementação*

Nesta secção é descrito o ambiente de desenvolvimento e os aspectos mais significativos da implementação de APIs e funcionalidades para o Contact Center – Linha Branca.

4.4.1 Ambiente de Desenvolvimento

O projecto foi desenvolvido num PC *Compaq EYD Series* com um processador *Intel Pentium III* a 450Mhz, e 128Mb de memória RAM. O sistema operativo utilizado foi o *Microsoft Windows 2000 Professional*.

Para a edição de código, seja *Java*, JSP, IDL, HTML ou outros, foi utilizado o editor genérico *EditPlus 2.01a*. Esta ferramenta foi configurada para utilizar o compilador de *Java* incluído no *Java 2 SDK, Standard Edition 1.3* da *Sun Microsystems*, e o compilador de IDL/*Java* presente na plataforma *Borland Inprise VisiBroker for Java 4.0*.

Para suporte aos servidores necessários para o funcionamento da WIS, foram utilizadas duas máquinas da CPC TA, intituladas `Pandora` e `Athena`. O `Athena` disponibiliza o servidor de Bases de Dados, através do *Microsoft SQL Server 2000*. Na máquina `Pandora` foram colocados os servidores *web* e aplicacional *Corba*, utilizando o *Allaire JRun 3.1* e o *Borland Inprise VisiBroker for Java 4.0* respectivamente.

4.4.2 Aplication Programming Interface

As APIs (*Application Programming Interface*) produzidas correspondem a métodos genéricos com o objectivo melhorar a estruturação do código fonte e facilitar o desenvolvimento geral do sistema. Seguidamente são descritos as principais classes e módulos utilitários implementados[1]:

[1] A utilização destas APIs é demonstrada com maior pormenor no Anexo D – "*Guia do Programador*".

- **`FormataTexto.java`** – Trata-se de uma classe *Java* comum a todas as aplicações WIS, com uma diversa colecção de métodos estáticos para o tratamento e formatação de texto. Nesta classe foi acrescentado um método que recebe texto com o formato geral de parâmetros recebidos num URL (`param1=valor1&param2=valor2&...`), e transforma cada par parâmetro/valor em campos de *input hidden* de modo a serem acrescentados em formulários HTML. Esta função é útil em muitos casos, quando se pretende mudar as chamadas a páginas de *hrefs* para formulários, ou para integrar *links* produzidos por outros utilitários em *forms* HTML. A classe `UtilLinhaBranca` contem também um método semelhante, mas refinado para suportar características adicionais, relativas à página intermédia utilizada. Anteriormente, a classe `FormataTexto` continha também vários métodos para a formatação de texto, como a substituição de *Strings*, tradução de caracteres especais para HTML, *JavaScript*, etc...

- **Ordenação** – A generalidade das listagens do Contact Center desenvolvido suportam a possibilidade de ordenação, ascendente ou descendente, por um ou mais campos em simultâneo. Para converter a descrição dos campos seleccionados numa *String* com os nomes efectivos dos campos na Base de Dados, foi criada a função `geraOrderBy`, na classe `ListagemUtil`. Nesta classe foram ainda incluídos métodos generalizados para gerar o símbolo que permite ao utilizador ter acesso à selecção de campos. Este símbolo é colocado no canto superior esquerdo de cada listagem. Estes métodos são normalmente chamados indirectamente, através da classe `UtilLinhaBranca`, que envia parâmetros predefinidos para a utilização específica no projecto Linha Branca.

- **`UtilLinhaBranca.java`** – Para além de refinamentos de métodos definidos nas classes de uso geral já referidas, esta classe dispõe também de outros métodos para a conversão de dados para formatos específicos do Contact Center – Linha Branca. Como funções para geração automática dos cabeçalhos comuns à maioria das páginas, e para formatação de datas.

- **`inicioPagina.jsp`** – Todas as páginas JSP da Linha Branca devem incluir este ficheiro. Este JSP fornece acesso a várias propriedades, objectos, variáveis e *cookies* de uso geral para as aplicações.

- **`intermedia.jsp`** – Esta página tem o objectivo de informar o utilizador que o sistema está a carregar os dados da página seguinte, devendo ser incluído na generalidade dos *links* para novas páginas. Esta página aceita como parâmetro o nome da página de destino, assim como os respectivos parâmetros a reenviar.

- **`voltarUtil.jsp`** – Em alguns casos, a navegação para efectuar certas operações pode envolver uma grande sequência de páginas, podendo ser necessário manter o estado de páginas iniciais ou intermédias. Anteriormente, a única maneira para resolver era reenviar todos os parâmetros de cada página para a página seguinte. Este método pode tornar-se bastante complexo em páginas com um maior nível de profundidade, em que se tem que tratar parâmetros recebidos de várias páginas anteriores. Para resolver este problema foi criada uma *script* JSP que gera automaticamente os parâmetros recebidos de páginas anteriores de modo a, em cada ficheiro, criar *links* para voltar à página actual, voltar à pagina anterior, ou manter o estado da navegação.

Outros métodos utilitários não referidos incluem classes e *scripts* para formatação de dados não relacionados com interfaces, conversão e formatação de datas, validação de formulários, paginação de listagens, geração de menus e submenus, etc.

4.4.3 Funcionalidades

Após a definição e implementação da arquitectura e APIs a utilizar, basta apenas desenvolver os interfaces e funcionalidades a incluir no sistema.

A estrutura e a disposição geral do interface com o utilizador são exemplificados no Anexo C, "*Nova Linha Branca Vs Antiga Linha Branca*", onde na secção de *Interfaces*, são comparados ecrãs do Contact Center desenvolvido com o interface da anterior Linha Branca.

A implementação de uma funcionalidade típica, engloba os seguintes procedimentos:

1. Criação de estruturas de dados, colocadas num ficheiro IDL, para guardar a informação transferida entre o *Transaction Server* e o *Session Manager*, através de *Corba*.

2. Criação de uma classe *Java* para implementação dos serviços necessários para a obtenção ou actualização de informação na Base de Dados.

3. Criação de um ou mais *JavaBeans*[1] para invocar os serviços implementados no ponto anterior, e disponibilizar a informação para a camada superior.

4. Criação de páginas JSP para a obtenção da dados disponibilizados através dos *JavaBeans*, e a respectiva geração de interfaces com o utilizador e disposição da informação obtida.

Dado não terem ocorrido muitas alterações estruturais no desenvolvimento de serviços transaccionais, certos serviços puderam ser reutilizados a partir de anteriores Contact Centers. Contudo, foi quase sempre necessário adaptar os serviços importados às novas necessidades e requisitos da Linha Branca. Quanto às páginas JSP e *JavaBeans*, não se verificaram muitas possibilidades de reutilização, dadas as diferentes metodologias e interfaces utilizados.

Na fase actual de desenvolvimento, foram implementados total ou parcialmente, todos os módulos aplicacionais especificados no capitulo anterior, "*Apresentação do Problema*".

As funcionalidades não implementadas dizem respeito a módulos verticais, com dependências em várias aplicações. Este é o caso dos requisitos relacionados com campanhas de *eMarketing*, cujo desenvolvimento se encontra na fase inicial para o *Supervisor Desktop*. De igual forma no *Agent Desktop*, a promoção de campanhas não é suportada (dado estas ainda não existirem). Ainda na aplicação para os supervisores, as opções para a criação e consulta de *querys* e mapas também não foram implementadas, devendo ser posteriormente adaptadas a partir da criação de segmentos, dadas as semelhanças tecnológicas. Finalmente, o módulo de dados estatísticos presente no *Supervisor Desktop* e *Administration Center*, não foi implementado dado não ser considerado prioritário.

Todas as restantes funcionalidades foram implementadas conforme as especificações de requisitos funcionais e não funcionais.

---

[1] Normalmente é criado um *bean* para cada serviço.

5 Conclusões e Perspectivas de Trabalho Futuro

Para finalizar, são apresentadas as conclusões do trabalho efectuado no âmbito do estágio curricular, referindo também possíveis melhoramentos e evoluções do sistema desenvolvido.

5.1 Conclusões do Trabalho Realizado

O trabalho efectuado teve como objectivo o desenvolvimento do projecto Contact Center – Linha Branca. A Linha Branca corresponde a um Contact Center genérico, utilizado como base de desenvolvimento de outros Contact Centers à medida das instituições.

O sistema desenvolvido teve como base uma versão melhorada da plataforma WIS, de modo a preparar a migração para a futura WIS2. Houve uma fase inicial de adaptação e familiarização com as novas metodologias da WIS, seguindo-se uma reestruturação da arquitectura do Contact Center para melhor se adaptar às novas definições da WIS e a exigências futuras. O aspecto mais significativo foi a mudança das camadas de apresentação/sessão para seguir uma filosofia de *JavaBeans*. Esta mudança trouxe benefícios para uma melhor estruturação de código fonte, maior separação de camadas e uma melhor representação de informação.

A nível funcional, na porção do projecto implementada até à data inclui-se o portal de entrada (*Info Desk*), a aplicação para a comunicação com os clientes (*Agent Desktop*), o tratamento de pedidos (*BackOffice*) e a administração do sistema (*Administration Center*). A aplicação para supervisão e definição de parâmetros do negócio (*Supervisor Desktop*), está parcialmente concluída. Apesar de alguns módulos ainda não estarem implementados, os restantes módulos encontram-se em pleno funcionamento. A Linha Branca está assim apta para servir como base para a criação de novos Contact Centers nas instituições, cumprindo assim o principal objectivo do projecto. De facto a nova Linha Branca foi já usada em algumas demonstrações a potenciais clientes, apresentando resultados positivos. As funcionalidades de *eMarketing*, ainda não implementadas, podem sempre ser implementadas "a pedido", caso uma instituição eventualmente o exija.

Numa perspectiva global, a estruturação da arquitectura e APIs utilizadas contribuíram para uma melhor uniformização dos processos de trabalho, nomeadamente na implementação de metodologias que permitam uma cada vez maior facilidade e velocidade de desenvolvimento, nunca abdicando da qualidade.

5.2 Perspectivas de Trabalho Futuro

Dado que a implementação do projecto Linha Branca se prolonga para além da duração do estágio curricular, está já previsto o desenvolvimento de novas funcionalidades para complementar o sistema.

A curto prazo, o desenvolvimento deverá concentrar-se na implementação da aplicação de *eMarketing* e o respectivo suporte. Deste modo, é prevista a implementação das seguintes funcionalidades:

- Criação de Segmentos e Campanhas no *Supervisor Desktop*.

- Apresentação das Campanhas activas no *Agent Desktop*.
- Definição de Grafos de perguntas/respostas para Campanhas. Esta funcionalidade está ainda a ser analisada, e poderá ser incluída na forma de uma *applet* no *Supervisor Desktop*, ou através de uma aplicação separada (sem interface *web*).
- Especificação e implementação da aplicação de *eMarketing*.
- Apresentação de Estatísticas no *Supervisor Desktop* e *Administration Center*.
- Implementação da criação de *Querys* e Mapas de dados definidos pelo utilizador no *Supervisor Desktop*.

Numa fase posterior deverá ser efectuada uma reestruturação da camada transaccional, e possivelmente uma adaptação das restantes camadas, para a integração final na nova plataforma WIS2.

De salientar também que a Linha Branca é a base de desenvolvimento de futuros Contact Centers, de modo que a grande parte dos Contact Centers implementados nas instituições serão adaptações do sistema produzido. Por sua vez, alguns módulos podem ser posteriormente acrescentados à Linha Branca a partir de possíveis expansões desenvolvidas nas instituições.

Referências e Bibliografia

[1] Object Management Group. *OMG Unified Modelling Language Specification, v1.3.* 1999

[2] Ian Sommerville. *Software Engineering, 5th Edition.* 1995

[3] Companhia Portuguesa de Computadores – Tecnologias Avançadas. *WIS – Contact Center – Especificação Técnica.* 2002

[4] Sun Microsystems. *Java 2 Platform, Standard Edition* – http://java.sun.com/j2se/

[5] Sun Microsystems. *JavaBeans Component API* – http://java.sun.com/j2se/1.3/docs/guide/beans/

[6] Sun Microsystems. *JavaDoc Tool Home Page* – http://java.sun.com/j2se/javadoc/

[7] Netscape Communications Corp. *Core JavaScript Reference* – http://devedge.netscape.com/library/manuals/2000/javascript/1.5/reference/

[8] *Companhia Portuguesa de Computadores* – http://www.cpcis.pt/

ANEXO A: Glossário

API – *Application Programming Interface*. Uma *API* define funções que podem ser chamadas por qualquer aplicação que pretenda interagir com o sistema que as implementa.

Applet – Aplicação cujo *interface* está integrado numa página WEB.

Cache – Uma cache é um sistema de acesso rápido utilizado para o armazenamento de informação temporária.

Cookie – Mecanismo usado pelos servidores *web* para guardarem valores nos *web browsers* clientes. O *web browser* o valor num ficheiro de texto (*cookie*), sendo posteriormente acessível a partir do servidor.

CORBA – *Common Object Request Broker Architecture*. Arquitectura para o desenvolvimento de sistemas baseados em objectos heterogéneos distribuídos. "Heterogéneos" significa que os objectos podem ser desenvolvidos em qualquer linguagem e operar em ambientes computacionais distintos.

Debugging – Acto de depuração de código fonte, de modo a identificar e corrigir erros (*bugs*).

Firewall – Dispositivo de *Hardware* e/ou *Software* com o objectivo de filtrar e monitorizar o trafico de dados que entra e sai de uma rede. As *Firewalls* são utilizadas principalmente para proteger redes e máquinas de ataques exteriores, e para impedir que certos tipos de dados entrem ou saiam da rede interna.

Home Banking – Sistema automatizado para a execução de operações bancárias através de telefone, *internet*, etc. O utilizador de *Home Banking* tem disponível muitas das funcionalidades das caixas multibanco sem precisar de se deslocar de casa ou escritório.

HTML – *Hypertext Markup Language*. Linguagem que permite criar documentos de hipertexto. Os documentos criados podem conter texto, imagens, sons, vídeos, referências a outros documentos, etc. O HTML é a principal linguagem utilizada na criação de páginas *Web*.

Java Server Pages – Tecnologia que estende as funcionalidades de um servidor *web*, permitindo a geração dinâmica do código que será retornado ao cliente. As páginas JSP são constituídas por código HTML "embutido" em código *Java*. Geralmente, o código HTML define o interface gráfico de utilizador; o código *Java* permite o acesso a recursos do sistema e gera código HTML adicional.

JDBC – *Java Database Connectivity*. API para aplicações *Java* que oferece funções para interacção com Sistemas de Gestão de Bases de Dados. Existem funções para estabelecer conexões ao SGBD, enviar instruções SQL para pré-processamento e execução, etc.

JSP – Ver *Java Server Pages*.

Middleware – Aplicação ou conjunto de aplicações situado entre um cliente e uma base de dados.

Query – Representa um pedido de informação a uma Base de Dados.

Servidor Aplicacional – Aplicação que suporta os módulos da camada intermédia de uma arquitectura em três ou N camadas.

Servidor Web – Aplicação que permite disponibilizar páginas web numa rede de computadores.

SGBD – Sistema de Gestão de Bases de Dados. Aplicação que controla o acesso de outras aplicações aos dados e meta-dados de uma base de dados.

SOAP - *Simple Object Access Protocol*. Protocolo baseado em XML, para a troca de informação em ambientes distribuídos.

Software House – Empresa cuja área de negócio assenta sobre a produção e venda de *software*.

SQL – *Structured Query Language*. Linguagem standard utilizada para efectuar operações em Bases de Dados.

URL – *Uniform Resource Locator*. Corresponde ao endereço global de documentos na *internet*, por exemplo "`http://www.fe.up.pt/index.html`".

Web Browser – Aplicação que permite aceder a um servidor *Web* e visualizar a informação por ele disponibilizada.

XML – *Extensible Markup Language*. O XML é uma forma flexível de definir formatos de informação.

ANEXO B: Plataforma WIS

B.1 Introdução

Para responder aos novos desafios trazidos pela *Internet*, o Departamento de Projectos Especiais da CPC TA desenvolveu a plataforma WIS (*Web Information System*). Esta plataforma *middleware* foi criada para permitir um desenvolvimento mais rápido de aplicações de grande dimensão, garantindo segurança, performance e integração de serviços de *Internet* e *Intranet* tais como *Home Banking*, Contact Center, Correctoras *On-line*, *eCommerce* e Sistemas de Segurança.

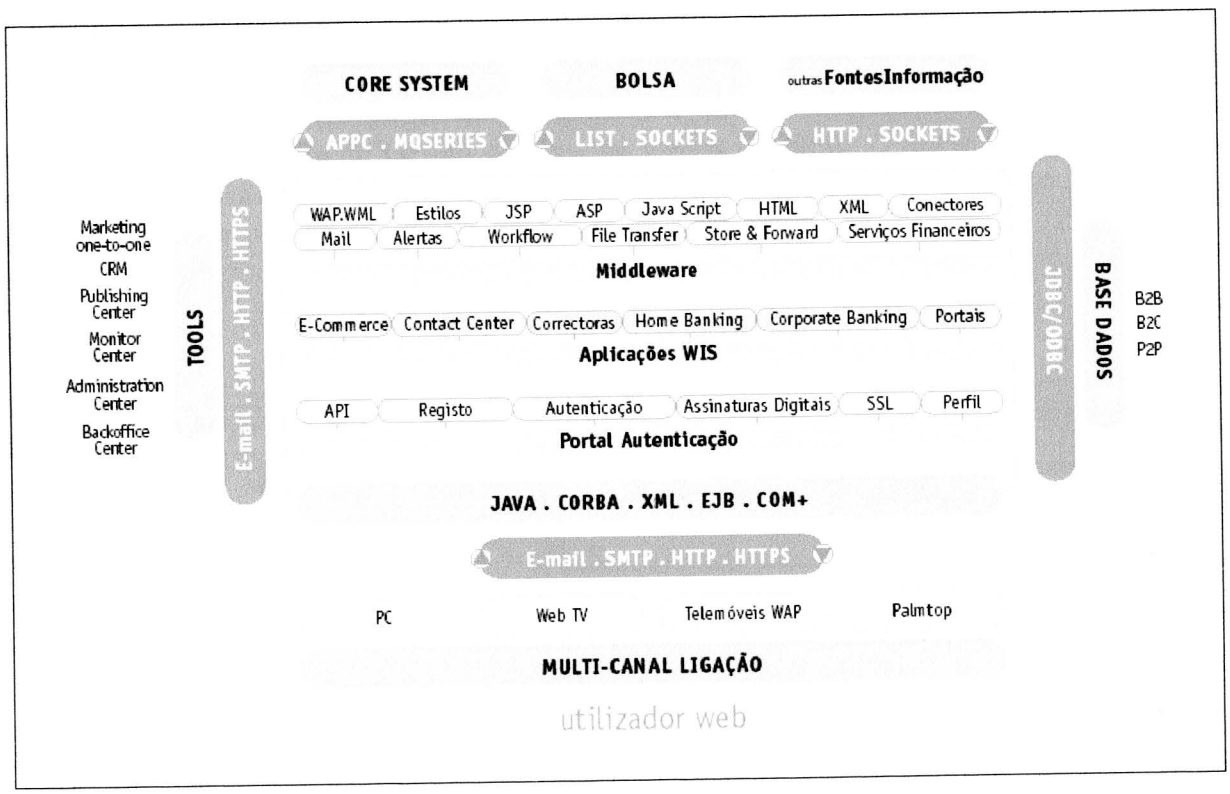


Figura B.1 – Aplicações da Plataforma WIS

Os clientes da CPC TA são na sua maioria da área financeira, dado que a segurança e integridade dos dados são elementos cruciais do seu negócio. Com este factor em conta, a WIS garante elevados níveis de segurança através de vários dispositivos: SSL (*Secure Sockets Layer*), Certificados Digitais, *Firewalls*, etc.

A WIS suporta elevados níveis de utilizadores, ou seja, é escalável e portanto apta a suportar mais carga sem degradar os níveis de serviço. Possui tolerância a falhas e balanceamento de carga entre os *browsers* e servidores *web*, e entre servidores *web* e servidores aplicacionais. Deste modo, o número de máquinas pode ser incrementado caso se detectem eventuais perdas de performance, garantindo assim a escalabilidade da solução por forma a evitar eventuais estrangulamentos que resultem da utilização massiva da plataforma.

Outro aspecto significativo é a capacidade de integração entre aplicações WIS. As aplicações podem utilizar serviços transaccionais comuns, permitindo uma fácil expansão das funcionalidades, diminuindo o custo de pacotes agregados de várias aplicações. A estrutura da plataforma WIS é flexível e adaptável às dimensões do negócio da instituição.

Existe ainda um elevado numero de *webtools* para monitorização, publicação de conteúdos, administração da plataforma, entre outros. A plataforma também gera *logs* detalhados com todos os pedidos e operações efectuados, e os eventos e excepções resultantes.

B.2 Arquitectura

As tecnologias base são *Java* e *Corba*, sendo este ultimo utilizando como infra-estrutura de comunicação e instanciação remota de objectos. Estas tecnologias têm a vantagem de tornar a plataforma independente do sistema operativo.

A plataforma WIS possui uma arquitectura de três camadas (*3-tier*): Apresentação, Transaccional e Sistema de Gestão de Base de Dados (SGBD) – *Figura B.2*. A camada de Apresentação está ainda dividida em duas sub-camadas lógicas: *Presentation Manager* e *Session Manager*.

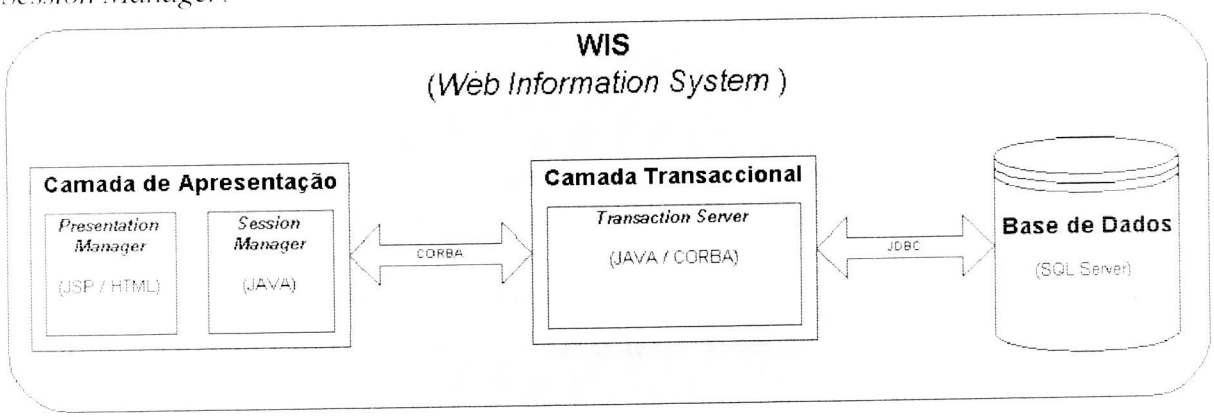


Figura B.2 – Arquitectura Base da Plataforma WIS

O modelo de camadas permite uma optimização de custos e tempos de desenvolvimento de novos canais aplicacionais, possibilitando a reutilização de serviços de aplicacionais e respectivos interfaces com outros sistemas. Esta separação garante ainda niveis de segurança mais elevados, podendo as camadas ser fisicamente separadas por *firewalls* ou outros dispositivos de segurança.

B.2.1 Sistema de Gestão de Base de Dados

A WIS suporta grande parte dos SGBDs, necessitando apenas de um *driver* actualizado para a ligação através de JDBC. Usualmente a WIS utiliza Bases de Dados *Microsoft SQL Server* ou *Oracle*.

Para além das Bases de Dados específicas para cada aplicação WIS, existem também Bases de Dados incluídas na WIS para o uso geral das aplicações. Entre outras, estão disponíveis Bases de Dados para publicação de conteúdos (*publishing*), gestão de contactos, *eMarketing*, etc. Existe também a Base de Dados base da WIS, que contém todos os dados relativos a utilizadores, aplicações, menus, etc.

Habitualmente as instituições já possuem Bases de Dados e outros sistemas remotos, que necessitam de interagir com a informação nas Bases de Dados WIS. Dependendo das plataformas existentes na instituição, a replicação e sincronismo dos dados pode ser feito directamente através de SGBDs compatíveis em ambos os lados, ou através de outros motores de dados que traduzem a informação dos sistemas remotos da instituição e os enviam para a camada transaccional da WIS.

B.2.2 Camada Transaccional

O *Transaction Server* é o núcleo da plataforma. Assente em servidores aplicacionais, é constituído por um motor transaccional que serve de infra-estrutura para o desenvolvimento de serviços. Um serviço pode ser definido como uma acção que implementa regras de negócio.

Para o desenvolvimento de serviços o motor transaccional oferece uma série de funcionalidades, nomeadamente:

- Acesso simples a repositórios de informação, com *pool* de ligação a Bases de Dados e conectores a sistemas remotos;
- Grande abstracção em relação aos mecanismos de comunicação e segurança;
- Mecanismos de tratamento de erros;
- Mecanismo de criação/gestão de *logs*;
- Elevado nível de configuração através da utilização de ficheiros de propriedades.

O uso de JDBC disponibiliza métodos simples para o estabelecimento de conexões a Bases de Dados, execução de comandos SQL e processamento de resultados obtidos através de *querys* SQL. Contudo o JDBC não permite um número ilimitado de conexões, e o tempo de espera para abrir ou fechar uma conexão é significativo. Para evitar estas limitações, foi criada uma *pool* de conexões, configurável por número de conexões simultâneas. Quando um serviço tenta aceder a uma Base de Dados pela primeira vez, é aberta uma conexão por JDBC na *pool*. Quando o serviço deixa de necessitar da conexão, esta não é fechada, mantendo-se disponível para outros serviços que eventualmente acedam à mesma conexão. Deste modo, reduz-se significativamente o tempo gasto na criação e eliminação de conexões por JDBC. A *pool* tem um número limitado de conexões abertas, e quando este limite é atingido, a conexão que não é usada à mais tempo é fechada. As conexões também são fechadas após um período de tempo sem utilização.

O motor transaccional é altamente configurável através de ficheiros de propriedades. Estes ficheiros permitem definir inúmeros parâmetros, como as configurações dos servidores *Corba* e *firewall*, parâmetros de autenticação, sistema de *logging*, definição das Bases de Dados disponíveis, *pools*, etc. Para cada aplicação WIS existe também um ficheiro de configuração com todos os serviços disponíveis para a aplicação, indicando o método que implementa o serviço e o nível de autenticação necessário para um utilizador executar o serviço.

No uso geral, um serviço é implementado por um método estático *Java* que executa as operações necessárias. Estas operações incluem a obtenção, inserção, actualização ou remoção de dados no SGBD da WIS ou em sistemas de informação remotos. Habitualmente, os dados de entrada e/ou saída de cada método são colocados em estruturas definidas num

ficheiro IDL, de modo a serem transportados entre o *Transaction Server* e a camada de Apresentação.

Os IDLs são ficheiros *Corba* usados para definir estruturas de dados. Ao compilar um ficheiro IDL (através do comando *idl2java*) são geradas várias classes *Java*. Cada estrutura é traduzida para uma classe *Java* que pode ser instanciada livremente, de modo utilizar-se objectos equivalentes à estrutura criada. As restantes classes geradas são usadas pelo *Corba* para a transferência dos objectos entre as camadas.

Cada serviço recebe ainda como argumento uma instância da classe *ServicoSessao*. Cada vez que um utilizador entra numa aplicação é criado um objecto de sessão, e através desta classe é possível aceder a vários atributos da sessão, aplicação e utilizador. Entre outras propriedades, o *ServicoSessao* pode ser usado para obter conexões à *pool* de Base de Dados, atributos do utilizador actual, propriedades da aplicação, etc.

B.2.3 Camada de Apresentação – Session Manager

O *Session Manager* serve sobretudo como intermediário entre o *Transaction Server* e o *Presentation Manager*. É constituído por classes *Java* que ao serem instanciadas no *Presentation Manager* invocam serviços da camada Transaccional. Em certos casos, os dados obtidos do motor transaccional podem receber uma pré-formatação nesta camada, antes de serem retornados para o *Presentation Manager*.

Da mesma forma que as ligações entre a camada Transaccional e o SGBD são geridas através de uma *pool*, as conexões via *Corba* entre o *Session Manager* e o *Transaction Server* são também obtidas através de uma *pool* de conexões, de modo a diminuir os tempos de espera para a abertura e fecho de conexões.

As dados enviados ou recebidos do *Transaction Server* devem estar encapsulados em estruturas definidas em IDLs na camada Transaccional. Na maioria dos casos, o *Session Manager* recebe estas estruturas já construídas previamente no *Presentation Manager* e reenvia-as para o motor transaccional. Para além das eventuais estruturas de entrada, cada classe desta camada recebe ainda outros parâmetros enviados do *Presentation Manager*:

- *Flag de simulação* – Indica se o *Session Manager* deve invocar o serviço da camada transaccional, ou se deve simular o serviço localmente, sem utilizar o motor transaccional.
- *Objecto da ligação de Sessão* – Contém os dados de sessão da *pool* de ligação ao *Transaction Server*.
- *Objecto Trace* – Este objecto contém métodos para escrever nos *logs* criados. É usado principalmente para *debugging* e para colocar nos *logs* informação adicional sobre os eventos a decorrer no sistema.
- *ID da Sessão* – Identificador da sessão do utilizador actual. Esta variável só é necessária para invocar serviços que necessitem da autenticação prévia do utilizador.

O *Session Manager* reside em servidores *web*, juntamente com o *Presentation Manager*. Contudo os servidores *web* têm suportar o tratamento de classes *Java*, sendo normalmente usados *web servers* como o *JRun* ou o *TomCat*.

B.2.4 Camada de Apresentação – Presentation Manager

A plataforma WIS tem como camada superior o *Presentation Manager* (PM). Esta camada implementa os interfaces com o utilizador, de forma a dar respostas via HTTP/HTTPS. O PM é implementado utilizando as tecnologias mais usuais para a criação de páginas *web* estáticas e dinâmicas: HTML, *JavaScript*, CSS, JSP, etc; Sendo que normalmente o mais utilizado são *scriplets* JSP que geram o código HTML/*JavaScript* para a página pretendida. A utilização de JSP tem como grande vantagem a possibilidade de inclusão de código e classes *Java*. Assim sendo, as *scriplets* JSP criam instâncias de classes do *Session Manager* de modo a obter ou enviar os dados necessários.

O interface com utilizador, é característico de cada aplicação, não existido normas ou restrições da plataforma quanto à sua implementação. O único aspecto a ter em conta é que as tecnologias utilizadas devem permitir processamento de classes *Java* de modo a estabelecer a interligação ao *Session Manager*.

B.3 Evolução para a WIS2

De modo a dar resposta às novas exigências tecnológicas está actualmente em desenvolvimento a WIS2. Na versão *Java* desta nova plataforma, a mudança mais significativa será na camada transaccional. Apesar da estrutura definitiva ainda estar em estudo, tudo indica que se irá abandonar o *Corba*, devendo a comunicação entre as camadas de apresentação e transaccional passar a ser efectuada através de mensagens XML. Prevê-se que este sistema seja implementado através de SOAP, sendo os serviços disponibilizados por um servidor aplicacional específico, provavelmente o *WebSphere*. Tudo isto implica uma radical remodelação do motor transaccional da plataforma.

Na camada de apresentação também se verificarão algumas mudanças. A interacção entre as sub-camadas do *Presentation Manager* e *Session Manager* deverão seguir uma filosofia de *JavaBeans*, sendo as classes implementadas no *Session Manager*, e incluídas nas páginas JSP do *Presentation Manager*. O uso de *beans* permite uma representação mais componencial da informação, facilitando a sua reutilização.

O sistema de invocação de serviços terá também que ser significativamente alterado de modo a incluir o uso de mensagens XML através de SOAP.

ANEXO C: Nova Linha Branca Vs Antiga Linha Branca

Este anexo pretende demonstrar as principais alterações e melhorias efectuadas no Contact Center – Linha Branca, tanto a nível funcional como não funcional. Serão referidos apenas os aspectos mais significantes, mencionando as vantagens e desvantagens de cada caso.

Alguns dos termos e conceitos referidos neste anexo têm um teor técnico, e a sua compreensão pode implicar alguma familiarização com a plataforma WIS, e com a linguagem *Java*, incluindo os conceitos gerais de JSPs e *JavaBeans*.

C.1 Camada Transaccional

A nível das tecnologias e metodologias utilizadas, não ocorreram grandes alterações no que diz respeito ao *Transaction Server* da nova Linha Branca. Contudo algumas funcionalidades gerais exigem algum tratamento ao nível transaccional. Nomeadamente a utilização de ordenação e paginação de listagens, é necessário obter os dados da Base de Dados já previamente ordenados, e apenas retornar para a camada de apresentação os elementos respectivos ao número e tamanho da página pretendida.

Para implementar esta funcionalidade é necessário a produção de algum código adicional aos serviços e estruturas transaccionais. A pré-ordenação e paginação das listagens no nível transaccional permite uma diminuição do trafego entre o *Transaction Server* e o *Session Manager*, transferindo apenas os registos pertencentes à página actual.

Na versão anterior da Linha Branca estas funcionalidades não existiam, verificando-se uma menor complexidade na implementação de serviços para obtenção de dados.

C.2 Session Manager

A camada lógica de sessão foi significativamente alterada com a passagem da plataforma WIS original para a WIS melhorada. Nomeadamente, a utilização de *JavaBeans* veio modificar as metodologias utilizadas para a implementação desta camada.

Na versão original da Linha Branca, as classes gerais do *Session Manager* (SM) eram constituídas por um construtor que recebia todos os parâmetros necessários para a invocação do serviço, incluindo uma *flag* de simulação, uma instância da ligação da sessão (`LigacaoSessaoTS`), um objecto para permitir a escrita de informação para os *logs* (`Trace`), e os dados de entrada eventualmente requisitados pelo serviço. Todos estes parâmetros pressupõem a sua criação e inicialização no nível superior, ou seja no *Presentation Manager*. Os dados eventualmente retornados pelo serviço são guardados variáveis privadas das classes do SM, sendo acessíveis através de métodos `get` respectivos.

Na nova Linha Branca, com a utilização de *JavaBeans*, as classes *Java* seguem uma estrutura mais standard, em que todos os dados são colocados em variáveis privadas, sendo o interface com a camada superior disponibilizado através de métodos `set` e `get`. Para efectuar a invocação de serviços todos os *beans* devem estender a classe `ServicoPS`, herdando métodos e campos utilizados para o efeito. Entre os objectos herdados destaca-se uma instância das propriedades e configurações da aplicação actual, assim como um objecto `Trace`

para a eventual escrita em *logs* e a ligação de sessão ao *Transaction Server* (`LigacaoSessaoTS`). Deste modo estas características são transparentes ao *Presentation Manager*.

A invocação de serviços transaccionais é efectuada por um método colocado no *JavaBean*, similar ao construtor utilizado no antigo Contact Center. Contudo, este método não deve ser chamado directamente, mas através da super classe `ServicoPS`, de modo a ser invocado com a instância da `LigacaoSessaoTS` correspondente à sessão WIS do utilizador actual.

Comparando ambas as metodologias, a utilizada no novo sistema é claramente melhor estruturada, seguindo normas standard, permitindo uma melhor transparência ao nível dos *Session* e *Presentation Managers*. Contudo, a invocação indirecta dos métodos para a invocação dos serviços veio acrescentar alguma complexidade ao desenvolvimento do sistema, sendo a sua compreensão menos intuitiva.

C.2.1 Exemplo de Código Fonte da Antiga Linha Branca

O seguinte exemplo refere-se a uma classe típica no *Session Manager* da versão original da Linha Branca. A seguinte classe invoca um serviço para a obtenção de todos os itens na agenda de um agente especificado:

```
package cpcis.wis.jsm.contactcentre.agente;

import cpcis.wis.jsm.*;
import cpcis.wis.util.*;
import cpcis.wis.jts.util.*;
import cpcis.wis.jts.modPKI.*;
import cpcis.wis.jts.modComum.*;
import cpcis.wis.jts.contactCentre.agenda.params.*;
import cpcis.wis.jts.servicos.contactCentre.comum.*;

public class ListaAgendasAgente{

  private OutDetalheAgenda res;

  public ListaAgendasAgente(short simula,
                LigacaoSessaoTS sessaoTS,
                Trace trace,
                String idSessao,
                String idAgente) throws Exception {

     trace.trace(12," Vai ligar ao ts");

    Object[] servicos = new Object[1];
    servicos[0] = idAgente;

    //Vai o JTS
    trace.trace(3," Vai  efectuar o invoca");

    res=(OutDetalheAgenda)InvocaTS.invoca(sessaoTS.orb,
                    sessaoTS.sessaoPKI,
                    "cpcis.wis.jts.contactCentre.agenda.Agenda.getAgendasAgente",
                    servicos,
                    OutDetalheAgenda.class,
                    trace,
                    idSessao);

    trace.trace(3," Passou o Invoca");
  }
```

```
  public OutDetalheAgenda getAgendasAgente(){
    return res;
  }

}
```

C.2.2 Exemplo de Código Fonte da Nova Linha Branca

Seguidamente é mostrado um *JavaBean* implementado para a listagem de agendas no novo Contact Center desenvolvido. Em termos funcionais, em comparação com a classe anterior, este *bean* contém dados adicionais para o suporte para pesquisas e para paginação e ordenação da listagem retornada.

```
package pt.cpcta.wis.jsm.comum.contactos;

import pt.cpcta.wis.jsm.*;
import pt.cpcta.wis.util.*;
import pt.cpcta.wis.jts.modPKI.*;
import pt.cpcta.wis.jts.modComum.*;
import pt.cpcta.wis.jts.servicos.comum.contactos.listaAgenda.*;
import pt.cpcta.wis.jts.servicos.comum.contactos.trataContactos.*;

/**
 * Sistema: Contact Center - Linha Branca <br>
 * Aplicação: Portal <br>
 * Descrição: Bean para invocar o serviço para listar agendas. <br>
 * @author Rúben Pereira
 * @version 1.0.15.31
 */
public class ListaAgenda extends ServicoPS{

  /** Estrutura onde são guardados os dados da paginação e o <i>array</i> com as agendas */
  private OutListaAgenda outAgendas = new OutListaAgenda();

  /** Estrutura com guardados os dados de entrada (pesquisa e paginação) para obter a
   * lista de agendas */
  private InAgenda inAgenda = new InAgenda();

  /** Variável de Sessão */
  private String idSessao;


  /** Atribui um valor ao idSessao.
   * @param t Valor da Sessão.
   */
  public void setIdSessao(String t){
    idSessao=t;
  }
  public String getIdSessao(){
    return idSessao;
  }


  /** Atribui valores às opções para obter a lista de agendas.
   * @param in Estrutura com os dados de entrada.
   */
  public void setInAgenda(InAgenda in) {
    inAgenda = in;
  }
  public InAgenda getInAgenda() {
    return inAgenda;
  }

  /** Atribui ao utilizador para quem é a agenda.
   * @param destino Agente dono da agenda.
   */
  public void setDestinatario(String destino) {
```

```
    inAgenda.destino = destino;
  }
  public String getDestinatario() {
    return inAgenda.destino;
  }

  /** ... */

  /** Atribui um valor à página actual.
   * @param pagina    Página pretendida.
   */
  public void setPagina(int pagina) {
    inAgenda.page = pagina;
  }
  /** Devolve a página actual.
   * @return Página actual.
   */
  public int getPagina() {
    return outAgendas.pagina;
  }

  /** Atribui um valor ao tamanho de cada página.
   * @param tamPagina Tamanho de cada página.
   */
  public void setTamanhoPagina(int tamPagina) {
    inAgenda.tamPagina = tamPagina;
  }
  /** Devolve o tamanho de cada página.
   * @return tamanho de cada página.
   */
  public int getTamanhoPagina() {
    return inAgenda.tamPagina;
  }

  /** Atribui um valor à ordenação a efectuar na listagem.
   * @param ordem     Ordenação da listagem. Deverá ter o formato:
   * <code>"campo1, campo2, ..., campoN"</code>
   */
  public void setOrderBy(String ordem){
    inAgenda.orderBy = ordem;
  }
  public String getOrderBy() {
    return inAgenda.orderBy;
  }


  /**
   * Devolve a lista de agendas.
   * @return <i>Array</i> com todas as agendas.
   */
  public AgendaInfo[] getAgendas(){
    return outAgendas.elementos;
  }
  public void setAgendas(AgendaInfo[] agendas){
    outAgendas.elementos = agendas;
  }

  /** Devolve o número de páginas retornadas.
   * @return Número de páginas retornadas.
   */
  public int getNumPaginas(){
    return outAgendas.numPaginas;
  }
  public void setNumPaginas(int n){
    outAgendas.numPaginas = n;
  }
```

```
    /** Inicializa as estruturas de entrada e saída de dados. */
    public ListaAgenda() {
      outAgendas = (OutListaAgenda)Formatador.formataNulos(new OutListaAgenda());
      inAgenda = (InAgenda)Formatador.formataNulos(new InAgenda());
    }


    /**
     * Faz a invocação à camada transaccional para carregar os dados da BD.
     */
    public void getListaAgendaTS(LigacaoSessaoTS _lsts) throws WISException {
      trace.debug("ListaAgenda.getListaAgendaTS(): ENTROU.");

      if (!propriedadesAplicacao.getConfig().getPropertyAsBoolean("simulacao",
                                                                  false)) {

        trace.debug(" Vai ligar ao ts");
        Object[] servicos = new Object[1];
        servicos[0] = inAgenda;

        trace.debug(" Vai fazer o invoca");
        outAgendas = (OutListaAgenda) InvocaTS.invoca(_lsts.orb,
                        _lsts.sessaoPKI,
              "pt.sapra.wis.jts.servicos.comum.contactos.listaAgenda.ListaAgenda.ListaAgenda",
                        servicos,
                        OutListaAgenda.class,
                        idSessao);

        trace.debug(" terminou");
      }
    }

    /**
     * Este método deve ser chamado quando se quer fazer um update no transactis
     * Usa em conjunto com os comandos: inserir, alterar, remover.
     * @exception WISException esta excepcao é lançada pela plataforma WIS.
     */
    public void update() throws WISException {
      invocaMetodoLSTS();
    }

}
```

C.3 *Presentation Manager*

A migração do *Session Manager* para *JavaBeans* provocou também repercussões na implementação das páginas do *Presentation Manager*.

Na antiga Linha Branca, as páginas JSP que implementavam o interface com o utilizador, tinham uma estruturação limitada, contendo uma quantidade considerável de código repetitivo, não relacionado directamente com a apresentação de dados ao utilizador. Este código *Java* inclui a inicialização de variáveis gerais guardadas em ficheiros de propriedades e *cookies*, e também várias operações necessárias para a invocação de serviços por intermédio da classe respectiva do *Session Manager*. Estas operações abrangem a obtenção do objecto `Trace` e duma instância da *pool* de conexões para o *Transaction Server*. Posteriormente é obtido uma instância da `LigacaoSessaoTS` e é criada a classe do SM, enviando uma *flag* de simulação, a `LigacaoSessaoTS` e o `Trace` como os parâmetros.

Relativamente à estruturação da geração do código HTML que compõe o interface, existe um ficheiro JSP que ao ser incluído produz um cabeçalho genérico, comum à maioria das páginas da Linha Branca antiga. O tratamento de erros e excepções é efectuado na parte

final de cada página JSP. Mais à frente nesta secção é demonstrado um exemplo de uma página JSP típica da primeira versão da Linha Branca.

No novo Contact Center desenvolvido, o procedimento para invocação de serviços referido na secção anterior veio reduzir as operações efectuadas no *Presentation Manager*, originando também uma melhor definição e independência das camadas de Apresentação/Sessão. Contudo, a invocação de um serviço através de um *JavaBean* não é um procedimento imediato. Após a inicialização do *bean* no JSP, é necessário atribuir valores às propriedades `nomeAplicacao` e `metodo` (da super classe `ServicoPS`). O `nomeAplicacao` é necessário para a inicialização das propriedades da aplicação, possivelmente utilizadas no *bean*. O `metodo` deve indicar o nome do método da classe do SM que invoca o serviço, para que este seja posteriormente chamado pela super classe. Para efectivar a invocação do serviço, na página JSP deve ser chamado o método `update` do *JavaBean*, que irá provocar a criação da `LigacaoSessaoTS` e chamará o método do *bean* com o nome indicado anteriormente. Este procedimento é ilustrado no seguinte diagrama de sequenciação.

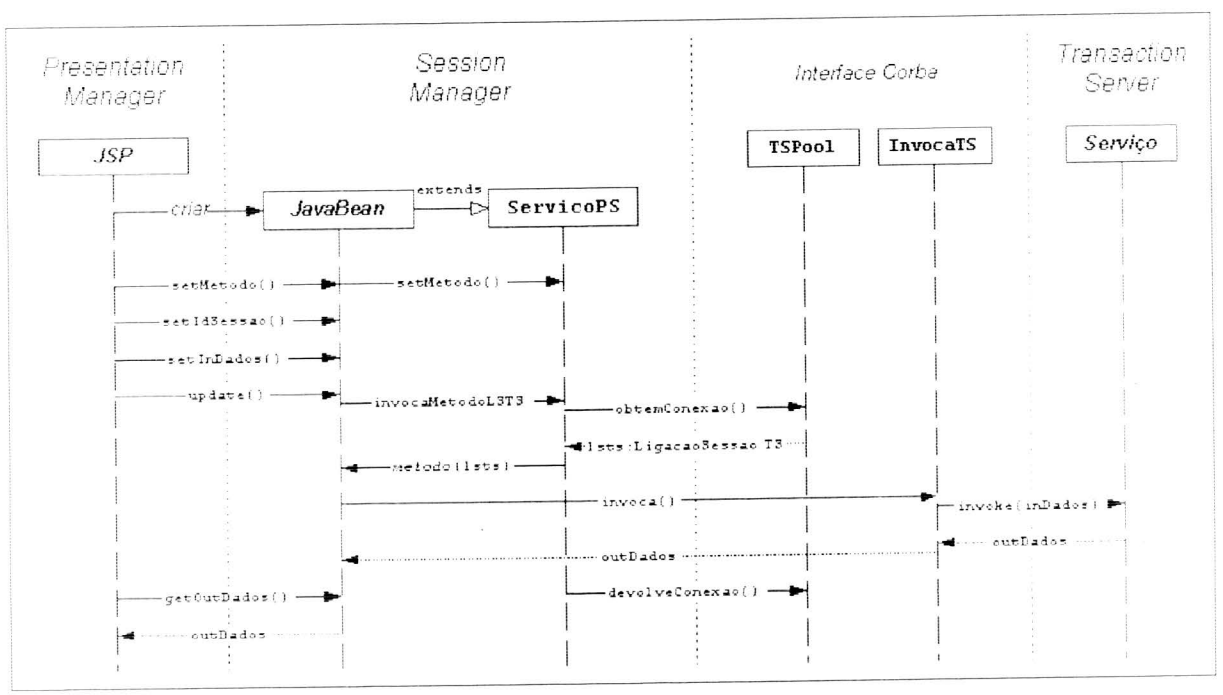


Figura C.1 – Diagrama de Sequenciação: Invocação de um Serviço

Para obter o acesso a uma grande quantidade de variáveis e parâmetros, guardados em ficheiros de propriedades e *cookies*, na nova Linha Branca todas as páginas JSP devem incluir o ficheiro `inicioPagina.jsp`. Este JSP define também uma página apropriada para o tratamento de erros. A inclusão deste ficheiro permite diminuir consideravelmente a quantidade código necessário para a construção de uma página JSP, sendo todas as variáveis gerais acessíveis de forma transparente para o criador das páginas *web*. A utilização de *JavaBeans* permite também a redução da necessidade de obter e tratar variáveis recebidas por *request*, podendo estas ser enviadas directamente para o *bean* e receber tratamento na camada inferior.

Existem também várias APIs para facilitar a criação e formatação de páginas utilizando o *design* geral da Linha Branca. A classe `UtilLinhaBranca`, disponibiliza vários métodos para a geração do cabeçalho comum à maior parte das páginas, conversão de parâmetros do URL em formulários, e geração do *link* para ordenação de listagens. Contudo, para o bom funcionamento da ordenação e paginação de listagens é necessário efectuar alguns tratamentos de parâmetros recebidos por *request*, e deverão ser criadas funções *JavaScript* com os nomes `ordena` e `novo`, para a actualização da ordenação e paginação, respectivamente. A geração dos *links* para as páginas seguintes ou anteriores das listagens são também gerados automaticamente com a inclusão do ficheiro `Paginacao.jsp` no local apropriado. Para auxiliar a navegação entre páginas, existe a API `voltarUtil.jsp` que disponibiliza *links* com o estado da página anterior, página actual, facilitar a navegação para as páginas seguintes ou para voltar às anteriores. A passagem entre páginas é também efectuada através de uma página intermédia, indicando uma mensagem para o que utilizador espere até que a próxima página seja carregada.

A utilização de *JavaBeans* e as APIs referidas trazem imensas vantagens para a simplificação e estruturação do código nos ficheiros JSPs, permitindo que mesmo pessoas com pouca experiência em programação possam facilmente criar páginas *web* obtendo uma boa representação e acessibilidade de dados através de *beans* apropriados. Por outro lado, o acréscimo de funções utilitárias pode também complicar a compreensão do código produzido, dada a existência de diversos dados originados de forma indirecta. Um número demasiado elevado de APIs pode também prejudicar a utilização das mesmas, sendo necessário manter os programadores informados de todos os métodos disponíveis.

C.3.1 Exemplo de Código Fonte da Antiga Linha Branca

Seguidamente é mostrado o código fonte de um ficheiro JSP para a listagem de *links* na aplicação *Agent Desktop* na primeira versão do Contact Center Linha Branca.

```
< @ page import = "cpcis.wis.util.*, cpcis.wis.jsm.*"%>
< @ page import = "cpcis.wis.jsm.contactcentre.agente.*"%>
<
// Le o Dir. virtual para saber a rede em que esta
    String vVirtualDir = LeDirectorioVirtual.get(request.getRequestURI());
    Trace vTrace = (Trace) application.getAttribute(vVirtualDir+".traceAplicacao");

// Le as propriedades normalmente
  String vEstilo = PropriedadesWIS.get(application, vVirtualDir,
                                                    "stylesheet.nome");
  String vSimula = PropriedadesWIS.get(application, vVirtualDir, "simulacao");

  TSPool vPool = null;
  LigacaoSessaoTS vSessaoTS = null;
  int idServidorCliente = -1;
  String idSessaoCliente = "0";
  try {

      //Ler IDSessao
    idSessaoCliente = TrataCookies.getValorCookie(request.getCookies(),
                                                  "idSessaoCliente");
    if (idSessaoCliente == null || idSessaoCliente.equals("")) {
       response.sendRedirect("expira.jsp");
       return;
    }
```

```
    //Ligacao ao TS
    vTrace.trace(12,"vai obter conexao");
    vPool = (TSPool) application.getAttribute(vVirtualDir+".poolTS");
    if (vPool != null)
      vSessaoTS = (LigacaoSessaoTS) vPool.obtemConexao(idServidorCliente);

   //OBJECTO para retornar a informação dos GRUPOS DE LINKS
   SPGrupoLinks links = new SPGrupoLinks(vSimula, vSessaoTS, vTrace);
   SPI_GrupoLinks[] grupoLinks = links.getGrupoLinks();
   String grupo="";%>
<html>
<head>
<title></title>
<LINK REL="STYLESHEET" TYPE="text/css" HREF="<%=vEstilo%>">
</head>
<script>
function abreJanela(link){
  window.open(link,'','toolbar=yes,scrollbars=yes,width=900, height=600,
              left=40, top=60');
}
</script>
<% String tit="Links"; %>
<body class="FUNDO">
<%@ include file="titulo.jsp"%>

<table border=0 align=center width=570>

  <%if (grupoLinks!=null){%>
  <%for (int i=0;i<grupoLinks.length;i++){
      if (!grupo.equals(grupoLinks[i].id_grupo)){
          grupo=grupoLinks[i].id_grupo;%>
   <tr><td> </td></tr>
  <tr>
     <td class="TEXTOBG" align=center
colspan="3"> <b><%=grupoLinks[i].nome_grupo%></b></td>
  </tr>
   <%}%>
  <tr>
    <td colspan="3" align="center">     
      <a class=OPCAO href=# onclick="abreJanela('<%=grupoLinks[i].link%>')">
        <%=grupoLinks[i].nome%></a>
    </td>
  </tr>
  <%}%>
 <%}else{%>
  <tr>
     <td align=center class=COMMENT><b> Não existem Links!</b></td>
  </tr>
 <%}%>
</table>

</body>
</html>
<%
    vTrace.saiu("CONTACT CENTRE LISTA LINKS");
  }
  //se ocorrer erro
  catch (Throwable excepcao){
    String vMsgDefault = PropriedadesWIS.get(application, vVirtualDir,
                                             "listaLinks.jsp");
    String vLinkErro = TrataExcepcao.getLinkDestino(vMsgDefault, excepcao, vTrace,
                       "listaLinks", 0, "listaLinks.jsp", "consultaintegrada.jsp");

    vTrace.saiu("vMsgDefault: "+vMsgDefault);
    vTrace.saiu("vLinkErro: "+vLinkErro);
    response.sendRedirect(vLinkErro);
    return;
  }
  finally {
    vTrace.trace(3,"Entrou no finally");
```

```
    if (vPool !=null){
      vPool.devolveConexao(vSessaoTS);
    }
  }
%>
```

C.3.2 Exemplo de Código Fonte da Nova Linha Branca

O próximo extracto de código é funcionalmente muito semelhante ao anterior, mas estruturado seguindo as novas normas e metodologias do *Presentation Manager*. De notar que nesta versão existe um serviço para obter os *links* e um outro para os grupos de *links*. Na versão existia apenas um serviço para as duas acções.

```
<%@ page import = "pt.cpcta.wis.jts.servicos.comum.publishing.trataLinks.*" %>
<%@ page import = "pt.cpcta.wis.jts.servicos.comum.publishing.listaGruposLinks.*"%>
<%
/**
 * Projecto: Contact Center - Linha Branca <br>
 * Aplicacão: Agente <br>
 * Descricão: Lista de Links. <br>
 * Data: 26-06-2002<br>
 * Autor: Rúben Pereira<br>
 */
%>
<%@ include file="inicioPagina.jsp" %>

<jsp:useBean id="listaLinks" scope="page"
                    class="pt.cpcta.wis.jsm.comum.publishing.ListaLinks" />
<jsp:setProperty name="listaLinks" property="nomeAplicacao"
                    value="<%=nomeAplicacao%>"/>
<jsp:setProperty name="listaLinks" property="metodo" value="getListaLinksTS"/>
<jsp:setProperty name="listaLinks" property="idSessao"
                    value="<%=idSessaoAgente%>"/>
<jsp:setProperty name="listaLinks" property="idAplicacao"
                    value="<%=idAplicacao%>"/>

<jsp:useBean id="ListaGruposLinks" scope="page"
                    class="pt.cpcta.wis.jsm.comum.publishing.ListaGruposLinks"/>
<jsp:setProperty name="ListaGruposLinks" property="nomeAplicacao"
                    value="<%=nomeAplicacao%>"/>
<jsp:setProperty name="ListaGruposLinks" property="metodo"
                    value="getListaGruposLinksTS"/>
<jsp:setProperty name="ListaGruposLinks" property="idSessao"
                    value="<%=idSessaoAgente%>"/>
<%
InfoListaLink[] listaDeLinks = new InfoListaLink[0];
Elemento[] grupos = new Elemento[0];

listaLinks.update();
listaDeLinks = listaLinks.getListaLinks();

ListaGruposLinks.update();
grupos=ListaGruposLinks.getListaGruposLinks();
%>

<html>
<head>
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="0">
<link rel="stylesheet" href='<%=vEstilo%>' type="text/css">

</head>
<body >
<%=UtilLinhaBranca.geraCabecalho("Links")%>
<br>
<table align="center" width="60%" cellpadding="3">
```

```
<%
boolean existemLinks = false;
for(int j=0;j<grupos.length;j++){

      boolean grupoTemLinks = false;
      for (int i=0; i<listaDeLinks.length; i++)
        if (listaDeLinks[i].assunto.equals(grupos[j].descricao_grupo))
            grupoTemLinks = true;

      if (grupoTemLinks) {
        existemLinks = true;
%>

<tr><td height="8" colspan="2"></td></tr>
<tr>
  <td class="CABECALHO2" width="20%"><%=grupos[j].descricao_grupo%></td>
  <td class="CABECALHO1" width="80%"> </td>
</tr>
<%
        for (int i=0; i<listaDeLinks.length; i++)
            if (listaDeLinks[i].assunto.equals(grupos[j].descricao_grupo)){
%>
<tr><td height="3" colspan="2"></td></tr>
<tr><td colspan="2">
  <a class="LINK" href="<%= listaDeLinks[i].link %>" target="_blank">
    <img src="imagens/seta.gif" border=0> <%= listaDeLinks[i].nome %></a>
</td></tr>
<%          }      // fim do for (links)

      }      // fim do if (grupoTemLinks)

}      // fim do for (grupos)

if (!existemLinks) { %>
      <tr><td class="TEXTO3" align="center">
          <br><br><br><br><i>Não Existem Links</i>
      </td></tr>
<% } %>
</table>

</body>
</html>
```

C.4 Interface com o Utilizador

Para o novo Contact Center o *design* foi remodelado de modo a dar um novo aspecto ao sistema e melhorar o nível de usabilidade para os utilizadores. De seguida são mostrados alguns exemplos de ecrãs de funcionalidades equivalentes para cada uma das versões da Linha Branca.



Figura C.2 – Antiga Linha Branca: *Info Desk*

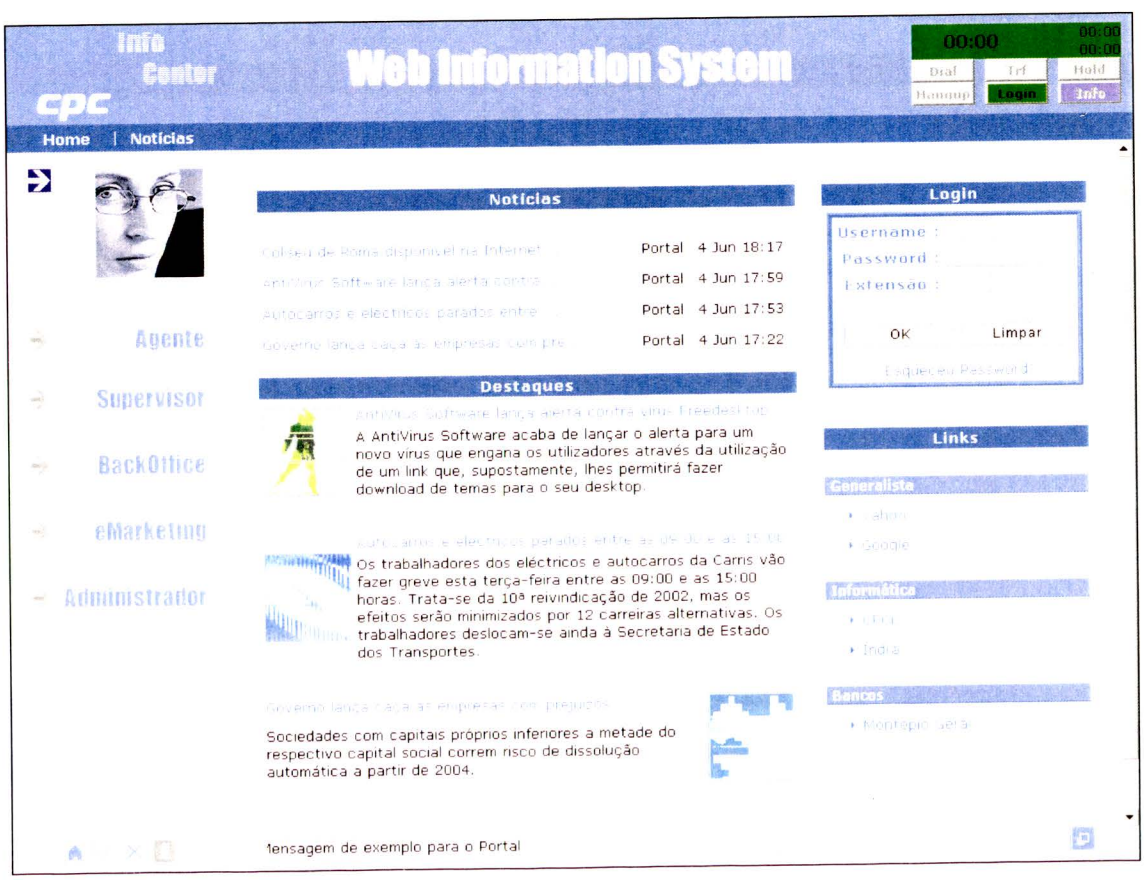


Figura C.3 – Nova Linha Branca: *Info Desk*

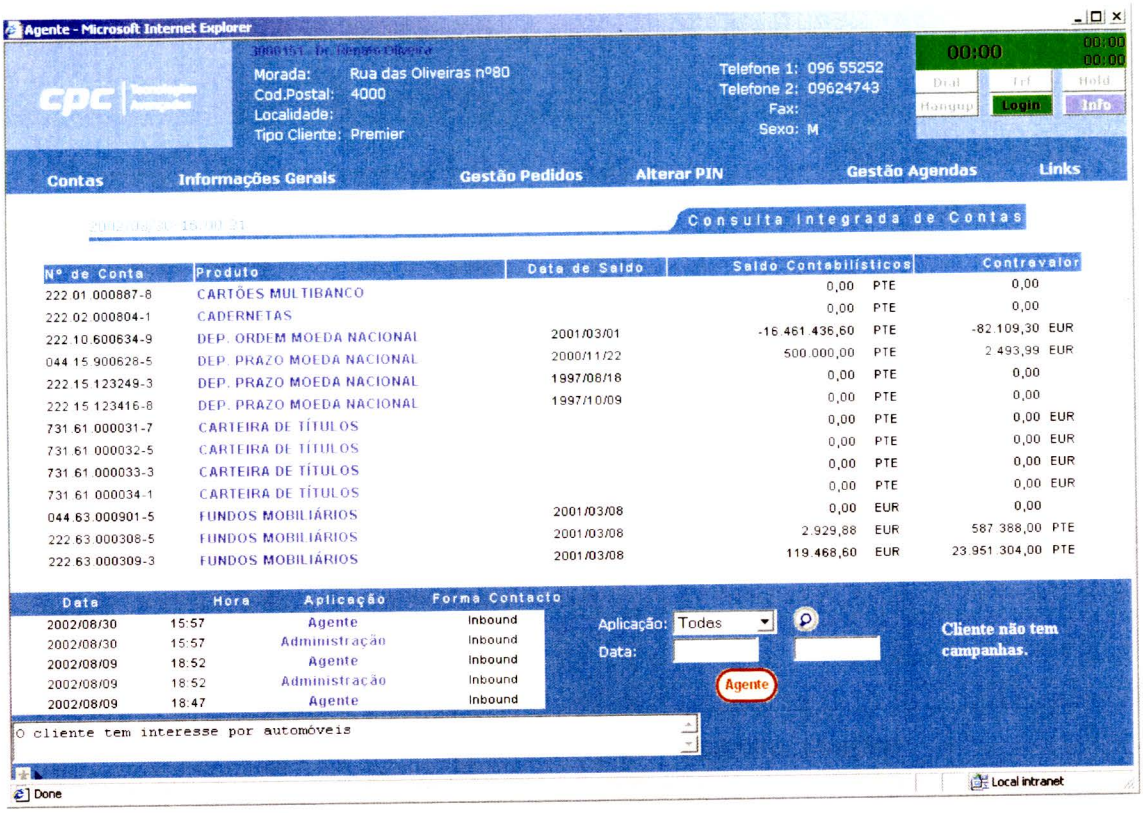


Figura C.4 – Antiga Linha Branca: *Agent Desktop*

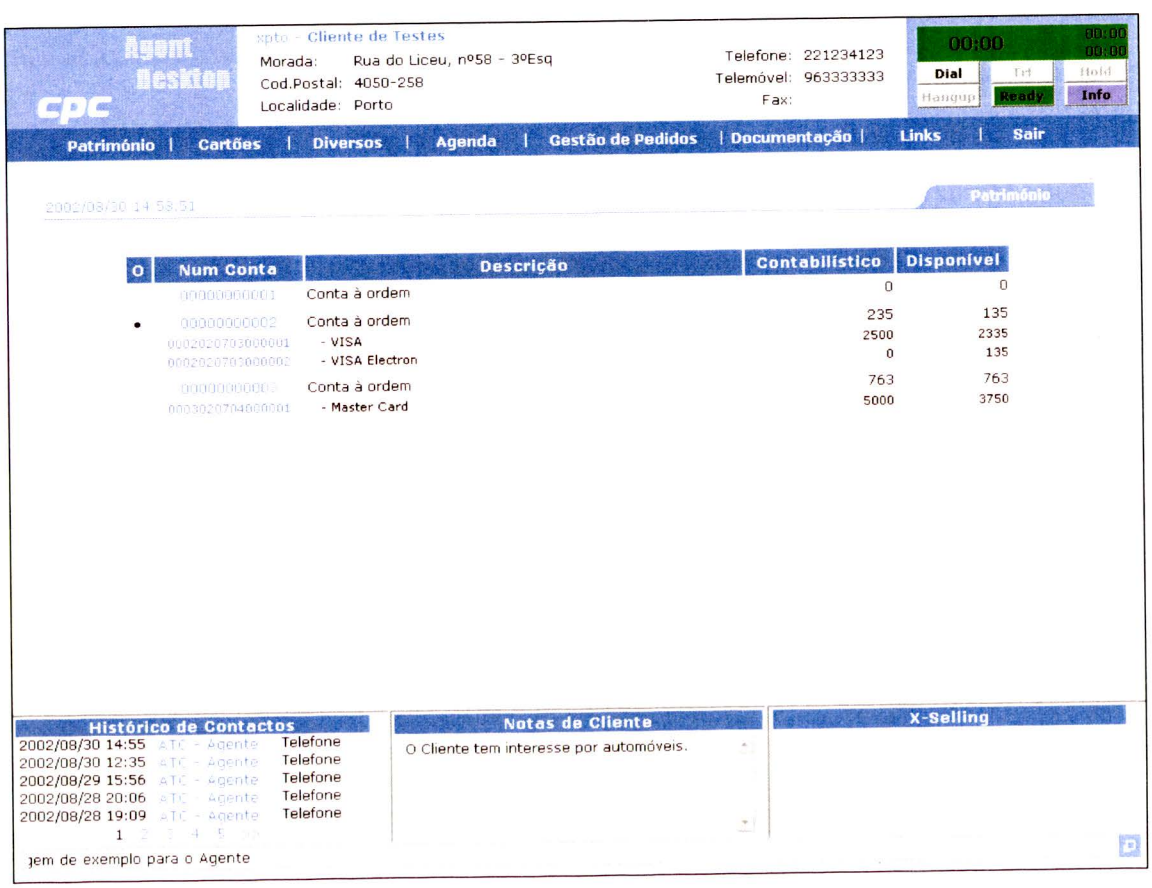


Figura C.5 – Nova Linha Branca: *Agent Desktop*

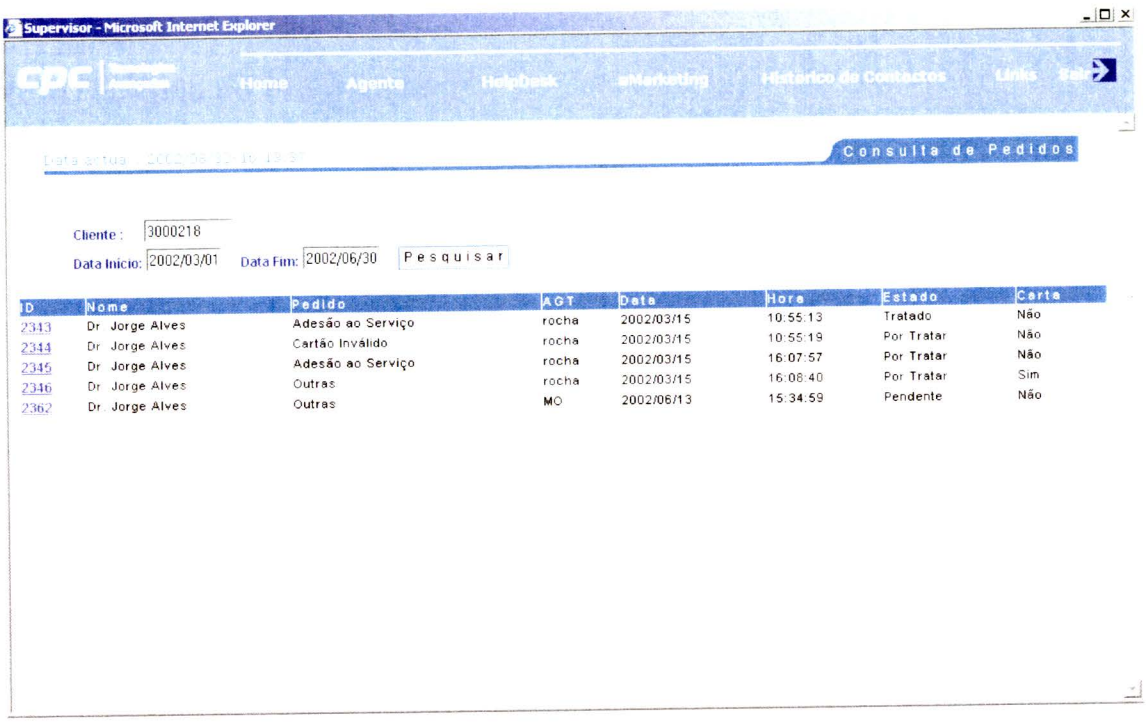


Figura C.6 – Antiga Linha Branca: *Supervisor Desktop*

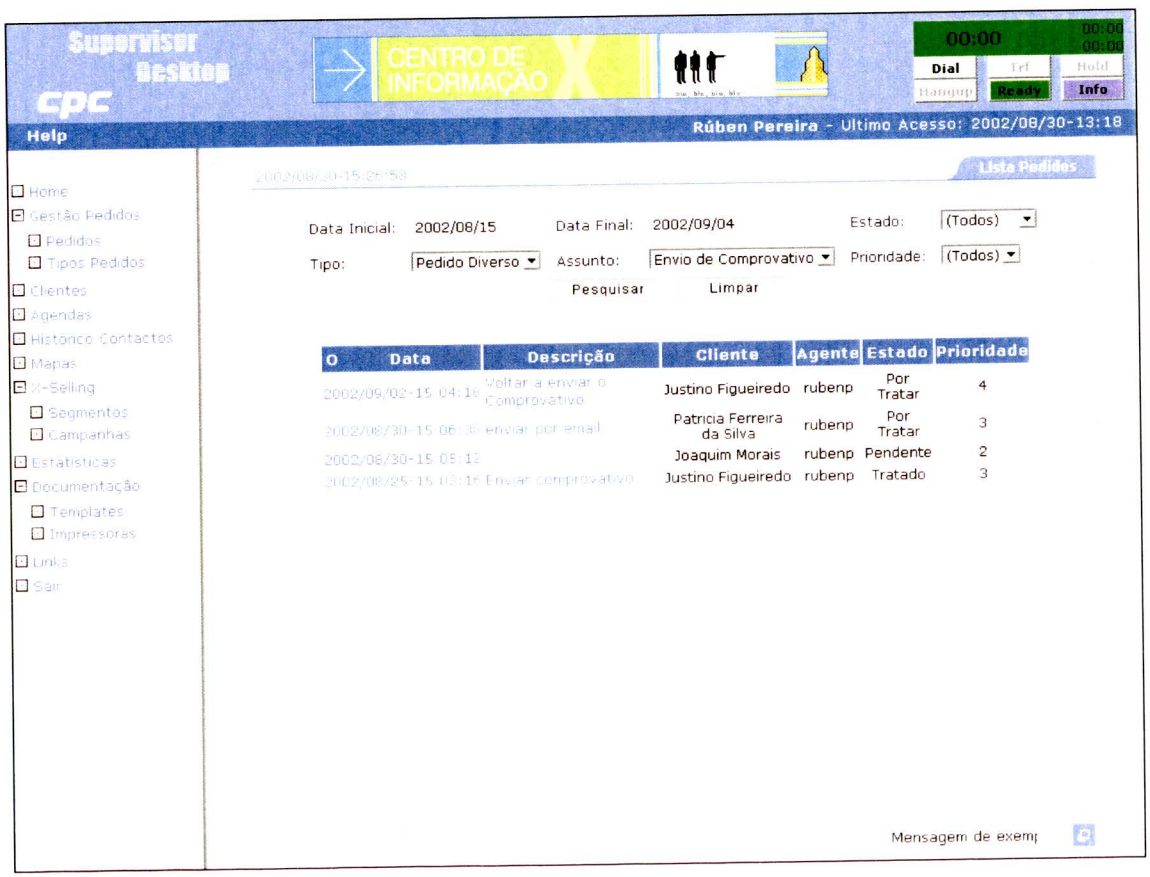


Figura C.7 – Nova Linha Branca: *Supervisor Desktop*

ANEXO D: Guia do Programador

GUIA DO PROGRAMADOR

Contact Center



4 de Setembro de 2002

Rúben Joás Pereira
CPC Tecnologias Avançadas

Índice

1. NORMAS DE CODIFICAÇÃO 3

1.1. Nomenclatura 3

1.2. Formatação 3

1.3. Comentários do Código 4

1.3.1. JavaDocs 4

1.3.2. Geração de JavaDocs no Contact Center 7

2. CONTACT CENTER – LINHA BRANCA 8

2.1. Java Transaction Server (JTS) 8

2.1.1. Serviços – Estrutura IDL 8

2.1.2. Serviços – Classe *Java* 9

2.1.3. Ficheiros `servicos.conf` 10

2.1.4. Ficheiros de Propriedades 11

2.2. Java Session Manager (JSM) 11

2.3. Java Presentation Manager (JPM) 15

2.3.1. Página Intermédia 16

2.3.2. Navegação 17

2.4. Listagens 19

2.4.1. JTS – Estrutura IDL 19

2.4.2. JTS – Classe *Java* 20

2.4.3. JSM 21

2.4.4. JPM 21

1. Normas de Codificação

1.1. Nomenclatura

Os nomes de classes, funções e variáveis deverão seguir certas normas para uma melhor compreensão do código por outras pessoas:

- Os nomes dos termos utilizados deverão ser sugestivos quanto aos dados que contenham e ao papel que desempenham no programa.
- O nome de uma classe deverá começar sempre por letra maiúscula, tanto na declaração da classe como no nome do ficheiro. Os nomes de funções e variáveis devem começar sempre por letra minúscula.
- Especialmente em linguagens de programação *case-sensitive* (como *Java*), quando o nome de uma classe, função ou variável for uma justaposição de mais que uma palavra, a primeira letra de cada palavra a seguir à primeira palavra deverá ser maiúscula.
- Em linguagens não *case-sensitive* que não mantêm a diferenciação entre letras minúsculas e maiúsculas no código (tipicamente *SQL*), as palavras que componham variáveis, métodos, campos ou tabelas deverão ser sempre em minúsculas e separadas pelo caracter '_' (ex: `data_criacao`).
- O nome de constantes deve ser todo em letras maiúsculas.
- Não devem ser usados caracteres especiais como acentos e cedilhas.
- Os nomes de variáveis não deverão ser excessivamente longos, devendo ter apenas as palavras necessárias para a sua fácil contextualização.

1.2. Formatação

O código deverá estar devidamente indentado, de forma a que o contexto de cada instrução seja facilmente perceptível.

Nos blocos entre chavetas ('{'), a chaveta inicial deverá estar na linha anterior ao bloco, e não numa nova linha:

```
if (a > b) {
      c = a;
      d = b;
}
else  {
      d = a;
      c = b;
}
```

E não:

```
if (a > b)
{
c = a;
d = b;
}
else
{
      d = a;
      c = b;
}
```

1.3. Comentários do Código

Todo o código *Java* deve estar comentado segundo as normas *JavaDoc* (http://java.sun.com/j2se/javadoc/).

Na parte inicial de cada ficheiro (tanto em *Java* como em outras linguagens) deverá constar o nome do projecto/aplicação, descrição da classe, autor, data de criação e eventuais autores de alterações ao código original, e a respectiva data de alteração. Por exemplo:

```
/**
 * Projecto: Contact Center <br>
 * Aplicação: Normas de codificação <br>
 * Descrição: Exemplo de um cabeçalho de um ficheiro <br>
 * Data: 12-04-2002 <br>
 * Autor: Rúben Pereira <br>
 * Alterado: J. Morais(03-05-2002); Rui Rocha(06-05-2002) <br>
 * @author Rúben Pereira
 * @author Rui Rocha
 * @version 2002.05.06
 */
```

A responsabilidade sobre um ficheiro cabe à ultima pessoa que o alterou. Deste modo é crucial que cada pessoa acrescente o seu nome sempre que alterar um ficheiro de código.

1.3.1. *JavaDocs*

O sistema *JavaDoc* permite que se produza documentação em formato *html* sobre o código desenvolvido. Para isso basta que o programador tome o cuidado de produzir certos comentários:

Descrição da classe – deverá conter o cabeçalho descrito anteriormente, devendo-se indicar quaisquer informação adicional que se ache necessário.
Descrição das variáveis globais ou constantes.
Descrição dos construtores – devendo-se indicar ainda uma descrição dos eventuais argumentos do construtor.
Descrição dos métodos – devendo-se indicar descrições dos argumentos e do valor retornado pelo método.

Exemplo de uma classe *Java* com comentários *JavaDoc*:

```
package exemplo;

import java.util.*;

/**
 * Projecto: Contact Center <br>
 * Aplicação: Normas de codificação <br>
 * Descrição: Exemplo um ficheiro com comentários
 * <i>JavaDoc</i>. <br>Esta classe tem uma variável global
 * <code>varGlobal</code> à qual podem ser atribuidos valores através
```

```
 * do método {@link #setValor(int)}. <br><br>
 * Data: 02-05-2002 <br>
 * Autor: Rúben Pereira <br>
 * @author Rúben Pereira
 * @version 2002.05.02
 * @see <a
href="http://java.sun.com/products/jdk/1.2/docs/tooldocs/win32/javadoc
.html#javadoctags">JavaDoc Tags</a>
 */
public class exemploJavaDoc {



/** Constante com o valor 5. */
      public final int CONSTANTE = 5;
/**
 * Variável com o valor {@link #CONSTANTE} mais o valor indicado pelo
 * utilizador.
 */
      private int varGlobal;

/**
 * Construtor.
 * Atribui ao valor de {@link #varGlobal} o <code>novoValor +
 * {@link #CONSTANTE}</code>.
 * @param novoValor Valor a adicionar.
 */
      public exemploJavaDoc(int novoValor) {
            this.varGlobal = novoValor + CONSTANTE;
      };

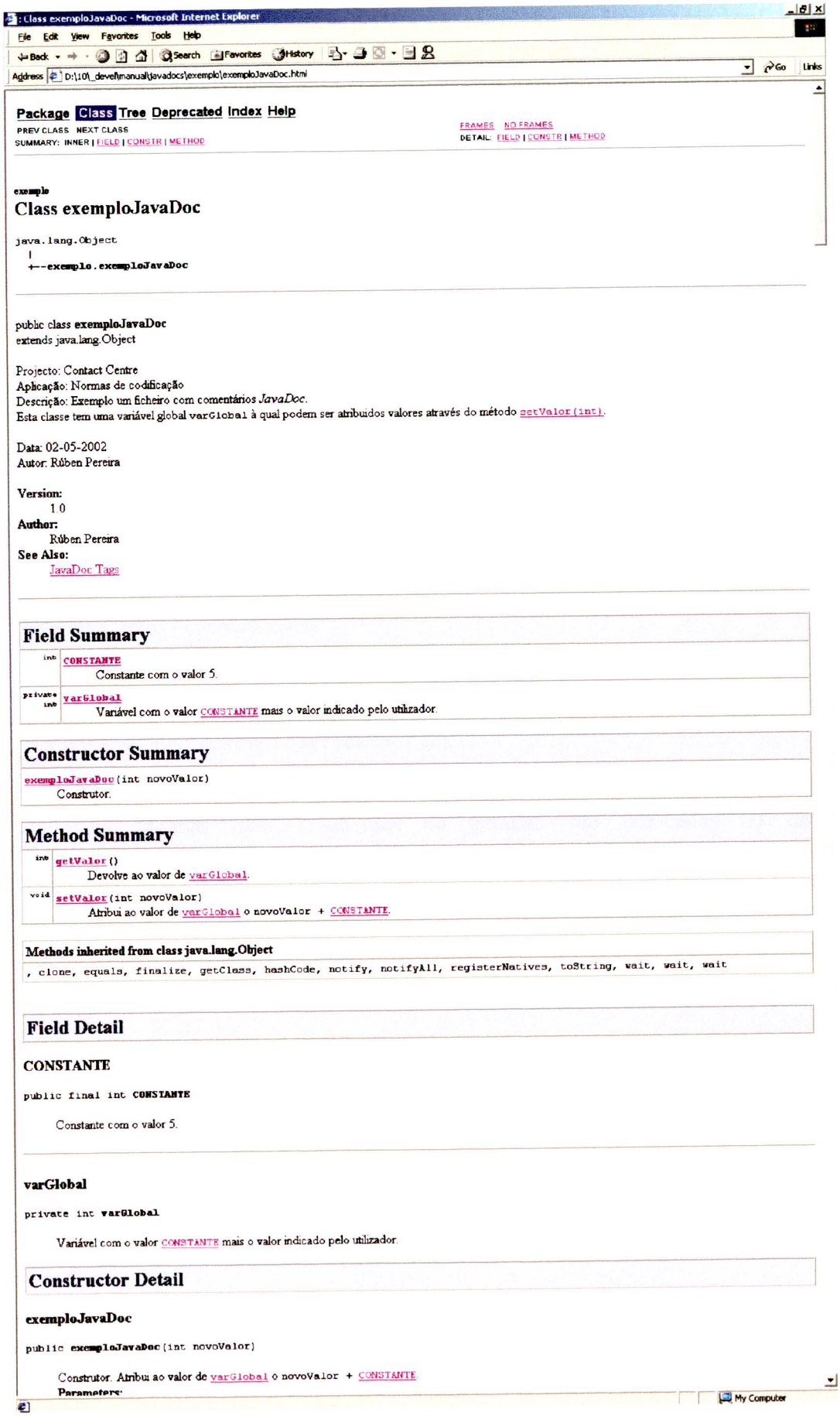
/**
 * Atribui ao valor de {@link #varGlobal} o <code>novoValor +
 * {@link #CONSTANTE}</code>.
 * @param novoValor Valor a adicionar.
 */
      public void setValor(int novoValor) {
            this.varGlobal = novoValor + CONSTANTE;
      };

/**
 * Devolve ao valor de {@link #varGlobal}.
 * @returns O valor actual de {@link #varGlobal}.
 * @exception NullPointerException se a variável não estiver
 * instanciada.
 * @see #setValor(int)
 */
      public int getValor() throws NullPointerException {
            return this.varGlobal;
      };



};
```

Ao compilar os *JavaDocs* para esta classe, o resultado seria uma página *html* do tipo:



Class exemploJavaDoc - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites History

Address D:\10_devel\manual\javadocs\exemplo\exemploJavaDoc.html

Go Links

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS

SUMMARY: INNER | FIELD | CONSTR | METHOD

FRAMES NO FRAMES

DETAIL: FIELD | CONSTR | METHOD

exemplo

Class exemploJavaDoc

```
java.lang.Object
  |
  +--exemplo.exemploJavaDoc
```

public class **exemploJavaDoc**
extends java.lang.Object

Projecto: Contact Centre
Aplicação: Normas de codificação
Descrição: Exemplo um ficheiro com comentários *JavaDoc*.
Esta classe tem uma variável global `varGlobal` à qual podem ser atribuidos valores através do método `setValor(int)`.

Data: 02-05-2002
Autor: Rúben Pereira

Version:
1.0
Author:
Rúben Pereira
See Also:
JavaDoc Tags

Field Summary

int	**CONSTANTE** Constante com o valor 5.
private int	**varGlobal** Variável com o valor CONSTANTE mais o valor indicado pelo utilizador.

Constructor Summary

exemploJavaDoc(int novoValor)
Construtor.

Method Summary

int	**getValor**() Devolve ao valor de varGlobal.
void	**setValor**(int novoValor) Atribui ao valor de varGlobal o novoValor + CONSTANTE.

Methods inherited from class java.lang.Object

, clone, equals, finalize, getClass, hashCode, notify, notifyAll, registerNatives, toString, wait, wait, wait

Field Detail

CONSTANTE

```
public final int CONSTANTE
```

Constante com o valor 5.

varGlobal

```
private int varGlobal
```

Variável com o valor CONSTANTE mais o valor indicado pelo utilizador.

Constructor Detail

exemploJavaDoc

```
public exemploJavaDoc(int novoValor)
```

Construtor. Atribui ao valor de varGlobal o novoValor + CONSTANTE.
Parameters:

My Computer

Nos comentários *JavaDoc* podem ser colocados quaisquer *tags html*. Para mais informações sobre *tags JavaDocs* e a criação dos mesmos, consultar:
http://java.sun.com/products/jdk/1.2/docs/tooldocs/win32/javadoc.html

1.3.2. Geração de JavaDocs no Contact Center

Para passar os comentários do código *Java* para as respectivas páginas *html* é necessário o uso do comando `javadoc.exe` (disponível no JDK).

Para facilitar o uso deste comando, foi criado um ficheiro *batch* para gerar automaticamente as páginas *JavaDocs*. Presentemente, este ficheiro encontra-se na máquina **Pandora** em `D:\apps\tools\makeDocs.bat`

No mesmo directório deverá existir o ficheiro `packagelist.txt` que deverá conter todos os *packages* a gerar. (cada directório criado que contenha classes *Java* (como por exemplo um novo serviço) é um novo *package* e deve ser acrescentado no ficheiro referido.)

Exemplo do ficheiro `packagelist.txt` :

```
pt.cpcta.wis
pt.cpcta.wis.cripto
pt.cpcta.wis.util
pt.cpcta.wis.jsm
pt.cpcta.wis.jts.modComum
pt.cpcta.wis.jts.modPKI
pt.cpcta.wis.jts.pki
pt.cpcta.wis.jts.sql
pt.cpcta.wis.jts.util
pt.cpcta.wis.jts.servicos.comum.ticker
...
```

As páginas *html* resultantes da geração são colocadas no directório `\apps\wis\home\documentacao\javadocs`, sendo acessíveis através da área de documentação no portal do Contact Center (http://pandora/).

2. Contact Center – Linha Branca

2.1. Java Transaction Server (JTS)

O Java Transaction Server (JTS) estabelece uma camada de ligação (Transaccional) entre as Bases de Dados e as camadas de Sessão/Apresentação.

Esta camada contém quatro directórios principais:

`wis\ts` – Contém o *core* do servidor, classes utilitárias e os **serviços**. Cada serviço é implementado através de uma classe *Java* e um ficheiro IDL, devendo ser colocado neste directório apenas a classe *Java*.
`wis\idl` – Contém os ficheiros IDL que definem as estruturas do JTS. Contém um subdirectório `base` com estruturas com definições gerais a usar para todo o JTS. Contém também o subdirectório `servicos`, que deverá conter os IDLs respectivos a cada serviço criado.
`wis\etc` – Contém os ficheiros que indicam os serviços suportados por cada aplicação (`servicos.conf`) e também os ficheiros com as propriedades (`.propriedades`).
`wis\gerado` – Contém as classes *Java* produzidas na compilação de um IDL.

Todas as classes criadas devem pertencer a um *package* apropriado, construído numa estrutura de directórios apropriada ao domínio onde estão inseridos, devendo assim os *packages* do JTS ser precedidos de **`pt.cpcta.wis.jts`**.

2.1.1. Serviços – Estrutura IDL

Cada serviço deverá ser composto uma classe *Java* e um ficheiro IDL.
O ficheiro IDL deve definir estruturas a usar nas transacções de dados. Estas estruturas tanto podem representar dados fornecidos como parâmetros para a pesquisa ou actualização de dados na Base de Dados, como os dados retornados para as outras camadas.
O IDL deverá ser colocado no directório apropriado em `wis\idl\servicos`.

Exemplo de um ficheiro IDL (*`wis\idl\servicos\comum\ticker\Ticker.idl`*):

```
#include <../../../base/modComum.idl>

module pt {
  module cpcta {
    module wis {
      module jts {
        module servicos {
          module comum {
            module ticker {

              struct InTicker {
                long idTicker;
```

```
            };

            struct OutTicker {
              long idTicker;
              string texto;
            };

          };
        };
      };
    };
  };
};
```

Se necessário, os IDLs de serviços poderão incluir `modComum.idl`. Os módulos colocados no IDL correspondem ao *package* correspondente do serviço.

As estruturas de um IDL precisam de ser compiladas, de modo a serem geradas as classes *Java* correspondentes.

Para isso deve ser usado o comando `idl2java.exe` (presente no *VisiBroker*) e com o parâmetro `-compile` no directório `wis\gerado`. Um exemplo da linha de comando seria:

```
F:\apps\wis\gerado>idl2java -compile F:\apps\wis\idl\servicos\comum\ticker\Ticker.idl
```

As classes *Java* geradas pelo comando anterior ficariam colocadas num directório com a estrutura definida pelos `modules` do IDL:

```
F:\apps\wis\gerado\pt\cpcta\wis\jts\servicos\comum\ticker\
```

2.1.2. Serviços – Classe *Java*

A classe *Java* deve conter apenas funções estáticas que normalmente acedem à Base de Dados, seja para obter, inserir, actualizar ou eliminar dados.

Cada função deverá aceitar como argumentos uma instância da classe `ServicoSessao`, passada pela camada de sessão (JSM), e uma estrutura presente no ficheiro IDL com os dados necessários para os querys ou actualizações na Base de Dados. Os argumentos devem ser sempre que possível estruturas presentes no IDL, podendo contudo também ser recebidas directamente estruturas simples como `Strings`.

Exemplo de uma função no JTS (*`TrataHelp.java`*):

```
public static InfoHelp detalheHelp(ServicoSessao ss, Identificacao id)
                        throws WISException,SQLException,WISInternalException {

    String sql="SELECT descricao, texto, inicial FROM help WHERE id_item="+id.id;

    WISConnection con=(WISConnection)ss.get("wis.interbancoATC");

    Statement stmt=con.createStatement();
    ResultSet rs=stmt.executeQuery(sql);
    if (rs.next())
        return (InfoHelp)Formatador.formataNulos(new InfoHelp(id.id,
```

```
                                rs.getString("descricao"),
                                rs.getString("texto"),
                                rs.getString("inicial").charAt(0)));
    else
        throw GlobaisTS.gme().cria(ss,1);

}
```

O valor retornado também deverá ser, sempre que se justifique, uma estrutura presente no IDL.

Nas *querys* SQL que estejam eventualmente presentes no método, os nomes das instruções SQL deverão ser sempre maiúsculas, e o nome das tabelas e campos deverão ser em minúsculas.

2.1.3. Ficheiros `servicos.conf`

Para um serviço ficar disponível para uma aplicação, é necessário adicionar ao ficheiro `servicos.conf` respectivo os nomes dos métodos de classes do JTS a disponibilizar.

O formato da lista dos serviços é:

```
metodo, package.classe, nivel_atenticacao
```

Por exemplo, para disponibilizar as funções `detalheHelp`, `insereHelp` e `alteraHelp` da classe `TrataHelp` na aplicação Administração do ATC, poderá introduzir-se no ficheiro `wis\etc\ts\atc\administracao\servicos.conf` as seguintes linhas:

```
detalheHelp, cpcis.wis.jts.servicos.interbanco.atc.administracao.trataHelp.TrataHelp, 2
insereHelp, cpcis.wis.jts.servicos.interbanco.atc.administracao.trataHelp.TrataHelp, 2
alteraHelp, cpcis.wis.jts.servicos.interbanco.atc.administracao.trataHelp.TrataHelp, 2
```

Os níveis de autenticação existentes no ficheiro `wis\idl\base\modPki.idl` :

```
const short NAO_AUTENTICADO = 0;
const short FECHADO = 1;
const short USERNAME_PASSWORD = 2;
const short USERNAME_PASSWORD_CERTIFICADO = 4;
const short USERNAME_PASSWORD_CODACESSO = 8;
const short USERNAME_CODACESSO = 16;
const short PRE_REGISTO = 32;
const short REGISTO_PREREGISTO = 64;
```

Os valores dos níveis de autenticação estão organizados na forma de potências de 2, podendo assim ser usado mais que um nível apenas somando os valores dos níveis pretendidos. Ou seja, para usar os níveis `USERNAME_PASSWORD` e `USERNAME_CODACESSO` em simultâneo deverá usar-se o nível de autenticação 18 (2+16).

2.1.4. Ficheiros de Propriedades

No directório `wis\etc` existem também vários ficheiros de propriedades (.propriedades) relativas à configuração de vários sistemas (Bases de Dados, logs, conectividades, etc).

Os ficheiros de propriedades deveram ser colocados num directório `propriedades` para cada subdirectório de `wis\etc\ts`. O nome do ficheiro deverá conter a estrutura de directórios separados por pontos. Por exemplo:

```
wis\etc\ts\propriedades\wis.propriedades
wis\etc\ts\atc\administracao\propriedades\wis.atc.administacao.proprie
dades
```

Exemplo de um ficheiro de propriedades (`wis.atc.administacao.propriedades`):

```
wis.pki.numFalhasAcesso=3

#-------------------------------------------------------------------------------------
#parametros de trace  se o nivel de trace for inferior a 2 ele nao escreve trace nenhum
#-------------------------------------------------------------------------------------
wis.trace.tee=true
wis.trace.nivel=90
wis.trace.bufferSize=0
wis.trace.excepcao=true
wis.trace.entrou_saiu=true
wis.trace.sql=true
wis.trace.pool=true
wis.trace.println=true
wis.intervaloPortas=9820,9829
wis.manutencaoTabelas.nomesBDs=wis.db,wis.interbancodb,wis.interbancoATC

# Identificador da aplicacao que esta a correr
# Serve para identificar a aplicacao do transaction e validar se o
# cliente tem acesso a esta aplicacao
wis.idAplicacao=6
```

2.2. Java Session Manager (JSM)

O Java Session Manager (JSM) é responsável por estabelecer a ligação entre a camada de Apresentação e a Transaccional (JTS).

Para cada serviço do JTS utilizado na camada de apresentação deverá existir um método numa classe no JSM para efectuar a invocação do serviço.

Os *packages* do JSM deve ser colocados no directório **`wis\sm`** devendo o nome ser precedido de **`pt.cpcta.wis.jsm`**.

Essa classe deverá obedecer a uma filosofia de *JavaBeans*, sendo composta por uma série de propriedades privadas com respectivos métodos (públicos) `set` e `get` para obter e atribuir valores.

Todos os *beans* devem estender a classe `ServicoPS`, de modo a obter acesso a métodos e propriedades da sessão, herdando as variáveis `propriedadesAplicacao`, `trace`, e `tspool`. Para a invocação de um serviço JTS, cada classe deve ter um método que trate os dados de entrada e saída e efectue a invocação. Este método não deverá ser chamado directamente, mas através dos métodos `invocaMetodoTS` ou `invocaMetodoLSTS`, herdados da classe `ServicoPS`. Antes da chamada a estes métodos deve ser atribuído à propriedade `metodo` o nome do método que efectua a invocação no *bean*, por exemplo `setMetodo("nomeMetodoJSM")`.

Para a melhor compreensão dos conceitos introduzidos, seguidamente é apresentado um exemplo de uma classe típica do JSM:

(wis\sm\pt\cpcta\wis\jsm\servicos\comum\publishing\ListaLinks.java)

```
package pt.cpcta.wis.jsm.comum.publishing;

import pt.cpcta.wis.jts.servicos.comum.publishing.trataLinks.*;
import java.util.Properties;
import pt.cpcta.wis.*;
import pt.cpcta.wis.jsm.*;
import pt.cpcta.wis.util.*;
import pt.cpcta.wis.jts.modComum.*;
import org.apache.log4j.Category;

/**
 * Projecto: Contact Center - Linha Branca <br>
 * Aplicação: Portal <br>
 * Descrição: Bean para invocar o serviço para obter os detalhes dos links <br>
 * Data: 22-05-2002 <br>
 * Autor: Rúben Pereira <br>
 * Alterado: (07-06-2002)<br>
 * @author Rúben Pereira
 * @version 2002.06.07
 */
public class ListaLinks extends ServicoPS{

  /** Variável onde são guardadas os links */
  private InfoListaLink[] listaLinks;


  /** Variável onde é guardada o valor do idAplicacao */
  private String idAplicacao;
  /** Variável onde é guardada a página actual */
  private int pagina;
  /** Variável onde é guardado o tamanho da página actual */
  private int tamPagina;
  /** Variável onde é guardado o numero de páginas */
  private int numPaginas;
  /** Variável onde é guardada a ordenação da listagem */
  private String orderBy;
  /** id de sessao */
  private String idSessao;


  /**
   * Atribui um valor ao idSessao.
   * @param sessao    Valor da Sessão.
   */
  public void setIdSessao(String sessao){
    idSessao=sessao;
  }
  /**
   * Devolve o id de Sessao.
   * @return idSessao.
   */
  public String getIdSessao(){
    return idSessao;
  }

  /** Devolve a lista de Links.
   * @return <i>Array</i> com todos links.
   */
  public InfoListaLink[] getListaLinks(){
    return listaLinks;
  }
  /** Atribui um valor a lista de Links.
   * @param lista    <i>Array</i> com todos links.
   */
  public void setListaLinks(InfoListaLink lista){
    listaLinks = lista;
  }

  /** Devolve o idAplicacao.
```

```
   * @return identificação Aplicação.
   */
  public String getIdAplicacao(){
    return idAplicacao;
  }


        /* ... */


  /** Devolve o número de páginas.
   * @return Número total de páginas retornadas.
   */
  public int getNumPaginas(){
    return numPaginas;
  }


  /** CONSTRUTOR - Inicializa as variàveis */
  public ListaLinks() {
        setIdSessao("");
        setPagina(0);
        setTamPagina(0);
        setOrderBy("");
        setIdAplicacao("");  //todas as aplicacoes
        setListaLinks(new InfoLink[0]);
  }


  /** Faz a invocação à camada transaccional para carregar os dados da BD. (sem
autenicação) */
  public void getListaLinksTS() throws WISException {
    trace.debug("Chamada ao METODO: ListaLinks.getListaLinksTS()");

    if (!propriedadesAplicacao.getConfig().getPropertyAsBoolean("simulacao", false)) {
        trace.debug("Vai ligar ao ts");

        InLinks in = new InLinks();
        in.idAplicacao = getIdAplicacao();
        in.pagina = getPagina();
        in.tamanhoPagina = getTamPagina();
        in.orderBy = getOrderBy();

        Object[] servicos = new Object[1];
        servicos[0] = (InLinks) Formatador.formataNulos(in);

        OutListaLinks dados = (OutListaLinks) InvocaTS.invoca(lsts.orb,
                    lsts.sessaoPKI,
       "pt.cpcta.wis.jts.servicos.comum.publishing.trataLinks.TrataLinks.getListaLinks",
                    servicos,
                    OutListaLinks.class);

        trace.debug("retornou do ts ");
        setListaLinks(dados.lista);
        setPagina(dados.pagina);
        setNumPaginas(dados.numPaginas);

    } else {        //Simulação
        InfoLink[] links = new InfoLink[1];
        links[0] = new InfoLink(-1, "http://www.cpcis.pt", "CPCIS (Simulação)", 1,
{"Portal", "Agente"});
        setListaLinks(links);
        setPagina(1);
        setNumPaginas(1);
    }

  }


  /** Faz a invocação à camada transaccional para carregar os dados da BD. (com
autenicação) */
  public void getListaLinksTS(LigacaoSessaoTS _lsts) throws WISException {
    trace.debug("Chamada ao METODO: ListaLinks.getListaLinksTS(_lsts)");

    if (!propriedadesAplicacao.getConfig().getPropertyAsBoolean("simulacao", false)) {
        trace.debug("Vai ligar ao ts");
```

```
        InLinks in = new InLinks();
        in.idAplicacao = getIdAplicacao();
        in.pagina = getPagina();
        in.tamanhoPagina = getTamPagina();
        in.orderBy = getOrderBy();

        Object[] servicos = new Object[1];
        servicos[0] = (InLinks) Formatador.formataNulos(in);

        OutListaLinks dados = (OutListaLinks) InvocaTS.invoca(_lsts.orb,
                    _lsts.sessaoPKI,
      "pt.cpcta.wis.jts.servicos.comum.publishing.trataLinks.TrataLinks.getListaLinks",
                    servicos,
                    OutListaLinks.class,
                    getIdSessao());
        setListaLinks(dados.lista);
        setPagina(dados.pagina);
        setNumPaginas(dados.numPaginas);
    }
  }


  /** Este método deve ser chamado quando se quer fazer um update no transaction
   * Usado em conjunto com os comandos: inserir, alterar, remover.
   * @exception WISException esta excepcao é lançada pela plataforma WIS. */
  public void update() throws WISException {
    if (idSessao.equals(""))
        invocaMetodoTS();
    else
        invocaMetodoLSTS();
  }

}
```

O exemplo anterior contém dois métodos com o nome `getListaLinksTS` que invocam o serviço do JTS. A diferença entre estes dois métodos é que no segundo o utilizador necessita de estar autenticado, tendo obtido um `idSessao`. A construção de métodos com e sem autenticação contém algumas variações, nomeadamente, a utilização do objecto de `LigacaoSessaoTS _lsts` (recebido como parâmetro) em vez de `lsts` (global herdada de `ServicoPS`), e também a inclusão do `idSessao` na invocação do serviço. Como foi referido a chamada aos métodos não deve ser efectuado directamente, mas através dos métodos `invocaMetodoTS()` (sem autenticação) ou `invocaMetodoLSTS()` (com autenticação – Ligação Sessão). Por sua vez, estes métodos deverão ser referenciados a partir de um método `update()` no *bean*. Por exemplo, a chamada do serviço a partir de uma página JSP poderia ser efectuado da seguinte forma:

```
<%
ListaLinks beanLinks = new ListaLinks();
beanLinks.setMetodo("getListaLinksTS");
beanLinks.setIdSessao(idSessao);
beanLinks.update();
%>
```

Nota: *Este exemplo não obedece às normas gerais na declaração de JavaBeans em JSPs, explicadas mais à frente na secção dedicada ao JPM.*

Nos métodos a incluir nos *beans* para a invocação de serviços, deve ser verificada o estado da propriedade `simulacao`, que se encontra definida em ficheiros de propriedades e é obtida a partir das `propriedadesAplicação`. Caso esta propriedade tenha o valor `true`, não deverá ser feita qualquer invocação ao JTS, podendo-se simular a transacção e retornar objectos predefinidos em código (*hardcoded*).

A função para invocar um serviço (`InvocaTS.invoca`) aceita como parâmetros, algumas propriedades da `LigacaoSessaoTS`, o nome do serviço a invocar (incluído num ficheiro `servicos.conf`), os parâmetros de entrada do serviço colocados no *array* de

objectos `servicos`, a classe do objecto retornado (`null` se o serviço não retornar nenhum valor), e o id da Sessão (caso o serviço necessite de autenticação).

2.3. Java Presentation Manager (JPM)

O Java Presentation Manager (JPM) é constituído pelas páginas *web* (sobretudo JSP) que implementam os interfaces das aplicações.

As páginas do JPM podem ser encontradas no directório **`wis\pm`**, existindo um directório para cada aplicação seguido de um directório `jsp`. Por exemplo, as páginas da aplicação Agente estão no directório **`wis\pm\agente\jsp`**.

Todas as páginas JSP devem incluir o ficheiro `inicioPagina.jsp`, de modo a terem acesso a várias variáveis guardadas em propriedades ou em *cookies*. Alguns exemplos são o `idAplicacao` actual (propriedade), `idSessaoAgente` e `idAgente` (*cookies*). Também instância objectos como o `trace`, para efectuar *debbuging* nas páginas.

No que diz respeito à inclusão de *JavaBeans*, implementados no JSM, estes podem ser incluídos como está ilustrado no seguinte exemplo:

```
<%@ include file="inicioPagina.jsp" %>

<jsp:useBean id="listaLinks" scope="page"
class="pt.cpcta.wis.jsm.comum.publishing.ListaLinks" />
<jsp:setProperty name="listaLinks" property="*" />
<jsp:setProperty name="listaLinks" property="nomeAplicacao" value="<%=nomeAplicacao%>"/>
<jsp:setProperty name="listaLinks" property="metodo" value="getListaLinksTS"/>
<jsp:setProperty name="listaLinks" property="idSessao" value="<%=idSessaoAgente%>"/>
<jsp:setProperty name="listaLinks" property="idAplicacao" />

<% listaLinks.update(); %>
```

O exemplo anterior instancia o *bean* através da *tag* `jsp:useBean`. Seguidamente pode-se usar as *tags* `jsp:setProperty` e `jsp:getProperty` para atribuir valores às propriedades do *bean* (para isto acontecer a propriedade tem que ter métodos `set` e `get` na classe que implementa o *bean*). Quando numa *tag* `jsp:setProperty` não é referido o valor a atribuir à propriedade (como na propriedade `idAplicacao` no exemplo), é atribuído automaticamente o valor de um argumento com o mesmo nome recebido por *request* (seja pelo *url* ou por *form*), sem ser necessário utilizar o método `request.getParameter` para receber os parâmetros enviados. Um caso especial é quando `property="*"`, o que provoca com que todos os parâmetros recebidos sejam analisados, e todas as propriedades que tenham métodos `set` correspondentes sejam actualizadas.

Ou seja, tendo em conta o exemplo anterior numa página JSP, referenciado o *bean* exemplificado anteriormente na secção do JSM, se por exemplo, a página fosse chamada na forma `listaLinks.jsp?pagina=3&idAplicacao=5&x=abc`, a referência ao *bean* iria dar origem à seguinte sequência de operações:

```
ListaLinks listaLinks = new ListaLinks();          // (useBean)
listaLinks.setPagina(3);                           // (setProperty - *)
listaLinks.setIdAplicacao(5);                      // (setProperty - *)
listaLinks.setNomeAplicacao(nomeAplicacao);        // (setProperty - nomeAplicacao)
listaLinks.setMetodo("getListaLinksTS");           // (setProperty - metodo)
listaLinks.setIdSessao(idSessaoAgente);            // (setProperty - idSessao)
listaLinks.setIdAplicacao(5);                      // (setProperty - idAplicacao)
```

De notar que o parâmetro `x=abc` é ignorado visto não existir nenhum método `setX` na classe `ListaLinks`. Os métodos `setNomeAplicacao` e `setMetodo`, não existem na classe `ListaLinks` mas são herdados da classe pai `ServicoPS`.

Para invocar um serviço do JTS a partir duma página JSP deverá sempre atribuir-se um valor ao **`nomeAplicacao`** (para inicializar as `propriedadesAplicacao`) e ao **`metodo`**, devendo também indicar-se o **`idSessao`** nos serviços que necessitem de autenticação. Para efectuar a invocação do serviço bastará então chamar o método **`update`** do *bean*.

No que diz respeito ao código HTML, existem também algumas normas a seguir. No cabeçalho HTML deverá existir *tags* para impedir que a página seja guardada em *cache*. Visto que a maioria das páginas são geradas dinamicamente no lado do servidor, se estas fossem guardadas na *cache* dos *browsers*, a sua correcta actualização poderia ficar comprometida.

Outro aspecto são os estilos (CSS) usados. Todos os estilos a usar estão contidos no ficheiro `estilos.css` (referenciada pela variável **`vEstilo`** no `InicioPagina`) e não deverá ser utilizado nem criado nenhum estilo que não esteja contido neste ficheiro. Caso seja necessário a utilização de um novo estilo não presente no ficheiro, este deve ser acrescentado ao ficheiro de estilos.

Nas páginas gerais de dados, deve existir um cabeçalho standard com o título da página actual. Este cabeçalho é produzido através do método **`geraCabecalho`**, presente na classe `UtilLinhaBranca`.

O seguinte exemplo mostra o uso das normas referidas:

```
<%@ include file="inicioPagina.jsp" %>

...

<html>
<head>
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="0">

<link rel="stylesheet" href='<%=vEstilo%>' type="text/css">
</head>

<body >
<%=UtilLinhaBranca.geraCabecalho("Links")%>

...
```

2.3.1. Página Intermédia

Todas as passagens de uma página para outra devem utilizar uma página intermédia que mostrará uma mensagem de espera enquanto a nova página não carrega.

Esta página intermédia (normalmente com o nome `intermedia.jsp`) aceita como parâmetro `destino` com o valor da página de destino final. Qualquer outro parâmetro enviado no URL para a página intermédia é reenviado para a página de destino.

Por exemplo:

```
<a href="intermedia.jsp?destino=dados.jsp">
```

Ao ser activado o *link* abria simplesmente a página `dados.jsp`.

```
<a href="intermedia.jsp?destino=dados.jsp&param1=val1&param2=val2">
```

Este *link* já abria a página `dados.jsp` com todos os parâmetros seguintes, sendo equivalente a um *link* na forma: `href="dados.jsp?param1=val1&param2=val2"`.

O funcionamento é semelhante no uso de *forms*. Os parâmetros que se pretenda passar à página de destino e que e não sejam visíveis no formulário, deverão ser colocados em campos do tipo `hidden`:

```
<form name="formDados" method="post" action="intermedia.jsp">
<input type="hidden" name="destino" value="detalhe.jsp">
<input type="hidden" name="idLink" value="<%= IdLink %>">
<input type="hidden" name="nome" value="<%= nome %>">
<input type="hidden" name="idAplicacao" value="">
<input type="hidden" name="pagina" value="<%= pagina %>">
<input type="hidden" name="ordem" value="<%= ordem %>">
...
```

Nota: A anterior utilização do *separador* `*/*`, e a construção de todos os parâmetros no campo `destino`, não são suportadas na nova página intermédia.

2.3.2. Navegação

Para facilitar a criação de funcionalidades de navegação (*links* para voltar, passagem de argumentos, etc) foi criada uma *script* JSP (`voltarUtil.jsp`) para a geração automática de *links*.

Este ficheiro deverá ser incluído na parte inicial das páginas, devendo ser previamente inicializada a variável `nomePagina`:

```
...
<% String nomePagina = "dados.jsp"; %>
<%@ include file="voltarUtil.jsp" %>
...
```

Após a inclusão da *script*, ficam disponíveis três variáveis JSP:

`voltarPaginaActual` – Contém parâmetros para voltar à página actual (de uma outra pagina) com todos os parâmetros recebidos, na forma:

```
back_page=pagina.jsp&back-param1=val1&back-param2=val2&...
```

`voltarPaginaAnterior` – Produz os parâmetros para voltar à página anterior, caso esta tenha enviado como parâmetro a *String* produzida previamente em `voltarPaginaActual`. Os parâmetros contidos nesta variável ficam na forma:

'pagina.jsp&param1=val1&param2=val2&...

voltarMantemPagina – Reproduz os parâmetros para voltar à página anterior produzidos previamente (em `voltarPaginaActual`) para permitir que estes dados não se percam em chamadas na própria página (paginação, ordenação, etc).

Aqui estão exemplos do uso de cada uma das variáveis:

Enviar parâmetros a uma próxima página para posteriormente **voltar** à **página actual**:

```
<a class="LINK" href="intermedia.jsp?destino=detalheNoticia.jsp&id=<%=
listaNoticias[i].idNoticia %>&<%= voltarPaginaActual %>">
```

Construir um link para **voltar** à **página anterior**:

```
<input type="button" class="BUTTON" name="voltar" value="Voltar"
onClick="location.href='intermedia.jsp?destino=<%= voltarPaginaAnterior %>'">
```

Manter os parâmetros recebidos de trás em chamadas para a própria página:

```
function novo(param) {
      location.href="intermedia.jsp?destino=noticias.jsp&paginacao="+param+
"&ordem="+document.formOrdenacao.ordem+ "&<%= voltarMantemPagina %>";
}
```

As variáveis produzidas apenas estão preparadas para serem acrescentadas a uma chamada *href* comum. Para utilizar chamadas a outras páginas através de *forms* existe a função **href2Form** da classe **UtilLinhaBranca**, que converte *strings* de parâmetros de URL (na forma das variáveis anteriores), e converte-as em campos do tipo `hidden`.

Por exemplo, se `voltarPaginaAnterior="pagina.jsp&param1=val1&param2=val2"`, então **UtilLinhaBranca.href2Form(voltarPaginaAnterior)** retornaria o seguinte código:

```
<input type="hidden" name="destino" value="pagina.jsp">
<input type="hidden" name="param1" value="val1">
<input type="hidden" name="param2" value="val2">
```

De igual forma o comando
UtilLinhaBranca.href2Form("back_page=pagina.jsp&back-param1=val1&back-param2=val2")
produziria o código:

```
<input type="hidden" name="back_page" value="pagina.jsp">
<input type="hidden" name="back-param1" value="val1">
<input type="hidden" name="back-param2" value="val2">
```

2.4. Listagens

Salvo algumas excepções, as listagens devem obedecer a certas regras de formatação, paginação e ordenação.

Para esse efeito existem várias funcionalidades já implementadas para facilitar a criação de listagens *standard*. Para o seu correcto funcionamento o programador deverá seguir certas normas na elaboração de estruturas, classes e páginas para a criação das listagens.

2.4.1. JTS – Estrutura IDL

Tanto os parâmetros necessários para efectuar o *query* como o valor retornado pela função (serviço) deverão ser colocados em estruturas de entrada e saída de dados no ficheiro IDL (como descrito na secção de **Serviços** do **JTS**). O nome dessas estruturas deverá começar por `In` ou `Out` respectivamente (por exemplo: `InListaEmails, OutListaEmails`). No caso de listagens, a estrutura de saída deverá conter um *array* de elementos de uma outra estrutura presente no IDL, que contém os dados a colocar em cada linha da lista. Por exemplo, as estruturas no ficheiro `TrataNoticias.idl` são:

```
struct InListaNoticias {
    long pagina;
    long tamanhoPagina;
    string orderBy;
};

struct InfoListaNoticias {
    long idNoticia;
    long idAplicacao;
    string titulo;
    string resumo;
    char destaque;
    string descAplicacao;
    pt::cpcta::wis::jts::modComum::ObjDataHora dataCriacao;
};

typedef sequence <InfoListaNoticias> listaNoticias;
struct OutListaNoticias {
    listaNoticias lista;
    long pagina;
    long numPaginas;
};
```

Para o sistema de paginação, deverá incluir-se na estrutura de entrada as variáveis `pagina` e `tamanhoPagina`, que representam o número da página que se pretende mostrar, e o número de elementos por página.

De igual forma a estrutura de saída deverá conter variáveis com o número da página mostrada e o número total de páginas.

Para o sistema de ordenação, a estrutura de entrada deverá conter a variável `orderBy`, onde serão colocados os campos da BD pelos quais a lista deve ser ordenada.

2.4.2. JTS – Classe *Java*

Para efectuar a ordenação deverá-se colocar no fim do *query* SQL, a instrução `ORDER BY`, seguida da *string* contida na variável `orderBy` na estrutura de entrada da listagem (passada como argumento do método). Por exemplo:

```
String sql ="SELECT n.id_noticia, na.id_aplicacao, n.titulo, n.resumo, n.destaque,
a.descricao AS desc_aplicacao, n.data_criacao " +
        " FROM noticias n, noticiasaplicacao na, aplicacoes a " +
        "WHERE n.id_noticia=na.id_noticia AND a.id_aplicacao=na.id_aplicacao";
sql += (in.orderBy.equals("")? "": " ORDER BY " + in.orderBy);
```

Para a paginação é necessário calcular o número total de páginas e quais os registos a retirar no `ResultSet`, baseado no número de elementos por página, a página actual e o número de elementos no `ResultSet`.

Aqui está um exemplo de uma implementação:

```
...
//PARA OBTER O NUMERO TOTAL DE LINHAS
Statement stmt2 = publishingCon.createStatement();
ResultSet rs2 = stmt2.executeQuery("select @@rowcount");
if(rs2!=null && rs2.next()) {
    tamanhoTotal = rs2.getInt(1);
}
rs2.close();
stmt2.close();

int numPaginas = 0;
if (pagina==0) { //Se for ir buscar tudo pagina=0
    tamanhoPagina=tamanhoTotal;
    numPaginas = 1;
} else {
    numPaginas = (int)tamanhoTotal/tamanhoPagina + (tamanhoTotal%tamanhoPagina!=0?1:0);
    if (pagina > numPaginas)
        pagina = numPaginas;
    int posInicial = (pagina-1)*tamanhoPagina;
    if (posInicial > 0) rs.absolute(posInicial);
}

Vector vector = new Vector();
InfoListaNoticias info;
for (int i=tamanhoPagina ; rs!=null && rs.next() && i > 0; i--){
    info = (InfoListaNoticias) Formatador.formataNulos(new InfoListaNoticias());
    info.idNoticia = rs.getInt("id_noticia");
    info.idAplicacao = rs.getInt("id_aplicacao");
    info.titulo = rs.getString("titulo");
    info.resumo = rs.getString("resumo");
    info.destaque = rs.getString("destaque").charAt(0);
    info.descAplicacao = rs.getString("desc_aplicacao");
    info.dataCriacao = DataUtil.sqlTimestampToDataHora(rs.getTimestamp("data_criacao"));

    vector.addElement(info);
}

//Se não encontrou registos para aqueles parametros lança excepçao
if(vector.size()==0){
    throw GlobaisTS.gme().cria(ss, 15);
}//Se encontrou registos devolve um array
else{
    // O InfoListaNoticias[] é o campo que vai ser retornado na OutListaNoticias.
    InfoListaNoticias[] listaNoticias = new InfoListaNoticias[vector.size()];
    vector.copyInto(listaNoticias);
    OutListaNoticias outLista = new OutListaNoticias();
    outLista.lista = listaNoticias;
    outLista.pagina = pagina;
    outLista.numPaginas = numPaginas;
    return outLista;
}
...
```

2.4.3. JSM

Os *JavaBeans* da camada de sessão deverão conter as variáveis e métodos para passar os dados de paginação e ordenação da camada de Apresentação para o servidor Transaccional, e vice-versa.

A implementação proposta teria as variáveis (privadas) `pagina`, `tamanhoPagina`, `numPaginas` e `orderBy`. Devendo existir métodos *set* e *get* onde necessário.

Após a invocação do serviço JTS devem ser actualizadas as variáveis respectivas (`pagina` e `numPaginas`).

2.4.4. JPM

Na camada de apresentação é necessário incluir várias porções de código para implementar a paginação e ordenação da listagem.

Inicialmente na página *jsp* devera-se inicializar as variáveis respectivas e receber eventuais argumentos do *request*:

```
int page = 1;
int tamanhoPagina = 15;
int tamanho = 0;
if (request.getParameter ("pagina") != null)
    page = Integer.parseInt(request.getParameter("pagina"));

String campos = "Data/Titulo/Aplicacao/Destaque";
String camposBD = "n.data_criacao/titulo/desc_aplicacao/destaque";
String ordem = "Data DESC/Titulo";
if ((request.getParameter("ordem") != null) &&
            (!request.getParameter("ordem").equals("")))
    ordem = request.getParameter("ordem");
```

No exemplo anterior, a página actual é representada na variável `page`, podendo ser modificada se no *request* receber um parâmetro `pagina`. O `tamanhoPagina` indica o número de elementos por página, e o `tamanho` irá posteriormente conter o número total de páginas existentes.

Para a ordenação existem as variáveis `campos` e `camposBD`, representando os campos ordenáveis presentes no `SELECT` efectuado no JTS para obter a listagem. A diferença entre `campos` e `camposBD` é que a variável `campos` representa a descrição de cada campo (visível ao utilizador) e `camposBD` corresponde ao nome dos campos no `SELECT`, devendo incluir o nome da tabela se necessário.
A variável `ordem` contém os campos (descrições) pelos quais a listagem deve ser ordenada. Esta variável deverá ser inicializada com a ordenação pretendida, podendo ser alterada por *request*. A descrição pode ser acrescida de " `DESC`" para indicar uma ordenação decrescente.

Nas três variáveis de ordenação referidas os campos deverão ser separados por barras "/". A `ordem` poderá ser convertida para uma *string* pronta a adicionar a uma instrução `ORDER BY` de SQL através da função `geraOrderBy()` da classe `ListagemUtil`.

Seguidamente é necessário passar os dados de paginação e/ou ordenação para o JSM (que irá passar esses parâmetros para o JTS) e receber os resultados respectivos. Por exemplo:

```
<jsp:useBean id="ListaNoticias" scope="page"
class="pt.cpcta.wis.jsm.comum.publishing.ListaNoticias" />
<jsp:setProperty name="ListaNoticias" property="nomeAplicacao"
value="<%=nomeAplicacao%>"/>
<jsp:setProperty name="ListaNoticias" property="metodo" value="getListaNoticiasTS"/>
<jsp:setProperty name="ListaNoticias" property="pagina" value="<%=page%>"/>
<jsp:setProperty name="ListaNoticias" property="tamanhoPagina"
value="<%=tamanhoPagina%>"/>
<jsp:setProperty name="ListaNoticias" property="orderBy"
value="<%=ListagemUtil.geraOrderBy(campos,camposBD,ordem)%>"/>

<%
ListaNoticias.update();
InfoListaNoticias[] listaNoticias = ListaNoticias.getListaNoticias();

page = ListaNoticias.getPagina();
tamanho = ListaNoticias.getNumeroPaginas();
%>
```

Neste ponto, a listagem retornada já corresponde à página e ordenação indicada como *default* ou recebida no *request*.
Para utilizar o sistema generalizado de paginação e ordenação deverá ainda criar-se algumas funções de *JavaScript* e elementos HTML. O código já implementado de paginação chama a função de *JavaScript* `novo(int novaPagina)`, que deverá ser implementada pelo programador devendo (em principio) fazer uma chamada à própria página passando como argumento a nova página a mostrar, tendo em atenção os outros argumentos recebidos previamente pela página, que deverão ser reenviados.

De igual forma as funções utilitárias para efectuar a ordenação chamam a função `ordena()`, que deverá executar a ordenação (através da função `sortFieldsChooser` da *script* representada pela variável `jsOrdena`), e fazer uma chamada à própria página com a nova ordenação. Este método pode ser integrado com a função `novo(int novaPagina)`, em páginas que usem ambas paginação e ordenação.
Exemplo de uma implementação conjunta das funções de *JavaScript*:

```
...
<script src="<%= jsOrdena %>"></script>
<script>
function ordena() {
        sortFieldsChooser(document.formOrdenacao.ordem,'<%=campos%>');
        novo(<%=page%>);
}

function novo(param) {
        document.location.href = "listaNoticias.jsp? tipoPesquisa=<%= tipoPesquisa
%>&pagina=" + param + "&ordem=" + document.formOrdenacao.ordem.value
+ '&<%= voltarMantemPagina %>';
}
</script>
</head>
```

A função `sortFieldsChooser` deverá receber como parâmetros o objecto que contém a ordem actual (e que irá conter a nova ordenação) e todos os campos

disponíveis para a ordenação. O objecto `formOrdenação.ordem` representa o campo que deverá conter a ordenação actual. Normalmente, este campo (e o respectivo formulário) é gerado automaticamente através do método `ListagemUtil.geraCelulaOrdenacao`.

De momento, para efectuar a ordenação deverá ser colocado um ícone no canto superior esquerdo da tabela da listagem (junto ao cabeçalho).

Para este feito existem várias funções `geraCelulaOrdenacao` na classe `UtilLinhaBranca`. De um modo geral, esta função deverá ser colocada na primeira célula do cabeçalho da tabela, devendo ter como parâmetros o título (que deveria estar nessa célula) e a ordenação actual.

Aqui está um exemplo do uso geral deste método:

```
...
<table align="center" width="80%" border="0">
<tr>
        <td align="center" class="TEXTOBG">
<%= ListagemUtil.geraCelulaOrdenacao("Data", ordem) %>
        </td>
        <td align="center" class="TEXTOBG">Titulo</td>
        <td align="center" class="TEXTOBG">Aplicacao</td>
        <td align="center" class="TEXTOBG">Destaque</td>
</tr>
...
```

No código HTML gerado pelo método `geraCelulaOrdenacao` encontra-se um formulário (`formOrdenacao`) e para o seu correcto funcionamento, a função **não pode ser colocada dentro de outro *form***.

Finalmente, para gerar os *links* para a paginação deverá incluir-se o ficheiro `Paginação.jsp` no fim da tabela com a listagem:

```
</table>
<%@ include file="Paginacao.jsp"%>

</body>
</html>
```

disponíveis para a ordenação. O objecto `formOrdenação.ordem` representa o campo que deverá conter a ordenação actual. Normalmente, este campo (e o respectivo formulário) é gerado automaticamente através do método `ListagemUtil.geraCelulaOrdenacao`.

De momento, para efectuar a ordenação deverá ser colocado um ícone no canto superior esquerdo da tabela da listagem (junto ao cabeçalho).

Para este feito existem várias funções `geraCelulaOrdenacao` na classe `UtilLinhaBranca`. De um modo geral, esta função deverá ser colocada na primeira célula do cabeçalho da tabela, devendo ter como parâmetros o título (que deveria estar nessa célula) e a ordenação actual.

Aqui está um exemplo do uso geral deste método:

```
...
<table align="center" width="80%" border="0">
<tr>
        <td align="center" class="TEXTOBG">
<%= ListagemUtil.geraCelulaOrdenacao("Data", ordem) %>
        </td>
        <td align="center" class="TEXTOBG">Titulo</td>
        <td align="center" class="TEXTOBG">Aplicacao</td>
        <td align="center" class="TEXTOBG">Destaque</td>
</tr>
...
```

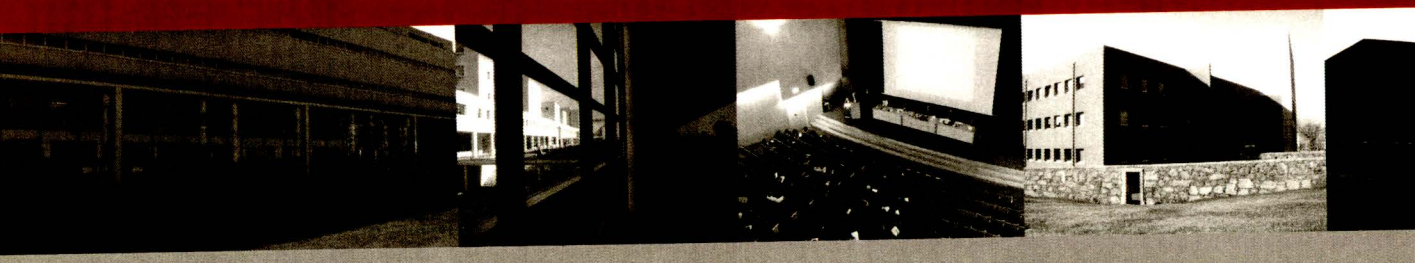
No código HTML gerado pelo método `geraCelulaOrdenacao` encontra-se um formulário (`formOrdenacao`) e para o seu correcto funcionamento, a função **não pode ser colocada dentro de outro *form***.

Finalmente, para gerar os *links* para a paginação deverá incluir-se o ficheiro `Paginação.jsp` no fim da tabela com a listagem:

```
</table>
<%@ include file="Paginacao.jsp"%>

</body>
</html>
```





FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000065321