Universidade do Porto
Faculdade de Engenharia
FEUP

Cristina Falcão Sousa Calheiros Santos

Metodologias de Desenvolvimento e Normalização Gráfica do Human Capital Team Tool na IBS Portugal

Anexo F: MethoDoc

IBS

Powerful software
Passionate people

International Business Systems

MethoDoc

ÍNDICE

1 Introdução

Este primeiro capítulo destina-se a clarificar qual o verdadeiro âmbito e objectivo(s) deste documento. É dada uma breve explicação sobre o que é um produto de *software standart* e finalmente é analisada a estrutura adoptada para os vários capítulos.

1.1 Objectivo

"O MethoDoc é um documento que permite a um novo elemento aprender tudo acerca do processo de desenvolvimento praticado no departamento de Java."

Esta foi a "visão" que deu origem ao MethoDoc. Durante uma fase inicial e depois de algumas reuniões, foi necessário reescrevê-la de modo a clarificar alguns pontos (p.e. o âmbito). Actualmente esta é a directriz fundamental do documento:

"O MethoDoc é um documento, focado no projecto HCTT, que materializa de forma organizada todas as regras e *know-how* da equipa, para cada fase do desenvolvimento."

1.2 Âmbito / Enquadramento

Numa primeira fase existiram dificuldades relativamente a definição das fronteiras deste documento. Deveria ser um "meta-manual" aplicável a todos os projectos da equipa de Java? Focar-se-ia apenas na fase de implementação? Há necessidade de um documento deste tipo? Que vantagens objectivas vai trazer?

O consenso a que se chegou está expresso na frase apresentada anteriormente. Este documento é "focado no HCTT", ou seja, destina-se a dar suporte ao projecto HCTT em concreto. No entanto, não será de estranhar que no futuro venha a ser útil à escrita de outros manuais semelhantes destinados a outros projectos.

Quanto à necessidade deste documento, esta deve-se essencialmente a dois factos:

- Os elementos que compõem a equipa de desenvolvimento podem mudar e consequentemente é necessário reter o conhecimento que cada indivíduo foi adquirindo.

- Ao longo das fases de desenvolvimento são estabelecidas regras das mais variadas espécies (interfaces, especificações, código, qualidade, etc). É vital que estas regras sejam **conhecidas**, **compreendidas**, **aceites** e **praticadas** por todos os elementos da equipa.

Num projecto de média/grande dimensão como é que sem a certeza de que **há regras** e estas **são cumpridas**, se pode afirmar que o processo de desenvolvimento é eficiente e o resultado tem qualidade? Se este documento tiver sucesso na sua missão, permitirá aproximar a equipa dos níveis de eficiência e qualidade necessários e que todos desejam.

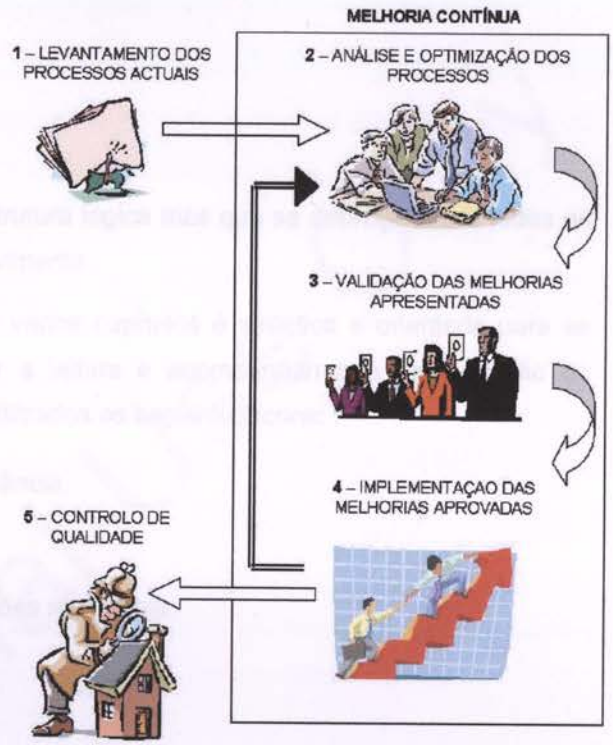
Por outro lado, continuará a ser fiel à primeira visão. Irá permitir uma rápida integração de elementos na equipa, já que toda a informação vital estará sintetizada e organizada, dando todas as orientações necessárias para que possam trabalhar de forma produtiva e coerente com os restantes colaboradores.

1.3 Ciclo de vida do *MethoDoc*

Este é um documento "vivo". Para que atinja os seus objectivos, tem que estar **sempre** de acordo com a realidade praticada no(s) departamento(s). Isto só pode ser conseguido se todos os elementos contribuírem no Levantamento dos processos actuais.

Podemos dividir este ciclo em duas partes distintas:

- Levantamento dos processos
- Melhoria contínua



O **levantamento dos processos**, consiste em "alimentar" permanentemente o MethoDoc com todas as regras/*know-how* existentes (que ainda não tenham sido escritos) ou que tenham sido criados/alterados.

Numa fase inicial este levantamento (como acarreta a sintetização de muita informação) poderá ser atribuído a 2 ou 3 pessoas, mas numa fase estável do documento é da **responsabilidade de todos** manter o documento de acordo com a realidade.

A **melhoria contínua** é igualmente uma tarefa que deve ser partilhada por todos, mas na qual os elementos associados à gestão dos departamentos envolvidos e da equipa têm um papel principal a desempenhar:

- Possuem uma visão mais abrangente do projecto.
- Detêm o poder efectivo para fazer com que as regras se cumpram.

Sobre o reflexo da realidade, "captado" no MethoDoc, é feita uma análise crítica com vista a apresentar sugestões de melhoria às regras praticadas. Estas sugestões, antes de serem implementadas, têm que ser previamente discutidas e validadas, se for o caso, pela gestão. Finalmente, o **controlo de qualidade** vem permitir ter a certeza sobre se as alterações estão ou não a ser efectivamente cumpridas.

1.4 Estrutura do MethoDoc

Optou-se por utilizar uma divisão em capítulos de acordo com os princípios da Engenharia de Software. O ciclo de vida de um produto de Software é composto por várias fases e genericamente identificam-se as seguintes:

- Levantamento de Requisitos/Especificação (Cap. 2)
- Desenvolvimento (Cap. 3 e 5)
- Validação/Testes (Cap. 4 e 5)
- *Deployment*/Manutenção (Cap. 6)

Assim sendo, o documento segue uma estrutura lógica mas que se debruça sobre todos os aspectos importantes do ciclo de desenvolvimento.

A informação que se pode encontrar nos vários capítulos é sintética e orientada para as regras e conhecimento útil. Para facilitar a leitura e acompanhamento da evolução do MethoDoc e aumentar a sua eficácia são utilizados os seguintes icons:

⚠ - Tópico de extrema importância.

NEW - Tópico recente.

UPDATED - Tópico que sofreu alterações relevantes.

Boa leitura...

1.5 Produtos de *software*

Quando se fala de produtos de software normalmente faz-se uma divisão em duas categorias principais:

- **Produtos genéricos** – São direccionados a um mercado potencial e procuram dar resposta a um conjunto de necessidades comuns a um leque de clientes.

- **Produtos específicos** – São desenvolvidos de acordo com as necessidades de um cliente tendo por base as suas ideias e especificidades.

O HCTT (Human Capital Team Tool) é um produto que pertence à categoria dos "genéricos" ou "*off-the-shelf*". De seguida, é dada uma visão dos primeiros passos, na IBS, de um projecto deste tipo.

1.5.1 Abordagem macro

Um produto genérico não nasce a partir da vontade de um cliente em especial, mas sim da observação das necessidades (actuais ou previsíveis) de todo um mercado potencial. Depois de surgir a ideia do produto é necessário aferir se terá mercado e posteriormente se é viável o seu desenvolvimento ou não.

1.5.1.1 Estudo de Mercado e Enquadramento

O objectivo do estudo de mercado é obter *feedback* sobre o interesse das empresas no produto. Normalmente isto é feito através de contactos com clientes potenciais, pertencentes à base instalada. Estes contactos permitem verificar se o produto suscita interesse, quais as alturas em que o mercado estará mais receptivo e através das opiniões dos clientes é possível enquadrar a ideia inicial para o produto no mercado real.

1.5.1.2 Estudo de Viabilidade

Se os resultados do passo anterior forem positivos, então há motivos para se avançar e o passo que se segue é determinar a viabilidade do projecto em várias directrizes, das quais se destaca a financeira (p.e. através de uma análise de retorno de investimento).

Através deste estudo é possível oficializar o projecto de desenvolvimento do produto, construindo uma equipa e elaborando um planeamento e respectivo orçamento.

1.5.2 Abordagem Tecnológica / Funcional (Alto-nível)

Esta é uma fase mais distribuída no tempo e não necessariamente estanque. Existe uma equipa internacional de impacto e estudo de questões tecnológicas. É esta equipa que decide quais as tecnologias e ferramentas que serão utilizadas no desenvolvimento do produto e isso garante entre outras coisas uma coerência em todos os projectos.

Relativamente à definição funcional, esta é definida em linhas gerais inicialmente e depois irá sofrer iterações ao longo do projecto. De modo a garantir que o projecto não se afaste da realidade, normalmente são aplicadas duas estratégias:

- Envolver um cliente real no processo de desenvolvimento.

- Formar um grupo de pressão (*Area Managers* e *Marketing*) que acompanhe a evolução do projecto a alto-nível.

2 Análise/Especificação

A análise e especificação é parte integrante de uma área muito importante da Engenharia de *Software*: a Engenharia de Requisitos. A Engenharia de Requisitos consiste no processo de captar e sintetizar de forma organizada as funcionalidades que o cliente pretende num sistema, bem como as restrições sob as quais esse sistema opera e é desenvolvido.

Um requisito pode ter um detalhe variado, desde uma frase abstracta de alto-nível até uma especificação funcional matemática.

Os tipos de requisitos são:

- Requisitos do Utilizador

  Frases em linguagem natural, acompanhadas de diagramas dos serviços que o sistema fornece e suas restrições operacionais. São escritos para os clientes.

- Requisitos do Sistema

  Um documento estruturado que define descrições detalhadas dos serviços do sistema. É escrito como um contracto entre o cliente e o fornecedor do sistema.

- Especificação do *software*

  Uma descrição detalhada do *software* que pode servir como uma base para um *design* ou implementação. É escrito para programadores.

No Departamento de Java elabora-se este último tipo de especificação: a especificação do *software*, orientada para os programadores do *Human Capital Team Tool*.

2.1 Documentação

A especificação é suportada por três documentos: um documento em *PowerPoint*, outro em *Word* (estes documentos estão interligados através de *links*) e finalmente um documento em *UML* contendo o modelo relacional da base de dados que irá suportar a(s) nova(s) funcionalidade(s). Estes documentos estão no CVS, no projecto "HCTTDOCS.

Neste capítulo serão desenvolvidos os processos relacionados com a definição do *lauyout* das interfaces e das tabelas da base de dados (que dão origem ao documento *PowerPoint* e *UML*, respectivamente). Os detalhes sobre os processos que originam o documento *Word* dizem respeito ao capítulo 3 "Desenvolvimento".

2.1.1 Documento *UML*

Contém o modelo relacional da base de dados que suportará os requisitos / funcionalidades especificados e deverá ser designado por **DATABASE<resumoDoAssunto>.MDL.**

Estes documentos estão **sempre** de acordo com o estado real da BD. Assim sendo, as alterações que devem ser efectuadas a este documento devem ser comunicadas aos analistas pelos programadores, pois são os analistas quem executam alterações as alterações.

Neste documento existem vários items que têm que ser especificados:

- Entidades
- *Nomes*: inclui o nome lógico e físico da tabela no AS400
- *Descrição*: inclui a maneira como acções sobre outras tabelas podem influenciar esta tabela e vice-versa
- *Relação*: inclui as chaves e as restricções
- Campos
- Nome
- Descrição
- Valores por defeito e sua descrição
- Valores possíveis e sua descrição

2.1.2 Documento *PowerPoint*

O interface gráfico estará todo definido num documento *PowerPoint*, que deve ser gravado com a seguinte designação: **SPEC_AAAAMMDD_AUX01_INTERFACEGRAFICO.PPT**, onde "AAAA" deve ser substituído pelo ano, "MM" pelo mês e "DD" pelo dia em que o documento foi concluído.

Existe uma página por defeito que pode ser usada como ponto de partida para construir todas as outras páginas. Podem ser encontrados vários exemplos de documentos *Powerpoints* de definição de interface gráfico no projecto "HCCTDOCS" do repositório CVS (já referido anteriormente). Este documento é da responsabilidade dos analistas.

Uma página deve ter :

- o nome dado pelo programador (esta actualização ao documento é feita posteriormente)
- toda a descrição visual para a sua construção (de acordo com as regras gráficas estabelecidas)
- todos os comandos que serão necessários para garantir a funcionalidade da página
- os *links* para as páginas acessíveis a partir dela.

2.1.3 Documento *Word*

Este é o documento de especificação principal e deverá ser designado por **SPEC_AAAAMMDD_00MAIN_DESCRIÇÃOSUMÁRIADOASSUNTO.DOC** onde "AAAA" deve ser substituído pelo ano, "MM" pelo mês e "DD" pelo dia em que o documento foi concluído.

Podem ser encontrados vários exemplos de documentos *Word* de especificação no projecto "HCCTDOCS" do repositório CVS (já referido anteriormente). Este documento é da responsabilidade dos analistas. O documento deve conter :

- **objectivo** : descrição sumária da funcionalidade que esta especificação documenta.
- designação do **documento auxiliar** a esta especificação (o documento *Power Point*)
- **descrição** de cada uma das **funcionalidades** implementadas por esta especificação.

  Para cada funcionalidade deverá ser incluído o *link* para o slide respectivo do documento *PowerPoint* que implementa esta página e a descrição das tarefas a executar pelos comandos usados, nomeadamente todas as alterações necessárias à base de dados.
- lista de comandos / páginas.

A definição dos comandos a implementar é da responsabilidade do programador, ou seja, será este a decidir quais os comandos necessários para a execução das funcionalidades pretendidas. Deverão ser tomados alguns cuidados na nomenclatura dos comandos, bem como na tomada de decisão sobre a qual *package* deve o comando pertencer. Para um melhor esclarecimento das regras a utilizar deverá ser consultado o capítulo 3.

2.2 Esquema relacional da BD

2.2.1 Entradas / Saídas

Para elaborar novas partes do esquema relacional da BD é necessário ser efectuada uma reunião com os responsáveis por decidir quais as novas funcionalidades a serem disponibilizadas no *Human Capital Team Tool*, Ernesto Nogueira, Bárbara Costa e Gonçalo Mata. Esta reunião serve para tomar conhecimento das novas funcionalidades a serem implementadas (incluindo a forma como será efectuada a interacção com o utilizador, que tipo de informação é necessária armazenar, manipular, visualizar, etc).

O resultado a produzir é o documento MDL enunciado anteriormente, que deverá ser escrito pelo programador caso os analistas não o tenham feito.

2.2.2 Ferramentas utilizadas

Para o desenho da arquitectura da base de dados é utilizado o *plugin Omondo* do Eclipse.

Para a implementação física do esquema é utilizada a ferramenta *Persistence Builder.* No capítulo 3 existe uma secção com o nome *Persistence Builder* onde poderá ser encontrado um guião de como utilizar esta ferramenta.

2.2.3 Definição de Tabelas

Cada nova tabela da base de dados deve conter dois campos adicionais, para além daqueles que são definidos como necessários. Esta estratégia é necessária para evitar modificações morosas nas tabelas quando são necessários novos campos nessas tabelas à medida que a aplicação evolui.

Os campos extra adicionados por defeito são um numérico (tipo DECIMAL) de (10,0) decimais e um alfanumérico (tipo VARCHAR) com 50 caracteres de comprimento máximo. De modo a terem uma nomenclatura comum na aplicação estes campos são sempre designados por FIELDNUM e FIELDALPHA respectivamente.

2.3 *Layouts* das interfaces

2.3.1 Entradas / Saídas

À semelhança do ponto anterior, é também necessário ser efectuada uma reunião com os responsáveis pelo projecto do *Human Capital Team Tool* de modo a projectar o *layout* das interfaces. De acordo com as novas funcionalidades (incluindo a forma como será efectuada a interacção com o utilizador, que tipo de informação é necessária armazenar, manipular, visualizar, etc) deverá ser planeada, nessa reunião, a interface que melhor suporte as funcionalidades pretendidas.

2.3.2 Ferramentas utilizadas

Para a definição do *layout* das interfaces é utilizado o *PowerPoint.*

Para a implementação das interfaces é utilizada a *framework UnilFace.* Através da ferramenta *GuiGen* o programador pode gerar os ficheiros JSP automaticamente a partir de ficheiros descritivos XML, chamados processos. No capítulo 3 existe uma secção com o nome *UnilFace - GuiGen* onde poderá ser encontrado um guião de como utilizar esta ferramenta.

2.3.3 Estrutura geral de uma página

Uma página "normal" é sempre apresentada numa janela do *browser.* Assim, os elementos visuais constituintes da página têm que se aproximar, o mais possível, aos elementos que, normalmente, encontramos numa aplicação para *Windows*, de modo a que o utilizador se sinta familiarizado. Uma página (normal) pode ser dividida em dois grupos, conforme o tipo de informação a apresentar: **listas** e **detalhe**.

A informação apresentada numa página deve ser, sempre que possível, o mais simples e clara possível. Não é aconselhável aproveitar todo o espaço em branco disponível senão as páginas tornam-se pesadas, "maçudas", complicadas para o utilizador e possuidoras de baixa *performance* tanto para o servidor como para o cliente.

Os espaços em branco são muito importantes na filosofia da interacção Homem←→Máquina. O visual das páginas deve ser atraente, contendo, sempre que

possível, elementos visuais de impacto médio (imagens, cores, etc...), mas que não causem uma sensação de desconforto por parte do utilizador.

2.3.4 Elementos de uma página

De seguida são descritos e explicados os vários elementos constituintes de uma página.

2.3.4.1 Títulos

Cada página tem, pelo menos, uma *form*. É na *form* que está situada toda a informação e é nela que essa informação é manipulada. Toda a *form* deve ter um título que identifique claramente que objecto, ou tipo de objecto é que está a ser apresentado e manipulado pelo utilizador. Qualquer informação adicional deve ser colocada no canto superior direito da página. Essa informação deve expressar ou auxiliar o utilizador acerca da acção que pretende realizar.

De seguida são apresentadas, mais detalhadamente, as regras para os títulos de uma página:

1. Numa lista, coloca-se o nome da entidade, p.e.Cursos.

2. Na descrição de uma entidade, coloca-se o nome da entidade, p.e. Curso.

3. No canto superior direito coloca-se, normalmente, a acção a executar, p.e. Criar, Editar, Anular, Adicionar responsáveis, etc. Esta regra não se aplica quando a página é de visualização de dados (por exemplo, dados de um nó, dados de uma lista, etc). Nesse caso não deve existir um título no canto superior direito.

2.3.4.2 Etiquetas para os títulos das forms nos Bundles

Os *bundles* contendo o par "etiqueta-tradução" para o título das *forms* são identificados, no XML, através do atributo `bundle` da tag `<title>`.

O *bundle* de uma página deve ser definido conforme o seu tipo. De modo a uniformizar e estruturar a criação de páginas, todos os *bundles* devem ter as etiquetas como substantivos

sempre na sua forma singular (ex: "idHumanResourceTitle" – esta página apresenta a informação relativa a um único HR.

Se for pretendido apresentar uma lista de HRs, a etiqueta presente no *bundle* da página será "idHumanResourceListTitle"). De notar que a diferença entre ambas reside no facto de a segunda apresentar a palavra "List". É esta a regra que deve ser aplicada sempre na construção de páginas.

Se o utilizador estiver a ver uma página relativa a um HR, quer seja para criar, editar, apagar, ou efectuar uma outra acção, a etiqueta do título no *bundle* de todas elas deve ser a mesma. O HR continua o mesmo, assim como todos os campos e respectiva informação. O que pode mudar é a informação adicional, que deve ser colocada no campo superior direito e que é dependente do nome da página especificada no XML.

2.3.4.3 Acções

Todas as opções disponíveis na aplicação estão representadas através de elementos visuais que permitem uma interacção entre a aplicação e o utilizador. **As acções devem, sempre que possível, ter como nomenclatura verbos na sua forma infinitiva** (criar, anular, importar, visualizar, sair, etc.), de modo a haver uma correspondência entre um desejo particular do utilizador (mudar de página, operar sobre uma *form*, etc.) e as funcionalidades disponíveis.

Essas opções estão distribuídas em três tipos de elementos: menús, botões e campos, apresentados a seguir.

Menús

A barra de menús é um elemento visual que é facilmente reconhecível e de fácil utilização por parte de qualquer utilizador que tenha conhecimentos de informática. Trata-se de uma barra situada, normalmente, no topo da área de visualização de uma página, e é constituída por palavras. Essas palavras são botões que servem para identificar grupos de opções (apresentadas numa barra vertical "deslizante" que é activada quando se selecciona o título do grupo).

Todas as opções dos menús devem ser acessíveis através de teclas de atalho (o *Human Capital Team Tool* ainda não permite essa funcionalidade, embora o tema esteja a ser estudado), conforme qualquer regra de GUI determine.

A barra de menús deve disponibilizar acções "gerais", ou seja, acções que não tenham directamente a ver com a página apresentada (ex. *Import* dos perfis: essa opção está disponível num menú porque não é possível fazer um *import* apenas de um perfil técnico, somente dos três conjuntos de *skills*).

Todas as acções disponíveis nos *bundles* para os menús devem conter, no par "etiqueta-tradução", a etiqueta "idM", de modo a se poder identificar a sua utilização (p.e. idMExit = Sair).

Uma página deve conter pelo menos um elemento que identifique univocamente o produto (neste caso, *Human Capital Team Tool)*, assim como a empresa que o criou (IBS). Geralmente, estes elementos visuais estão situados na barra de menus.

De seguida são apresentadas mais algumas regras para os menús:

1. No menu deve existir, sempre que necessário, como um ponto de menu, o nome da entidade e depois a opção de Sair. (Exemplo : Base de conhecimento -> Sair).

2. O Sair deve ser usado quando se sai de uma entidade onde se podem executar várias acções, como sair de um módulo, de um sub-módulo, da base de conhecimento, dos templates de acção, etc.

3. O Voltar deve ser usado quando se sai de uma única acção, como por exemplo, de uma visualização.

Botões

Os botões representam as acções que se podem realizar numa determinada *form*. Graficamente, são divididos em dois grupos distintos: botões "**normais**" e botões "***context-menu***".

Os botões "**normais**" são apresentados sob a forma de círculos contendo, cada um, um logótipo que ajude, visualmente, o utilizador a reconhecer e a perceber melhor a acção e o seu objectivo. O nome da acção especificada no XML deve ser igual ao nome do ficheiro de imagem ao qual a queremos associar. Deve conter o prefixo "action_". Existem nomes de acções específicos para determinar o comportamento da aplicação conforme a situação:

- **"_enable"**

  Se a acção conter este sufixo, o botão fica activo apenas se todos os campos obrigatórios estiverem correctamente preenchidos (ex: action_confirm_enable – neste caso, será apresentada uma imagem com um visto (indicado pelo sufixo "_confirm" e que só estará disponível se todos os campos de preenchimento obrigatório estiverem correctos).

- **"_now"**

  Se a acção conter este sufixo, quando o utilizador a seleccionar, a aplicação automaticamente fecha a janela actual, evitando a necessidade de se colocar código javascript no XML ou nalgum dos ficheiros JS que são utilizados.

Os botões "***context-menu***" são utilizados em *forms* que contenham listas de dados. Servem para o utilizador operar sobre uma determinada linha dessa lista. Para as invocar basta carregar com o botão direito do rato em cima da linha desejada. Imediatamente será apresentado um menú contendo todas as opções disponíveis.

As etiquetas dos *bundles* das acções também servem para determinar qual o seu aspecto. Assim, existem duas palavras "reservadas" que devem ser sempre utilizadas quando se justifica:

- **"Bold"**

  Esta palavra, se colocada como sufixo no *bundle* da acção, serve para o programador indicar que a tradução da acção será apresentada em "negrito". Tal como no Windows (e qualquer outra aplicação que respeite as normas de GUI), **qualquer acção que esteja representada a "negrito" deve ser a primeira** (ou única) **da lista de acções disponíveis em *context-menu* e serve para informar que se o utilizador fizer *double-click* sobre uma linha da lista, irá despoletar esta acção.**

- **"Underline"**

  Esta palavra, se colocada como sufixo no bundle da acção, serve para o programados indicar qual a letra da tradução que será sublinhada. Apesar de a aplicação ainda não fornecer teclas de atalho, convém referir que esta funcionalidade é muito importante para o utilizador experiente pois serve para criar uma interface Homem-Máquina bastante simples, além de já ser um *standard* nas aplicações que utilizem um GUI. Assim, se o sufixo estiver seguido de um número, esse número representa a letra da tradução que terá o sublinhado. Se não apresentar qualquer número, será a primeira letra da palavra traduzida a ter o sublinhado.

Convém referir que **todas as etiquetas** dos *bundles* das acções situadas em *context-menu* deverão ter o sufixo "Underline" e que, em cada *form*, apenas uma deverá ter o sufixo "bold". Estes sufixos **jamais** serão utilizados nas etiquetas dos *bundles* de acções "normais".

Campos

A aplicação *Human Capital Team Tool* permite que os tipos dos campos dos formulários sejam praticamente os mesmos fornecidos pelo HTML. Na aplicação existem campos de texto simples, *combo-boxes*, *checkbuttons*, imagens, *radio-buttons*, etc.

Se determinado campo for utilizado em mais que uma página, a sua tradução deve ser **sempre** a mesma. Além disso, se um mesmo objecto permite que a sua representação seja feita através de vários tipos de campos, ou mesmo campos diferentes, deve utilizar-se sempre o mesmo tipo de campo com o mesmo *bundle* (ex: nas listas de HRs, é comum apresentar o nome do HR de maneiras diferentes: nome próprio, primeiro e último nome, nome completo, "conhecido por", etc... As traduções dos campos nos *bundles* devem ser o mais curtas possíveis de maneira a que a informação apresentada por cada página não seja muito "pesada" para o utilizador).

2.3.5 Navegação

Em qualquer página do HCTT existem, como já foi referido anteriormente, botões "**normais**" e botões "***context-menu***". Qualquer um destes botões provoca a execução de uma acção que, normalmente, irá implicar o aparecimento de uma nova página ao utilizador. Esta página deverá **sempre** ser aberta numa nova janela (de dimensões menores à janela de onde foi originada a acção) e não na mesma janela. Esta nova janela deverá conter um botão "Voltar" que irá provocar o encerramento desta e a actualização (*refresh*) da janela anterior.

2.3.6 Design

Cada módulo do *Human Capital Team Tool* é identificado por uma cor, predominante em todo o módulo.

Cada elemento de uma página de um módulo(menús, listas, botões, etc) têm que obedecer à cor determinada para esse módulo.

Os elementos gráficos da aplicação são elaborados em regime de *outsourcing* pela empresa **Tecninet**, cujas instalações estão situadas dentro do espaço da IBS Ibéria, em Vila Nova de Gaia. Sempre que for necessário um elemento gráfico novo para um qualquer módulo da aplicação deve ser enviado um *e-mail* ao Gonçalo Mata requerendo esse novo elemento gráfico e explicitando em que consiste. O Gonçalo Mata será responsável por transmitir a informação necessária à Tecninet e, posteriormente, por enviar os elementos gráficos novos à pessoa que os requereu. Esta pessoa ficará, depois, responsável por inserir estes novos elementos gráficos na aplicação e disponibilizá-los para os restantes elementos da equipa de desenvolvimento.

2.4 Validação

Todos os documentos produzidos durante a fase de análise e especificação terão que ser enviados para o Gonçalo Mata (*goncalo.mata@ibs.pt*) de forma a serem validados.

3 Desenvolvimento

Todas as etapas do ciclo de vida de um produto de software são importantes, mas é nesta que os produtos ganham "vida" e consequentemente todas os erros têm visibilidade quase imediata. Devido ao carácter prático desta fase está rodeada de várias regras e conhecimento útil que vai sendo produzido com o desenrolar do projecto.

São especificadas as normas que se devem seguir na codificação. A equipa de desenvolvimento é constituída por vários elementos que terão, idealmente, que programar como um só, no sentido de não prejudicarem o trabalho uns dos outros e de manterem uniformidade/coerência entre as várias partes. Um dos pontos fundamentais desta coerência é a compreensão e respeito pela Arquitectura da aplicação, já que esta encerra as estratégias para questões fundamentais ao sucesso do projecto, como escalabilidade, extensibilidade e eficiência.

Nesse sentido, serão abordadas questões sobre a Arquitectura do HCTT e normas para a codificação. A documentação, nas suas várias facetas (técnica e alto-nível), é outros dos tópicos fundamentais pois garante a transparência do processo o que leva a ganhos de qualidade e produtividade. Tarefas como alteração e revisão de código, que são uma componente muito importante no ciclo de desenvolvimento de *software,* ficam extremamente facilitadas.

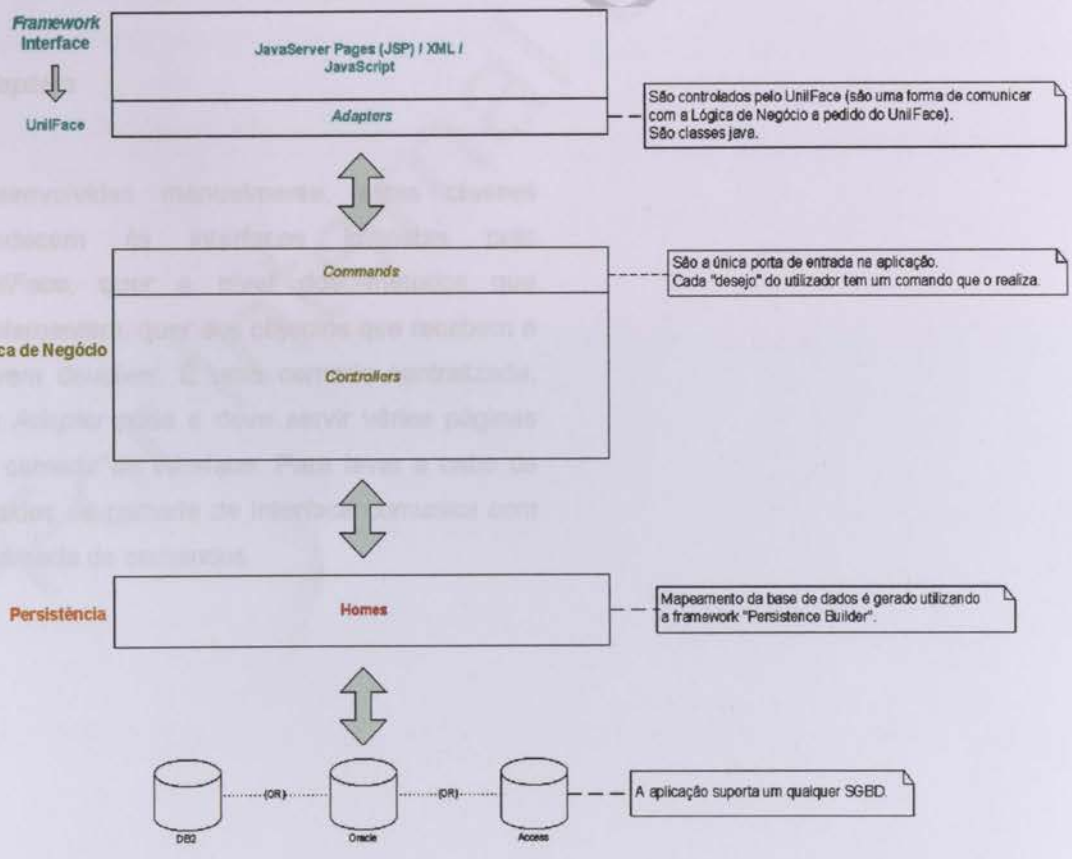
3.1 Arquitectura da Aplicação

Em aplicações de média/grande dimensão a Arquitectura é um tópico vital para o sucesso do projecto.

Em termos de software, para um determinado problema existirão sempre várias soluções. No entanto, como a Engenharia de Software veio demonstrar, há determinados esquemas/modelos/arquitecturas que não só resolvem os problemas como exploram simultaneamente questões de eficiência, robustez e escalabilidade. Ao permitirem ver o problema de uma forma mais alto-nível garantem uma uniformidade funcional e tecnológica da solução.

3.1.1 Descrição da Arquitectura

A arquitectura adoptada para o HCTT é uma arquitectura em camadas.



Idealmente, o interior de cada camada é totalmente isolado do exterior através de interfaces e apenas é permitida a interacção entre camadas adjacentes. Deste modo qualquer uma das

camadas pode ser totalmente substituída/remodelada desde que as interfaces sejam respeitadas. Aumentando um pouco a complexidade ganha-se a liberdade de poder mudar p.e. as interfaces da aplicação ou a Base de Dados.

Mas para que isto seja verdade é necessário que as regras, sobre as quais assenta esta Arquitectura sejam compreendidas e cumpridas com rigor.

3.1.2 Visão detalhada das Camadas

Interface

Esta camada é gerada a partir de uma *framework* da IBS, *UnilFace*[1]. Esta aplicação gera JSP's, a partir de processos XML. Um processo XML pode gerar uma ou mais páginas *Web* e cada página e respectivas acções são suportadas pela camada dos *Adapters*.

Adapters

Desenvolvidas manualmente, estas classes obedecem às interfaces impostas pelo *UnilFace*, quer a nível dos métodos que implementam, quer dos objectos que recebem e devem devolver. É uma camada centralizada, um *Adapter* pode e deve servir várias páginas da camada de interface. Para levar a cabo os pedidos da camada de interface comunica com a camada de comandos.

[1] Consultar os manuais do UnilFace para informação mais detalhada.

Comandos

Representam uma camada descentralizada que capta não só as acções dos utilizadores assim como os seus contextos. P.e. a mesma acção feita em páginas diferentes irá corresponder a dois comandos e não um. A acção é a mesma, mas os contextos não. Uma das aplicações desta camada é a segurança, permitindo controlar em detalhe todas as interacções a que os utilizadores têm acesso.

Controllers

Esta é a camada de lógica de negócio. Deve ser centralizada e orientada aos BO's. Isto é, os *Controllers* que forem implementados tendencialmente representarão um BO e lidarão com toda a lógica de negócio relativa. Para aceder à BD recorrem às classes da camada de persistência.

Persistência

Classes geradas pela *framework Persistence Builder* do *Visual Age*. Está encarregue de gerir todas as interacções com a BD. É uma camada complexa, à qual pertencem os BO que são utilizados nas restantes camadas.

Fronteiras das Camadas

De todas as camadas apresentadas, aquelas onde o programador tem que tomar decisões com impacto sobre a arquitectura são as seguintes:

- *Adapters*.
- Comandos.
- *Controllers*.

Embora muitas das restrições estejam implícitas no conceito de arquitectura em camadas e nas explicações já apresentadas iremos reiterar os pontos mais importantes.

Adapters

- As classes devem ser centralizadas. Se páginas diferentes pretendem aceder à mesma informação ou subconjuntos dessa informação deve apenas existir um *Adapter* que suporte as páginas e não vários.
- Os *Adapters* comunicam exclusivamente com os Comandos.
- Um método de um *Adapter* invoca apenas um comando já que este representa um pedido do utilizador através da interface.
- Não é implementada lógica nestas classes. Por vezes, o que poderá ser necessário é converter os resultados dos comandos para os objectos/estruturas de dados reconhecidas pelo *UnilFace*.

Comandos

- Existe um comando para cada acção possível na interface. Por isso, esta camada é descentralizada.
- Os comandos devem ser classes extremamente simples, sem lógica e que comunicam com os *Controllers* de modo a obter os dados ou executar as tarefas que precisam.
- Um comando pode invocar vários métodos de vários *Controllers* se tal for necessário para obter todas as informações que precisa ou executar todas as tarefas.

- Se existir lógica de negócio entre chamadas a métodos de *Controllers*, esta não deve ser implementada no comando mas passar para o *Controller*. Nesse caso, o comando passará a chamar um único método do *Controller* que por sua vez irá comunicar com os outros *Controllers* implementando a lógica necessária.

- Um comando não invoca outros comandos. Só em casos muito excepcionais é que isto acontece e em todo o caso deve ser discutido com os gestores do projecto.

Controllers

- Esta camada é centralizada pelo que, a existir, só deve ser um o *Controller* que representa um dado BO.

- O código deve ser centralizado mesmo a nível dos métodos, quer isto dizer que se deve evitar ter dois métodos a fazer o mesmo. Nestes casos, para decidir que *Controller* fica com o método deve-se escolher aquele que representar o BO a quem em termos lógicos se faria a pergunta.

- É nesta camada que é implementada toda a lógica de negócio.

- Um *Controller* pode e deve comunicar com outros *Controllers* sempre que necessitar de informações ou executar operações que estejam fora do seu "domínio".

- É nesta camada que se devem centrar os esforços de optimização de eficiência. Devido a ser a camada mais complexa e onde se situam, idealmente, os principais algoritmos, é aqui que optimizações a nível de JAVA e de acessos à BD têm especial impacto.

3.2 Frameworks

3.2.1 Persistence Builder

Nesta secção será apresentado um guião "passo-a-passo", com ilustrações, sobre como gerar as tabelas da base de dados e o respectivo mapeamento utilizando a *framework Persistence Builder*. Para este exemplo assume-me que a base de dados é DB2 e o servidor da base de dados é o **ibsd04**.

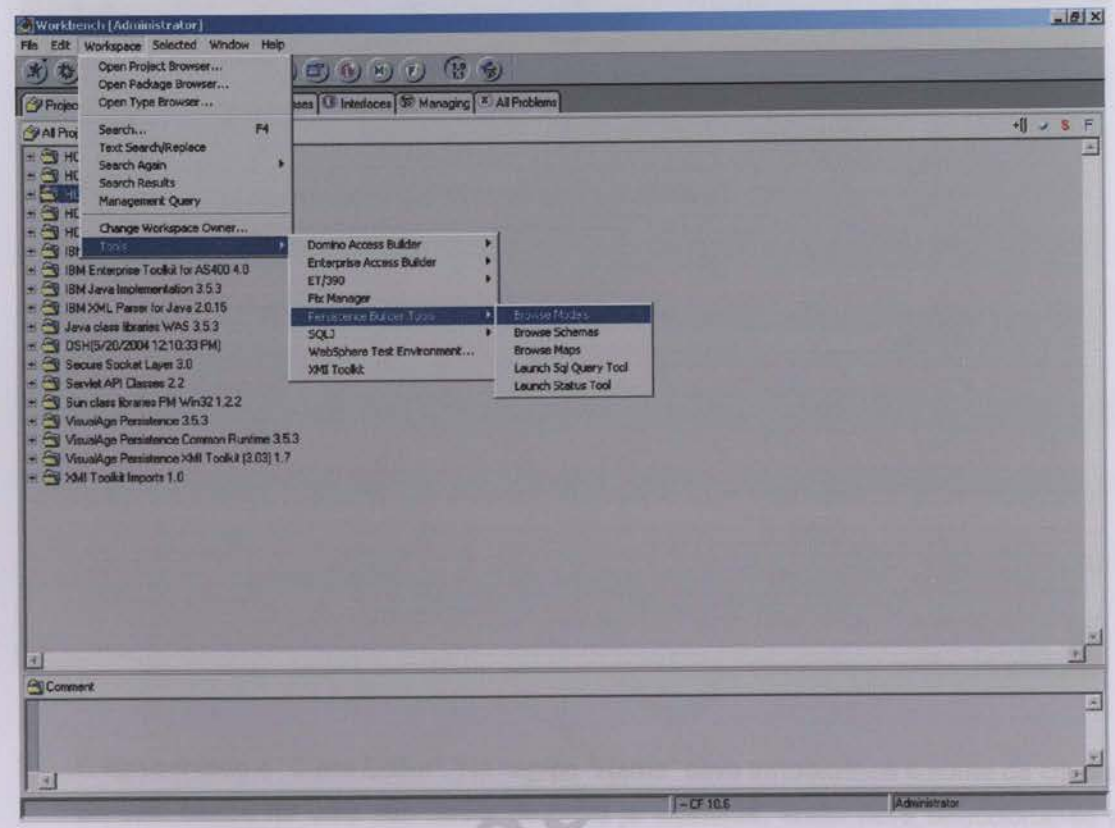
É através do IDE **Visual Age** que se acede à *framework*. Este IDE deve ser instalado no computador do programador e antes de começar a utilizar a *framework* propriamente dita, é necessário fazer *checkout* do CVS do Visual Age das seguintes *libraries*:

- IBM Enterprise Extension Libraries 3.5.3
- IBM Enterprise Toolkit for AS400 4.0
- IBM Java Implementation 3.5.3
- Java class libraries WAS 3.5.3
- Secure Socket Layer 3.0
- Servlet API Classes 2.2
- Sun class libraries PM Win32 1.2.2
- VisualAge Persistence 3.5.3
- VisualAge Persistence Common Runtime 3.5.3
- VisualAge Persistence XMI Toolkit (3.03) 1.7
- XMI Toolkit Imports 1.0

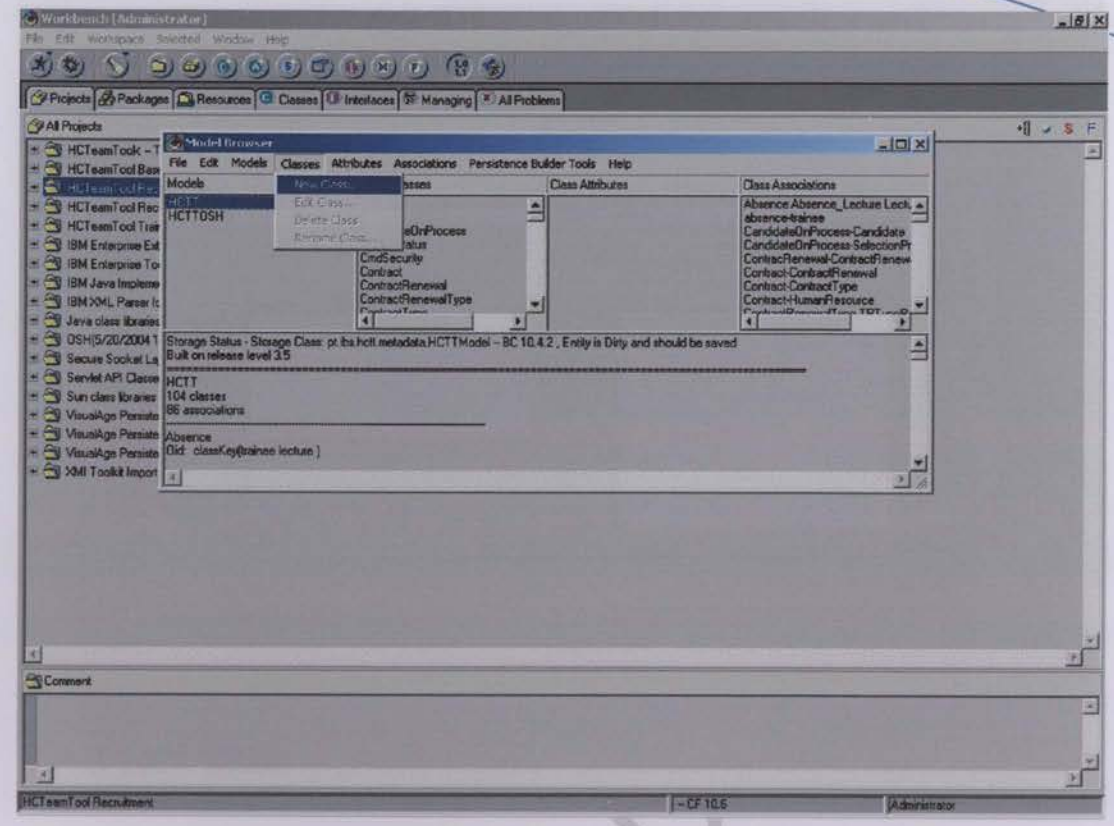
Como é óbvio, também é necessário fazer *checkout* dos projectos do HCTT e **deve tomar-se a atenção de fazer *checkout* sempre das últimas versões dos projectos.**

Os vários passos a seguir para a criação das tabelas da base de dados e do mapeamento desta são então:

1. Ir a ***Workspace -> Tools -> Persistence Builder Tools -> Browse Models***, tal como é indicado na figura a seguir.

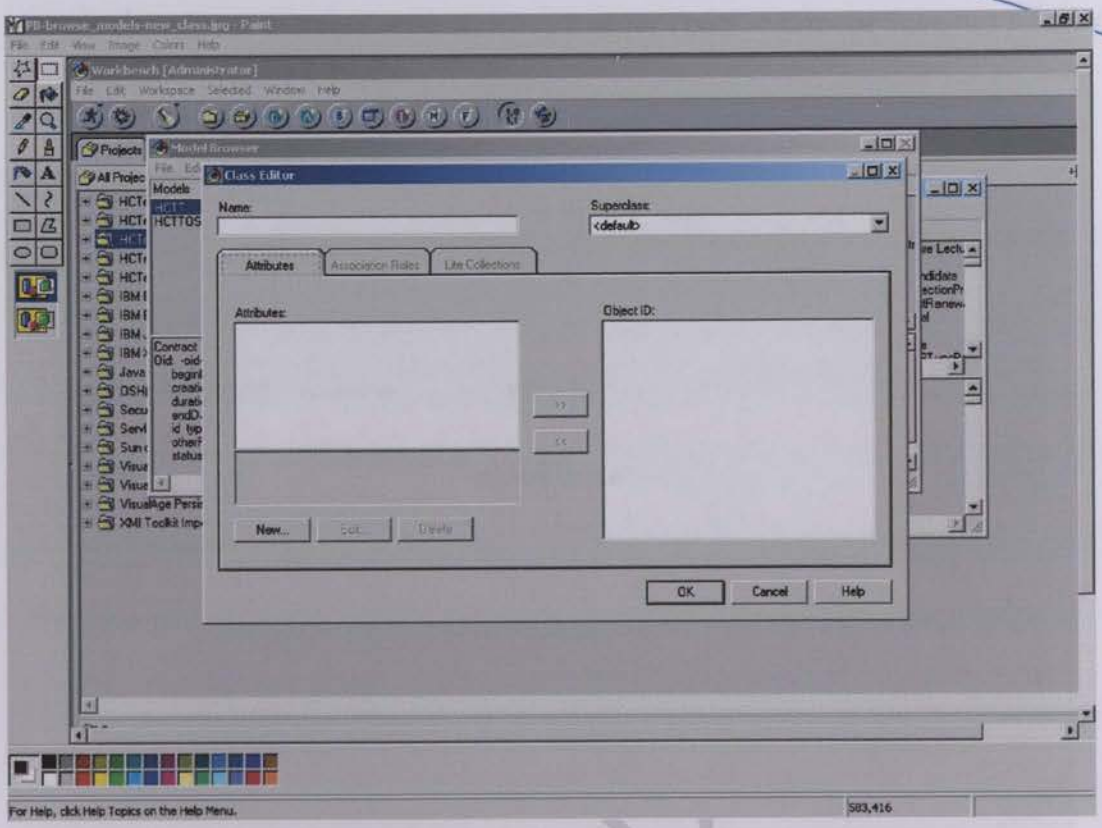


2. Seleccionar o *model* pretendido da coluna "Models" situada do lado esquerdo da janela "**Model Browser**". Ir a "***Classes -> New Class***" para criar uma nova classe (consultar figura seguinte).

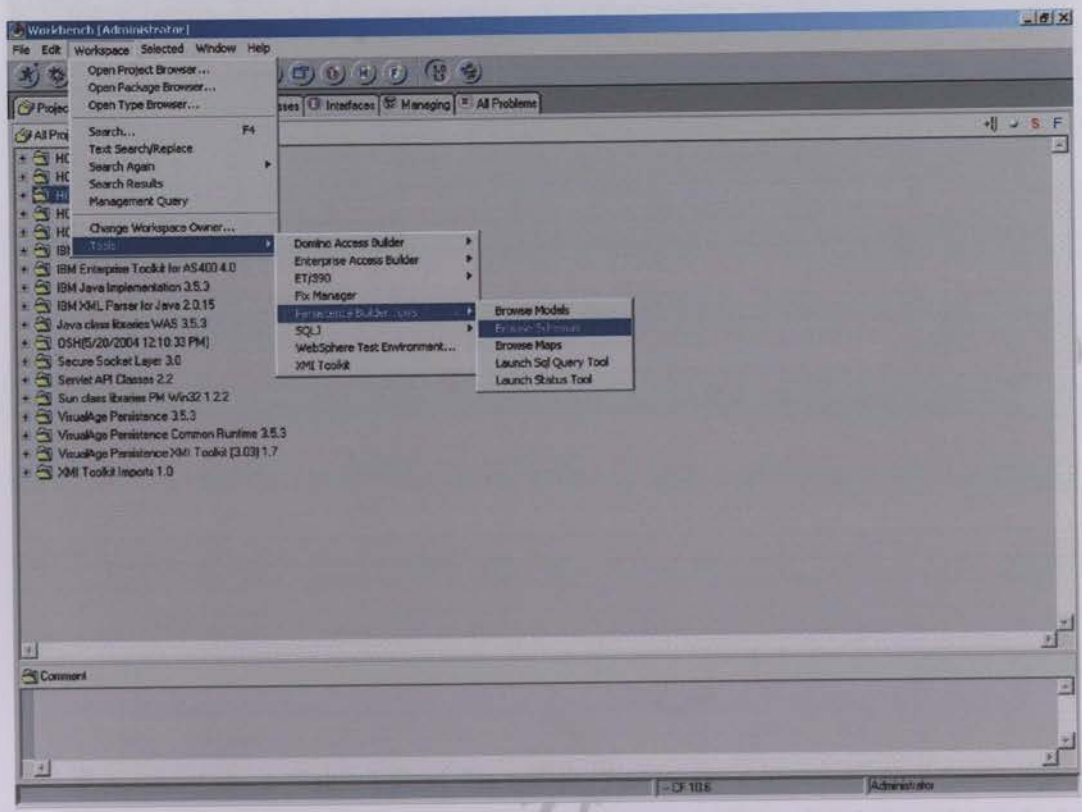


3. É apresentado o "Class Editor". No campo "Name" deve introduzir-se o nome da classe. Se esta classe for uma sub-classe de outra já existente, então deve seleccionar-se a superclasse da lista de classes apresentadas em "Superclass".

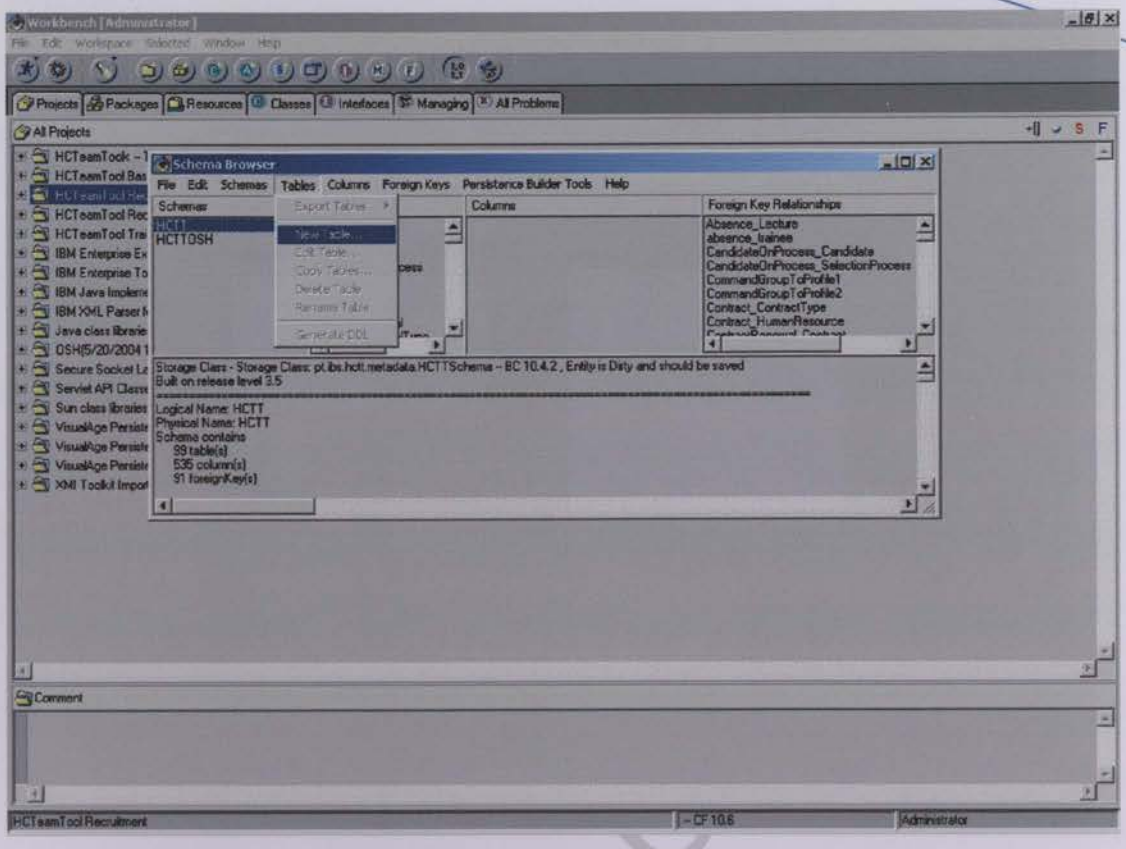
4. No separador "Attributes" irão aparecer listados os atributos da classe. Esses atributos têm que ser criados e, para isso, deve premir-se sobre o botão "**New**". Após a criação dos atributos, deve seleccionar-se da lista de "Attributes" aquele(s) que será (serão) o(s) ID(s) da classe e premir sobre o botão "**>>**" (ver figura seguinte).



5. Ir a ***Workspace -> Tools -> Persistence Builder Tools -> Browse Schemas*** tal como é indicado na figura a seguir.

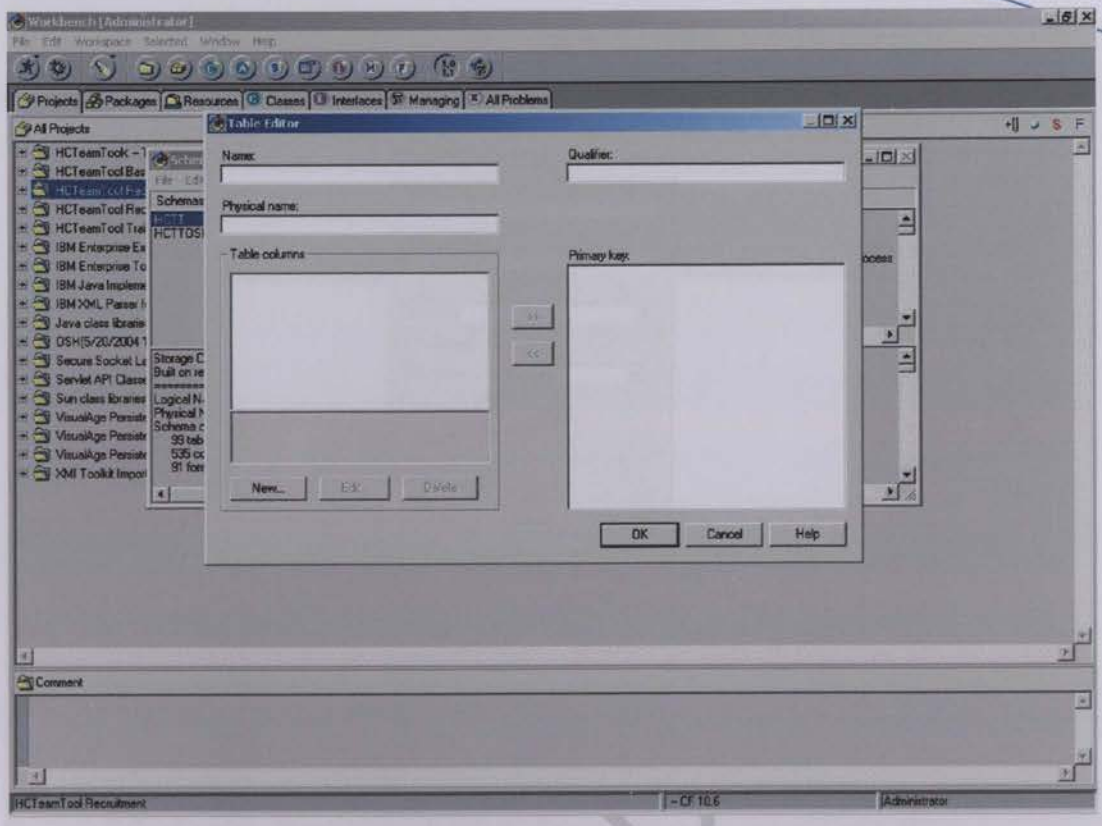


6. Seleccionar o *schema* pretendido da coluna "Schemas" situada do lado esquerdo da janela **"Schema Browser"**. Ir a ***"Tables -> New Table"*** para criar uma nova tabela (consultar figura seguinte).



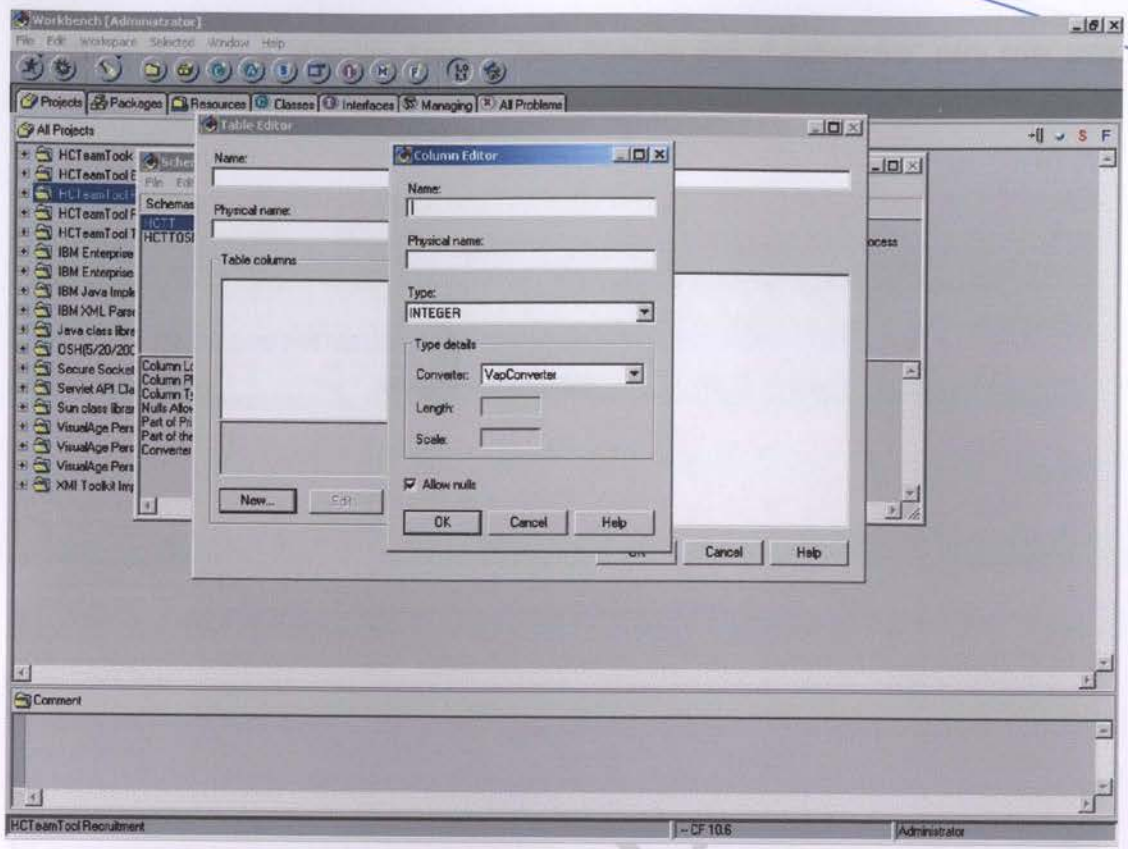
7. É apresentado o "Table Editor". No campo "Name" deve introduzir-se o nome lógico da tabela. No "Physical Name" deve introduzir-se o nome físico da tabela na base de dados e deve ter-se em atenção que **este nome só pode ser composto por 10 caracteres no máximo**.

8. No separador "Table Columns" irão aparecer listadas as colunas / campos da tabela. Essas colunas têm que ser criadas e, para isso, deve premir-se sobre o botão **"New"** (consultar figura a seguir).



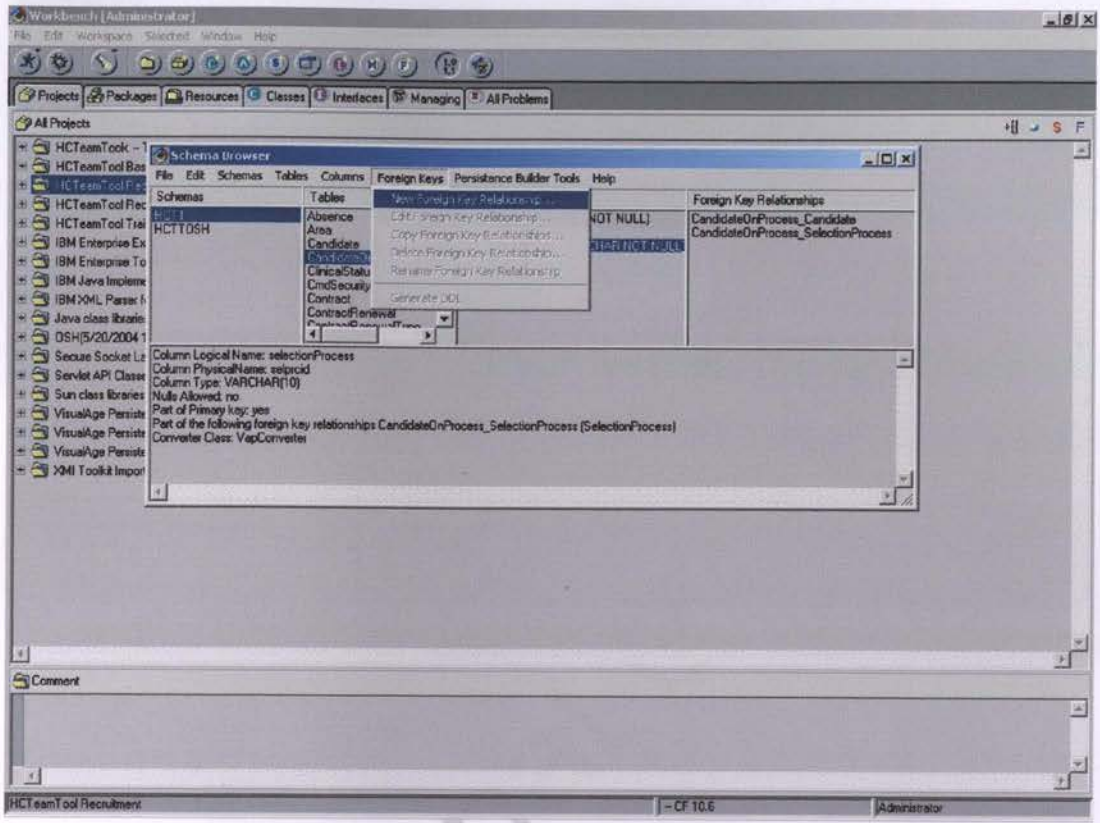
9. É apresentado o "Column Editor". No campo "Name" deve introduzir-se o nome lógico da coluna. No "Physical Name" deve introduzir-se o nome físico da coluna na tabela e deve ter-se em atenção que **este nome só pode ser composto por 10 caracteres no máximo**.

10. Em "Type" deve escolher-se o tipo do campo (decimal, varchar, etc) e deve ser colocado sempre "**Vap Converter**" em "Converter" nos "Type Details" (ver figura seguinte).



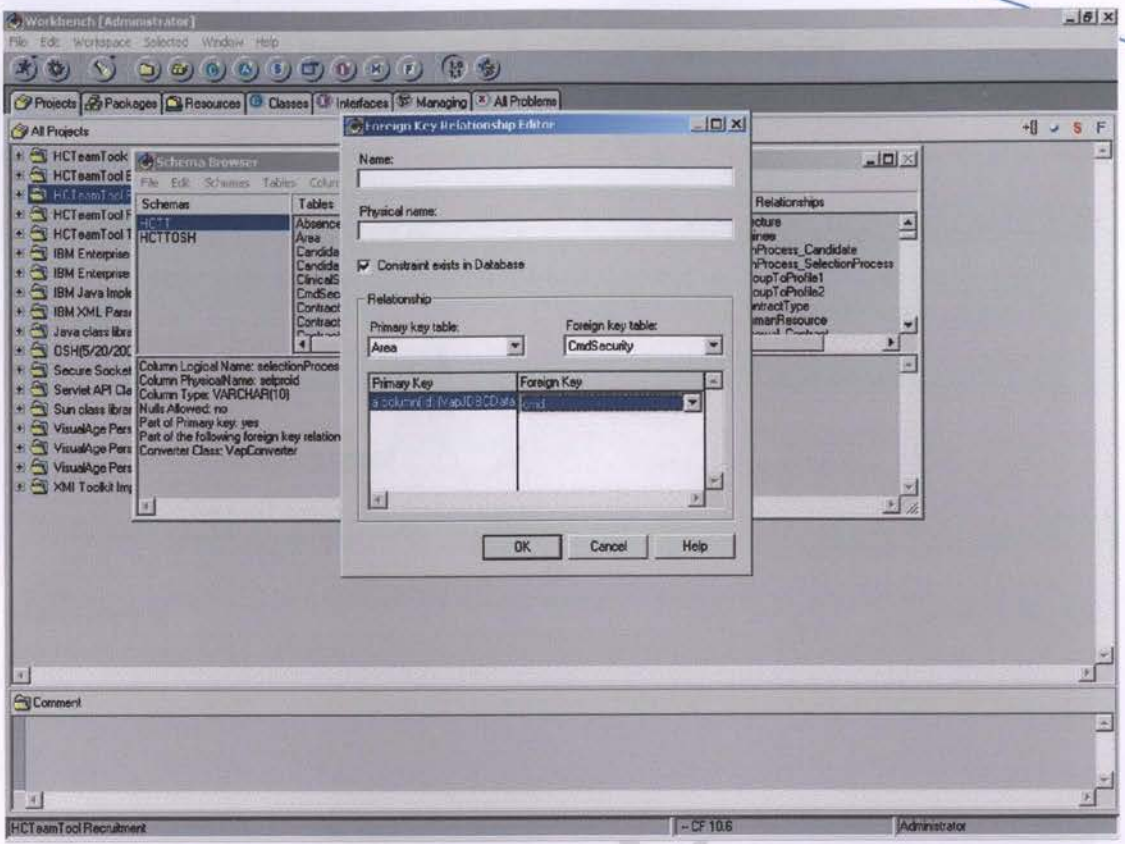
11. Após a criação das colunas / campos da tabela, retorna-se ao "Table Editor". Deve seleccionar-se da lista de "Table Columns" aquela(s) que fará (farão) parte da chave-primária da classe e premir sobre o botão ">>".

12. De seguida, Ir a "***Foreign Keys -> New Foreign Key Relationship***" para criar uma relação de chave-estrangeira (consultar figura seguinte).

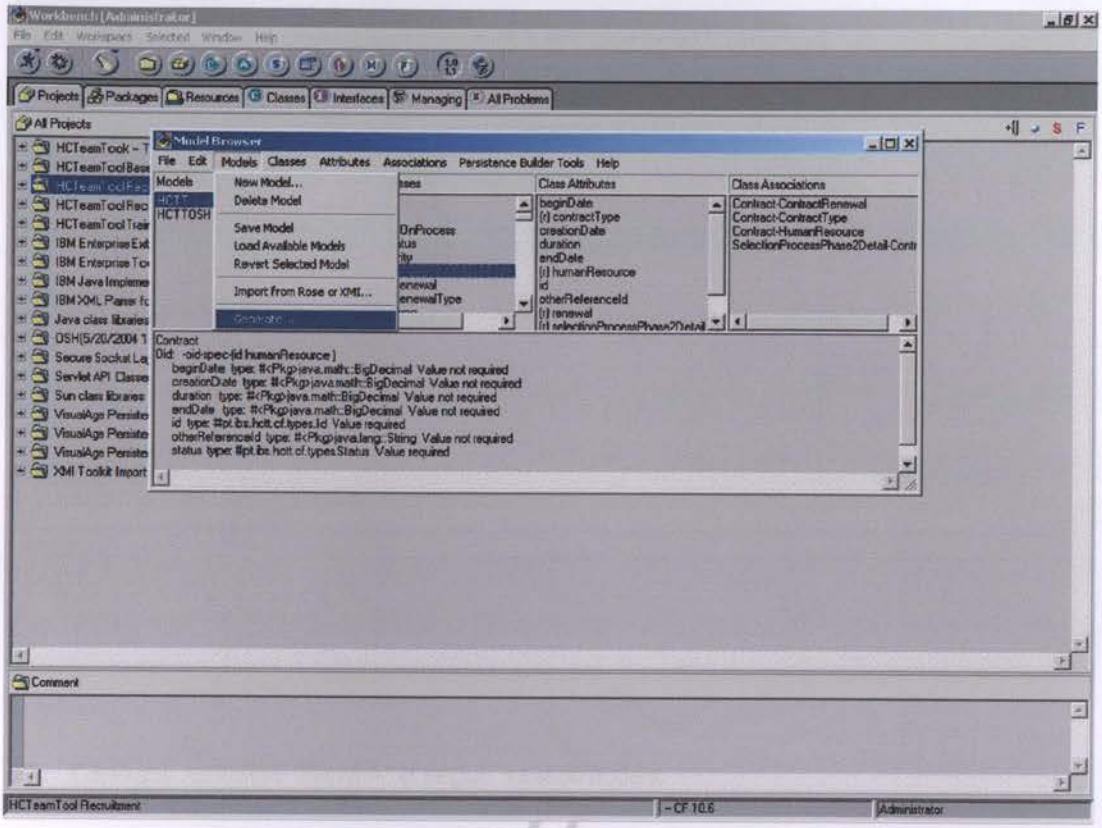


13. É apresentado o "Foreign Key Relationship Editor". No campo "Name" deve introduzir-se o nome lógico da relação. No "Physical Name" deve introduzir-se o nome físico da relação na base de dados e deve ter-se em atenção que **este nome só pode ser composto por 10 caracteres no máximo**.

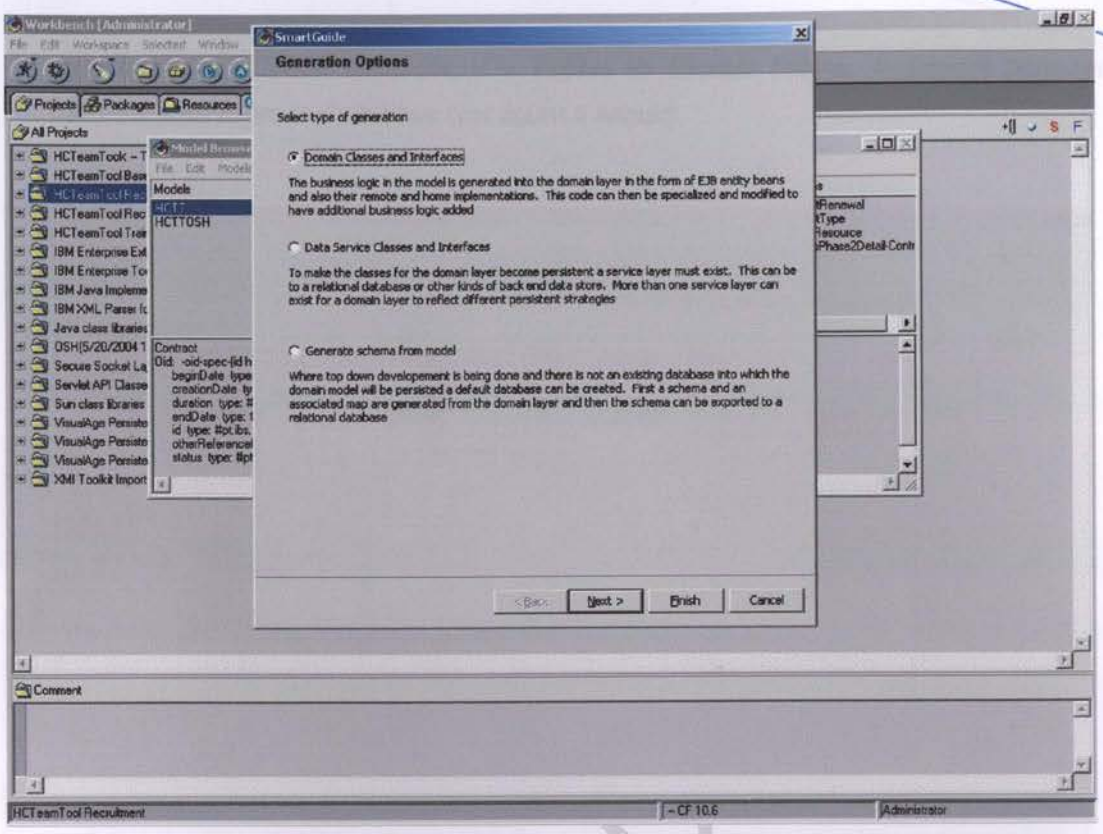
14. Na *box* "Primary key table" deve seleccionar-se a tabela à qual corresponde a chave-primária que se quer referenciar (e a coluna "Primary Key" situada abaixo desta *box* é automaticamente preenchida), e na *box* "Foreign key table" selecciona-se a tabela que contém a chave-estrangeira e, de seguida, o campo da tabela que será chave-estrangeira (essa selecção é feita na coluna "Foreign Key" situada abaixo da *box* referida) – ver figura seguinte.



15. Voltando ao ***Model Browser***, ir a ***Models -> Generate*** (ver figura).

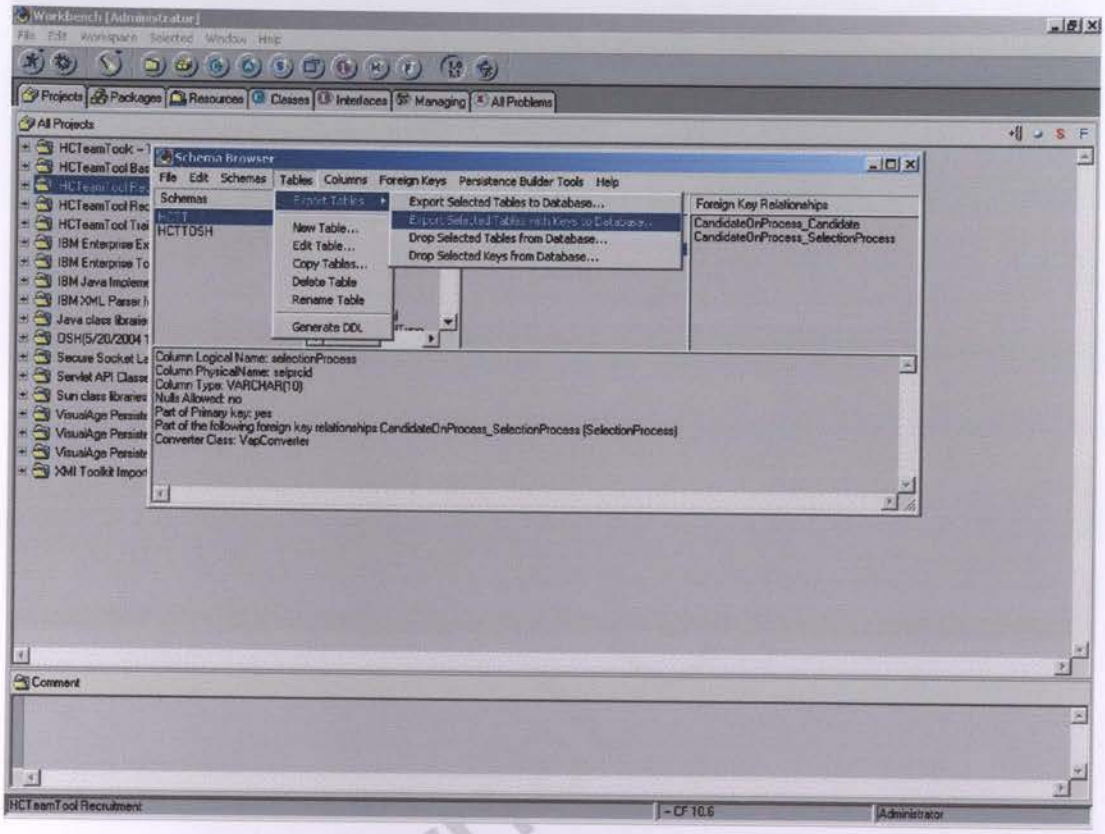


16. Ter atenção que, na janela apresentada a seguir se deve escolher sempre, no "Select type of Generation" a opção **"Domain Classes and Interfaces"** (consultar figura a seguir).



17. Carregar no botão **"Next >"** e seguir as instruções.

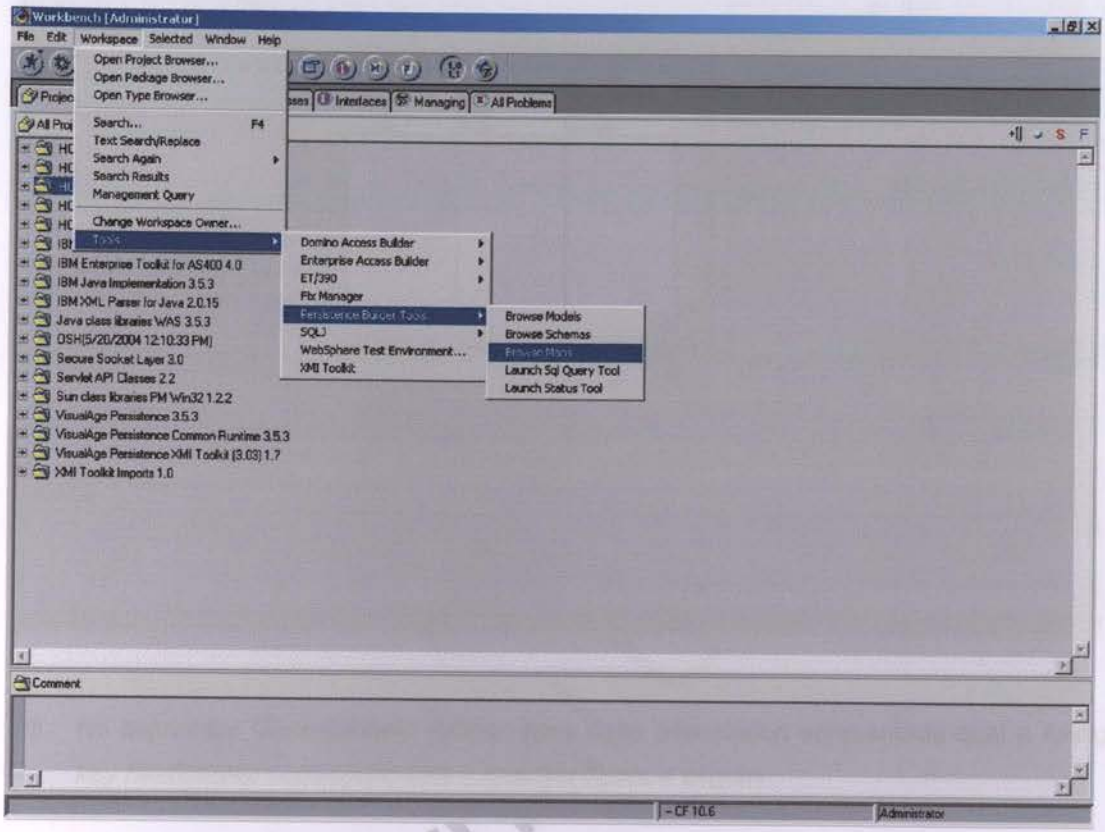
18. Voltando ao ***Schema Browser***, ir a ***Tables -> Export Tables -> Export Selected Tables with Keys to Database*** (ver figura a seguir).



19. Na janela apresentada a seguir deve ser escolhido, em "Connection type", a opção ***com.ibm.as400.access.AS400JDBCDriver*** e em "**Data Source**" deve ser colocado ***jdbc:as400:IBSD04;libraries=,HRMMB42F00;***

20. Nos campos "Username" e "Password" deve ser colocado o nome de utilizador e a palavra-chave de acesso do programador ao ibsd04 (ver figura seguinte).

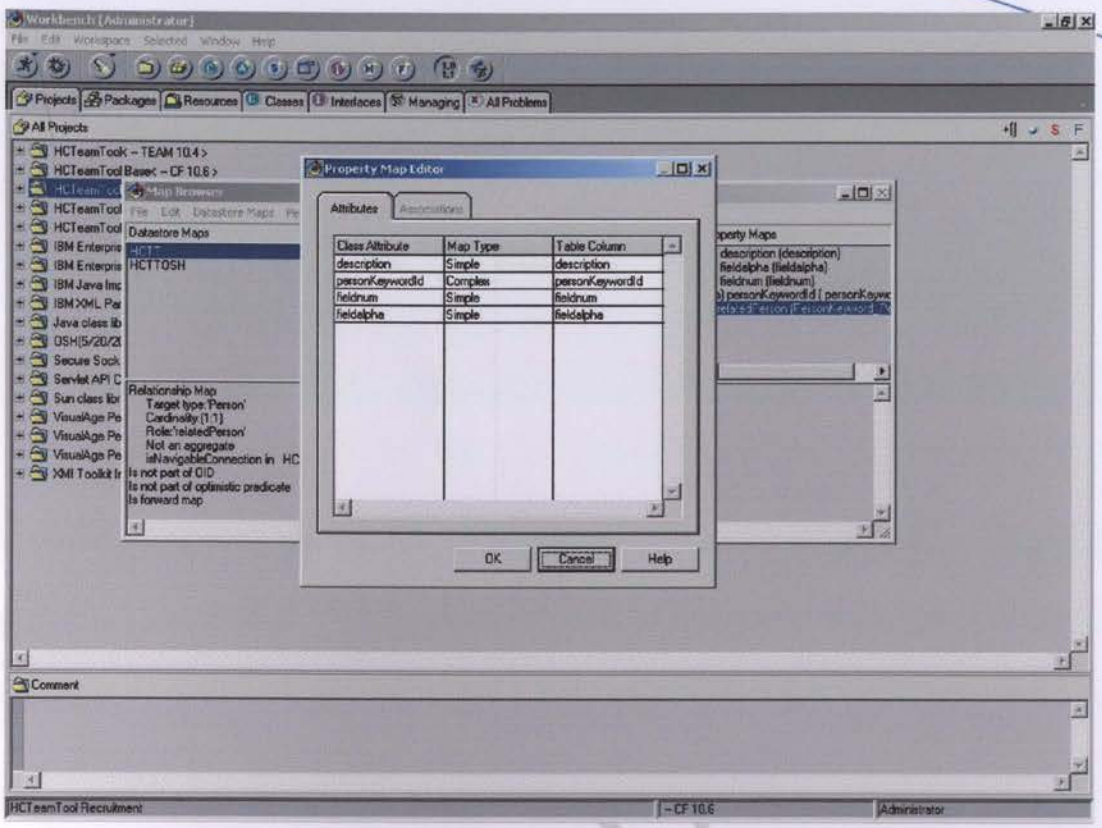
21. Ir a ***Workspace -> Tools -> Persistence Builder Tools -> Browse Maps*** (consultar figura a seguir).



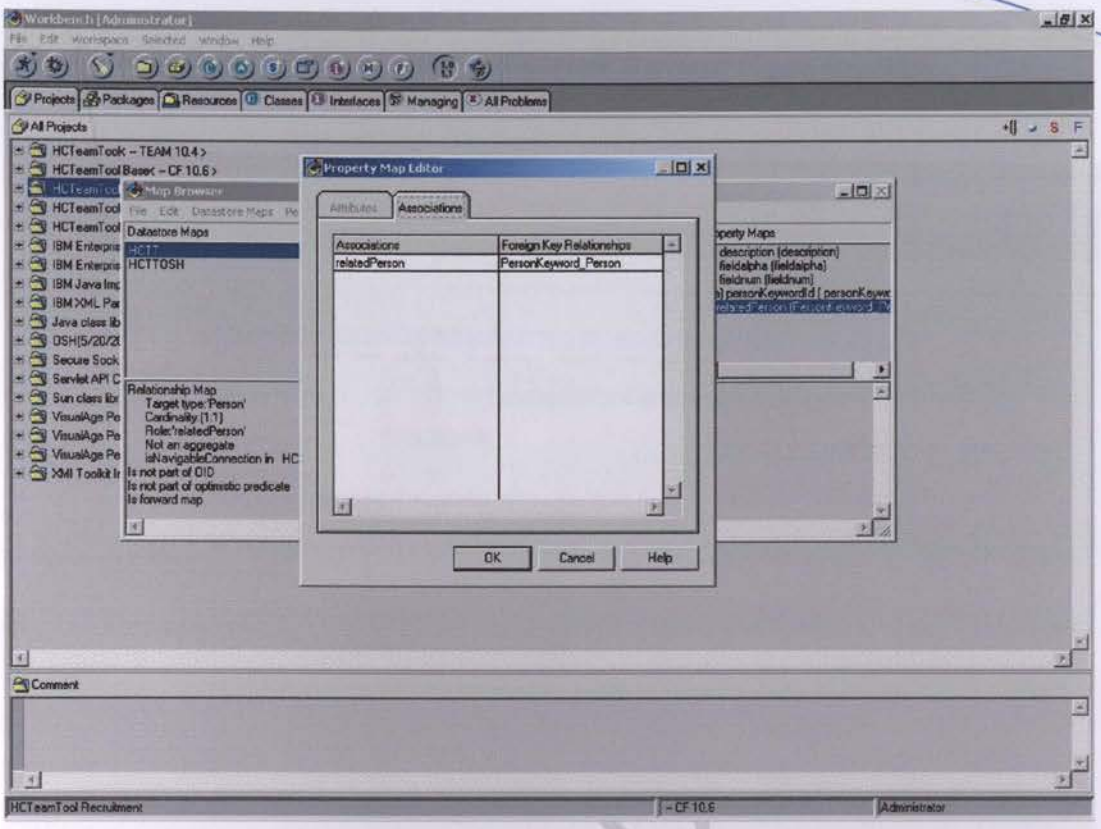
22. Seleccionar o *map* pretendido da coluna "Datastore Maps" situada do lado esquerdo da janela **"Schema Browser"** e seleccionar a tabela pretendida da coluna "Persistence Classes". Ir a ***Table Maps -> New Table Map -> Add Table Map with no inheritance*** (caso a tabela não tenha sub-tabelas). É apresentado o "Property Map Editor".

23. No separador "Attributes" , para cada atributo de classe, dizer se é tipo *complex* (tipo que não for básico, ou seja, que tenha sido implementado pela IBS), ou se é tipo *simple* (tipo simples ou primitivo).

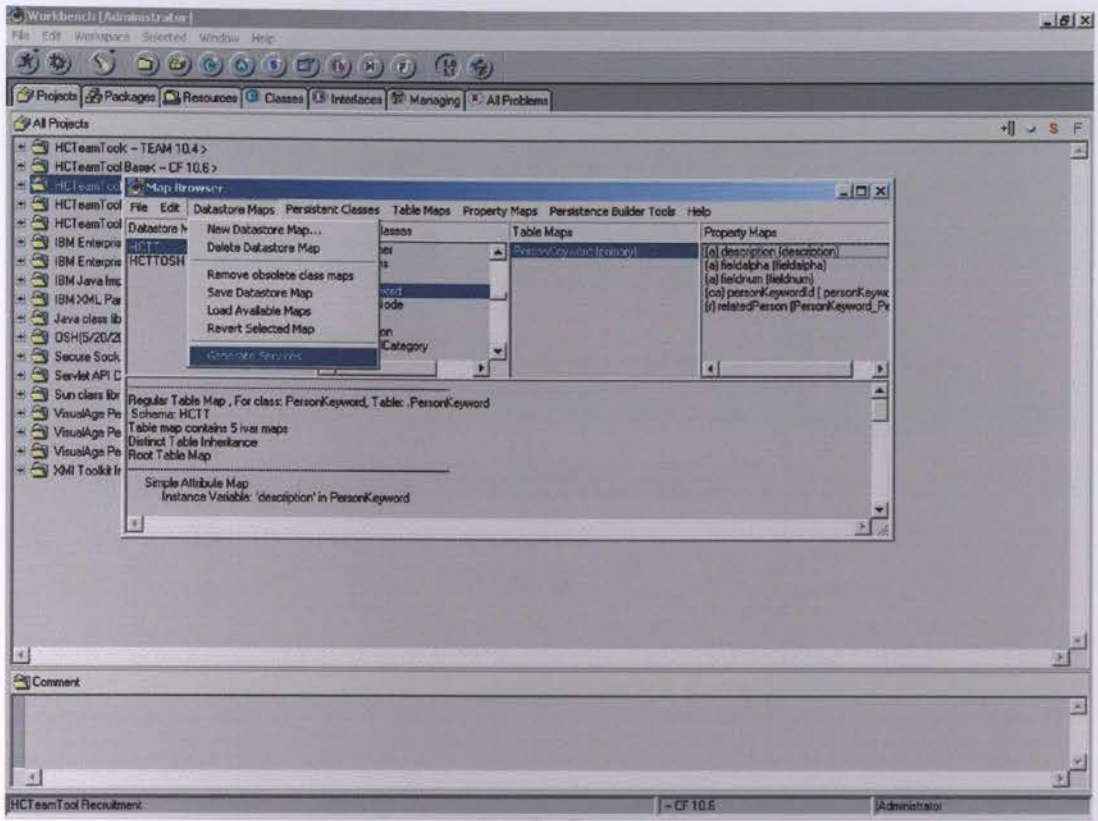
24. Em **"Table Column"** indicar a coluna / campo da tabela correspondente a cada atributo da classe. No caso do atributo de classe ser do tipo *complex* é necessário seleccionar o **"*Composer*"** que vai compôr o tipo (consultar figura a seguir).



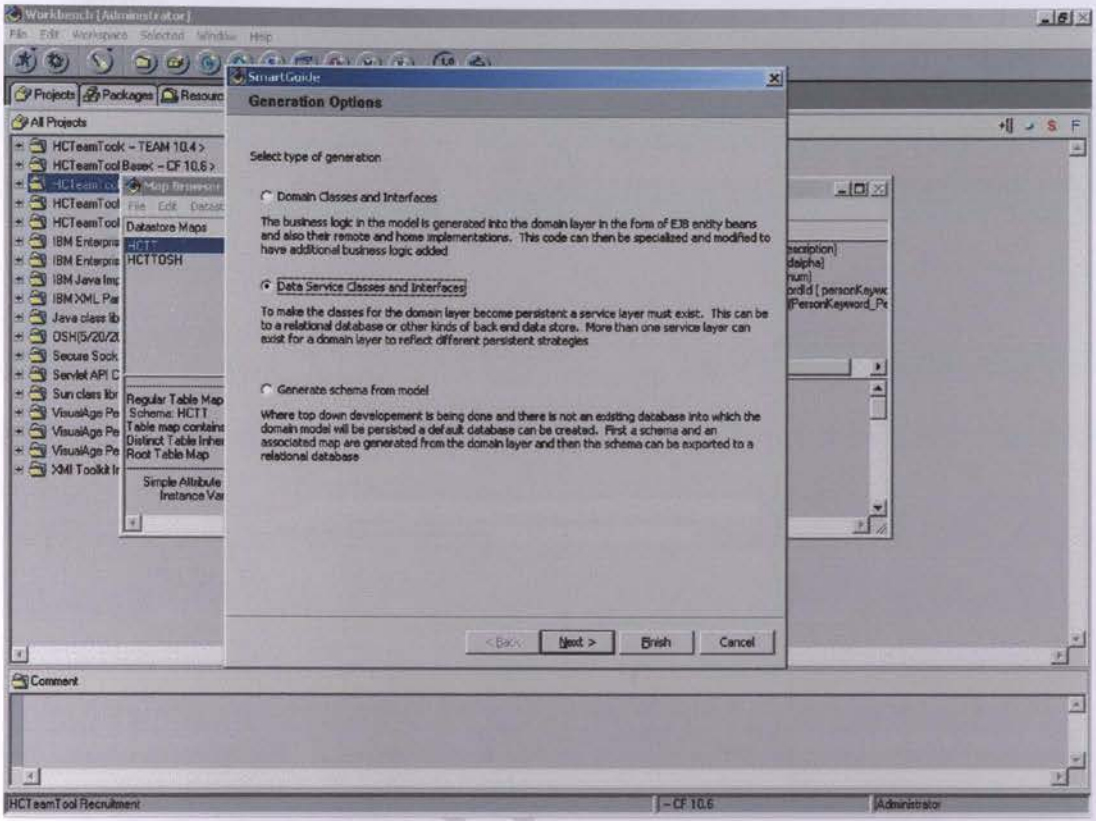
25. No separador "Associations" indicar, para cada *association* apresentada qual a *foreign key relationship* correspondente (consultar figura a seguir).



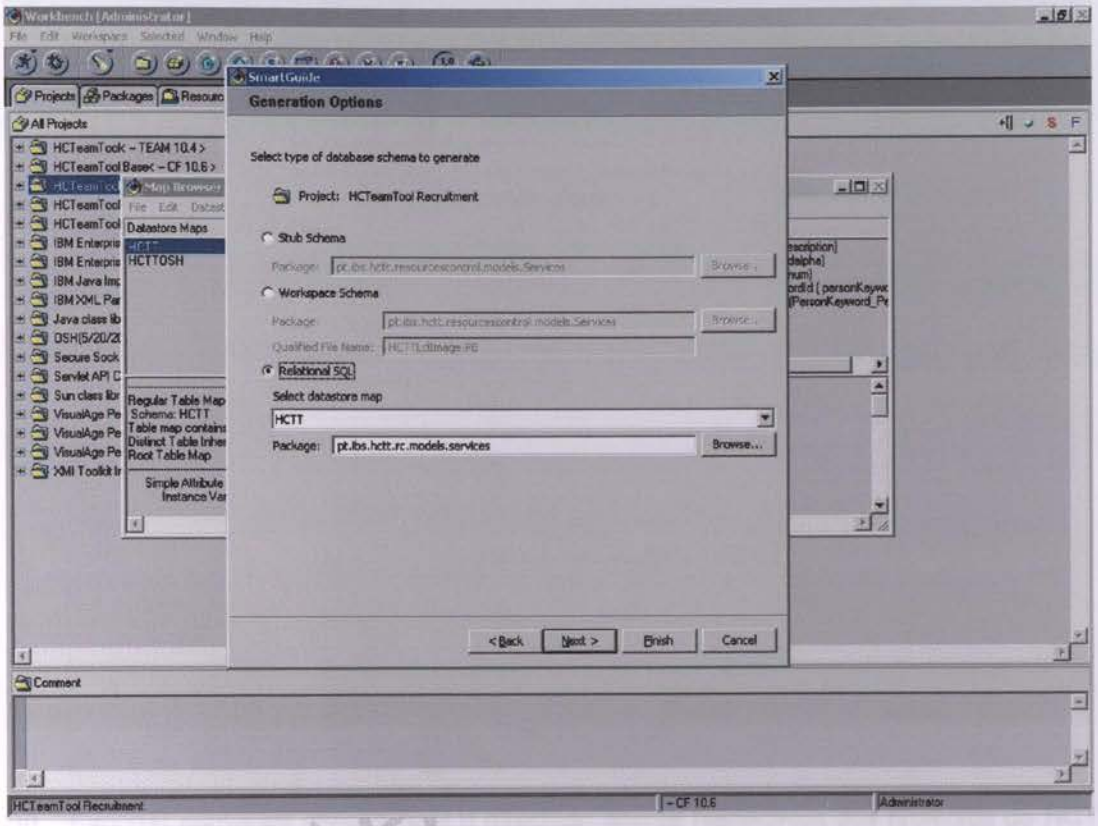
26. De seguida ir a ***Datastore Maps -> Generate Services*** (figura seguinte).



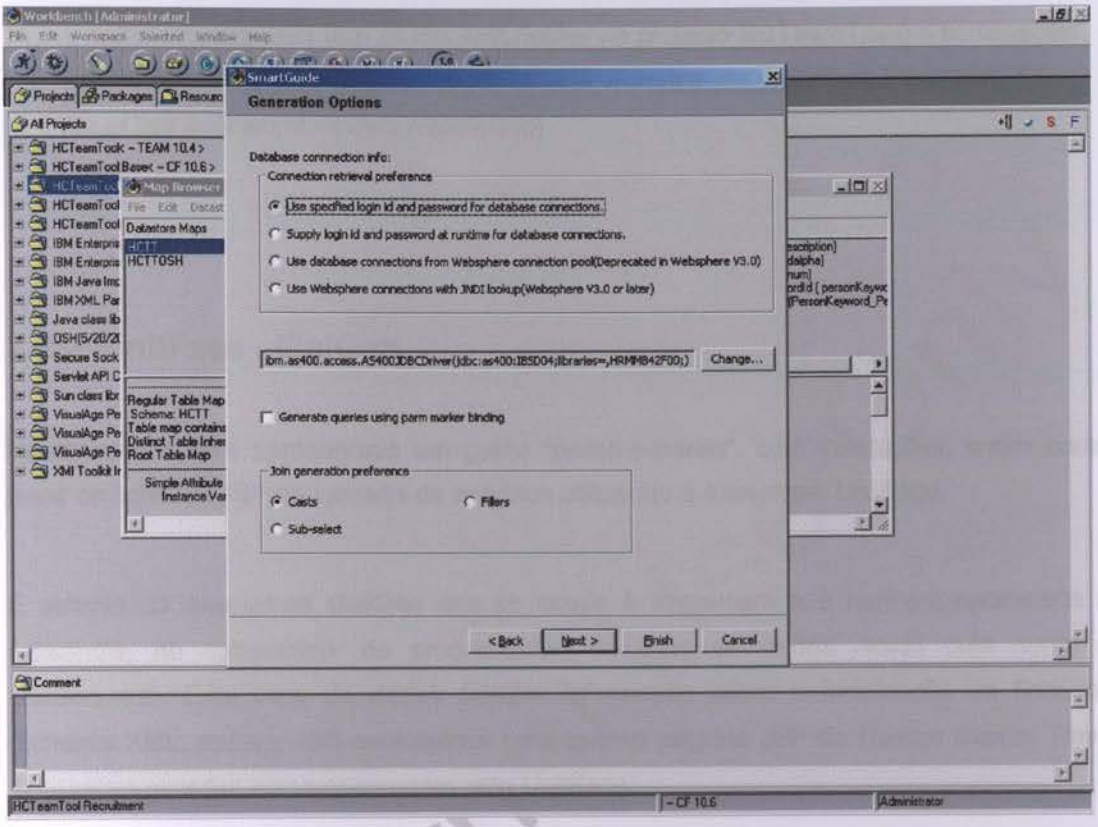
27. Na janela apresentada escolher **"Data Service Classes and Interfaces"** (figura a seguir).



28. Em **"Select type of database schema to generate"** escolher **"*Relational SQL*"**. Ter em atenção que, se for necessário, o *package* deve ser corrigido para que tudo esteja escrito em letra minúscula (por exemplo: *pt.ibs.hctt.helloworld.models.services*) – figura seguinte.



29. Na janela a seguir escolher ***"Use specified login id and password for database connections"*** e em "Join generation preference" escolher "***Casts***" (ver figura seguinte).



30. Exportar *packages* criados para o directório *source* respectivos dos projectos do HCTT e fechar o VisualAge.

31. Abrir o Eclipse e fazer *refresh* dos projectos para onde os *packages* foram exportados.

32. É gerado um *DataStore* por cada classe gerada anteriormente. Esse *DataStore* deve ser eliminado (os *DataStore* são utilizados para estabelecer a ligação à base de dados e no HCTT é utilizado, para isso, o ***HCTTDataStore***, localizado no *package* ***pt.ibs.hctt.cf.models.services*** do projecto **HCTeamTool**).

33. O ***HCTTDataStore*** referido acima como estando no *package* ***pt.ibs.hctt.cf.models.services*** do projecto **HCTeamTool** deve ser alterado de forma a referenciar o que há de novo (mais precisamente, os ficheiros ***Nome_qualquerHomeImpl.class***). Deve ir-se também ao método ***serviceObjectClasses()*** do ***HCTTDataStore*** para também referenciar o que foi criado (mais precisamente, os ficheiros ***nome_qualquerServiceObject.class***). Ter em

atenção que se o ***Nome_qualquerHomeImpl.class*** estiver numa posição *x* da lista, o ***nome_qualquerServiceObject.class*** também tem que estar **precisamente** na posição *x* da lista do método ***serviceObjectClasses()***.

34. Ir a ***JNDINames.java*** (em ***pt.ibs.hctt.helper*** no projecto **HCTeamTool**) e acrescentar nova **VARIAVEL_VAPHOME=caminho_até_ficheiro_da_interface** (exemplo: *ibs.pt.hctt.helloworld.models.HelloWorld*).

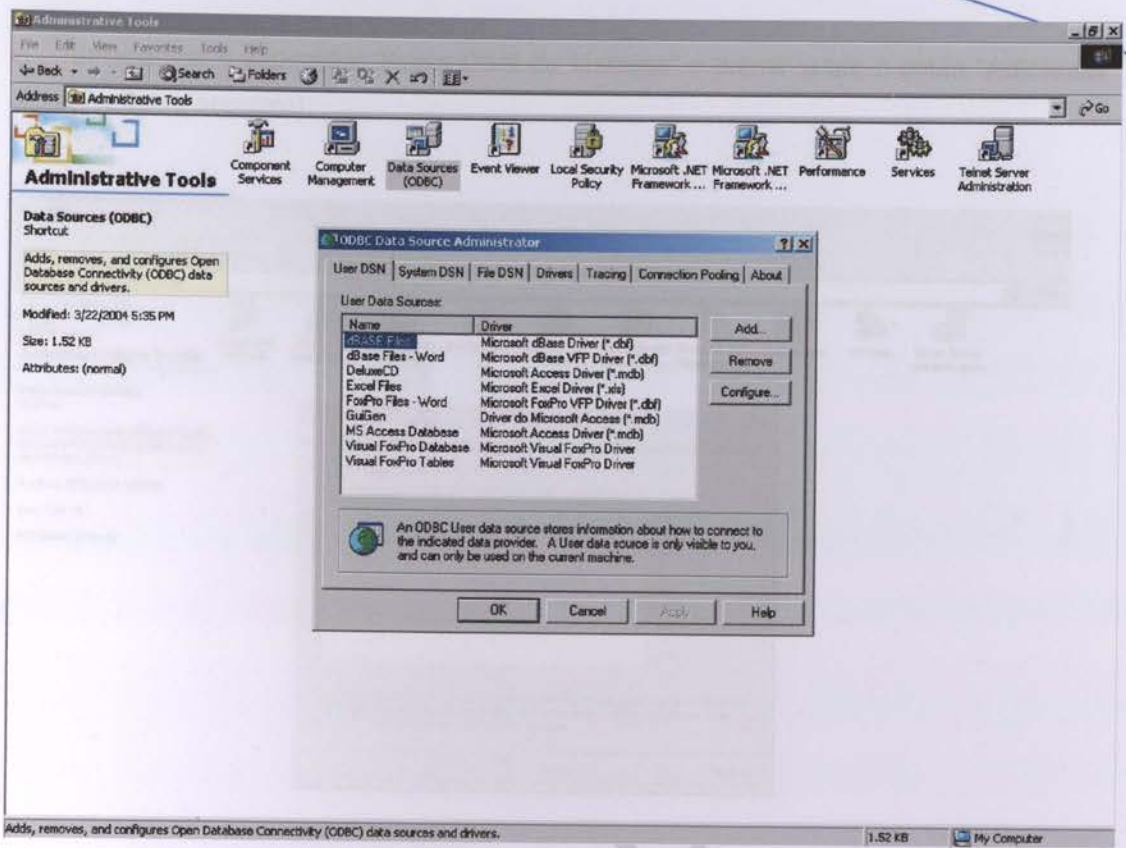
3.2.2 UnilFace - GuiGen

Nesta secção será apresentado um guião "passo-a-passo", com ilustrações, sobre como gerar os ficheiros JSP da camada de interface utilizando a *framework UnilFace*.

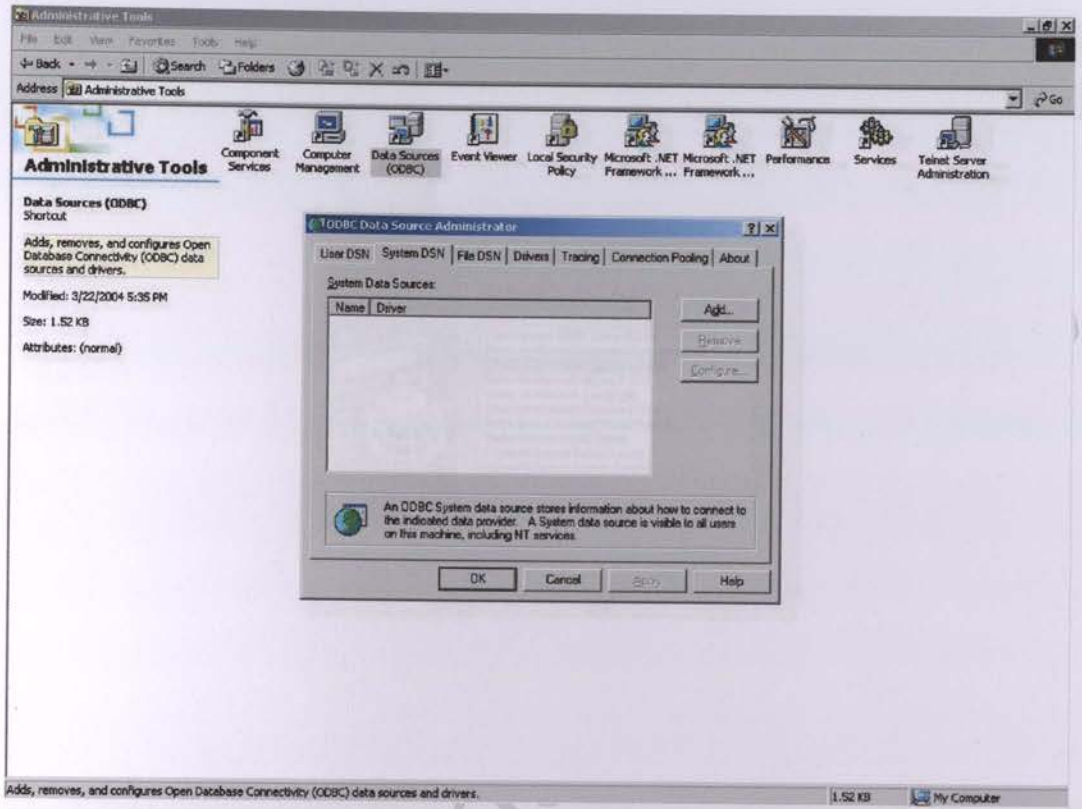
É através do executável ***GuiGen*** que se acede à *framework* e é também necessária a existência, no computador do programador, da base de dados *Acess* cujo nome é *GuiGen.mdb*. Esta base de dados contém informação sobre a localização de ficheiros (ficheiros XML, *patters,* etc) necessários para que as páginas JSP do *Human Capital Team Tool* sejam geradas automaticamente pelo *UnilFace*.

A forma como deve ser configurado o computador de modo a que o *GuiGen* possa utilizar esta base de dados é a seguinte:

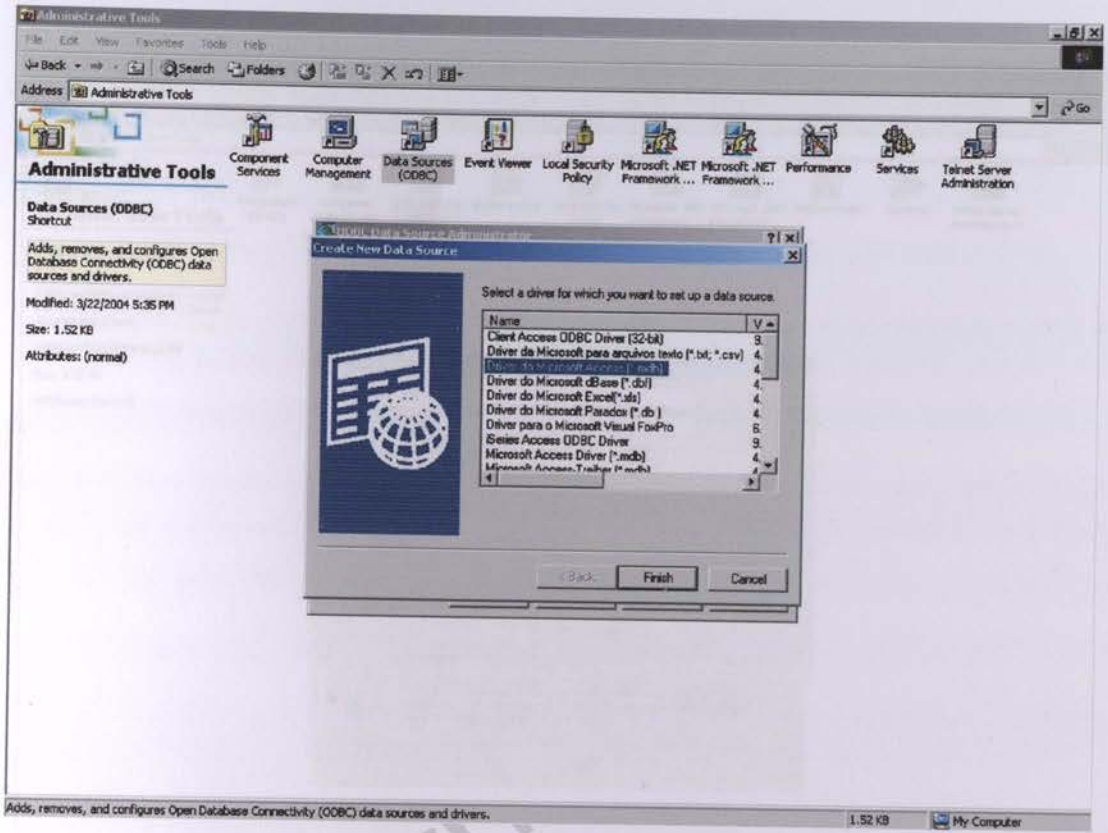
1. No computador, ir ao **Painel de Controlo -> Ferramentas administrativas -> Origem de dados (ODBC)**. É apresentada a seguinte janela:



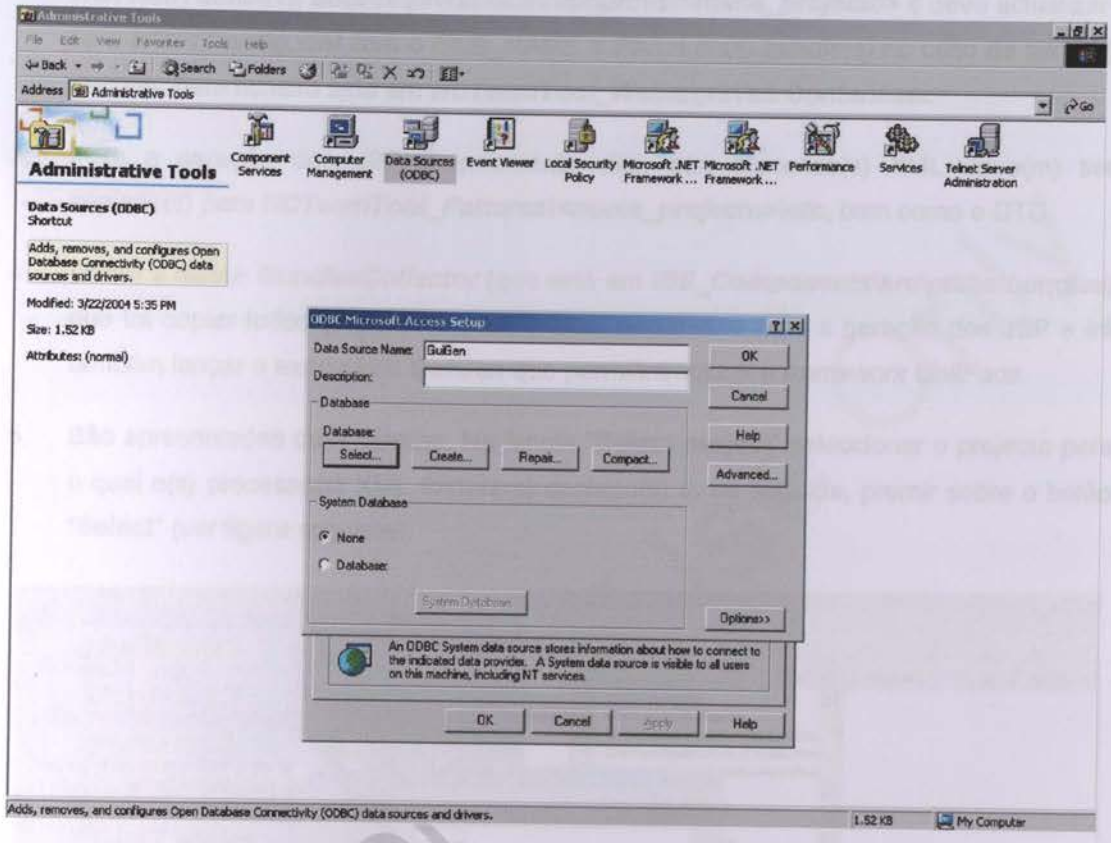
2. Seleccionar o separador **"Pasta DSN de sistema"** e premir sobre o botão **"Adicionar"** (ver figura seguinte):



3. É apresentada uma listagem de *drivers*. Escolher **"Driver do Microsoft Access (*.mdb)"** (consultar figura a seguir):



4. No formulário que é então apresentado, colocar como **"Nome de origem de dados"** *GuiGen*. Abaixo da *label* **"Database"** premir sobre o botão **"Seleccionar"** e fazer *browse* ao sítio onde a base de dados *GuiGen.mdb* está (ver figura a seguir):



Após a configuração apresentada acima é possível então utilizar o *UnilFace* para gerar JSP automaticamente tal como é indicado de seguida.

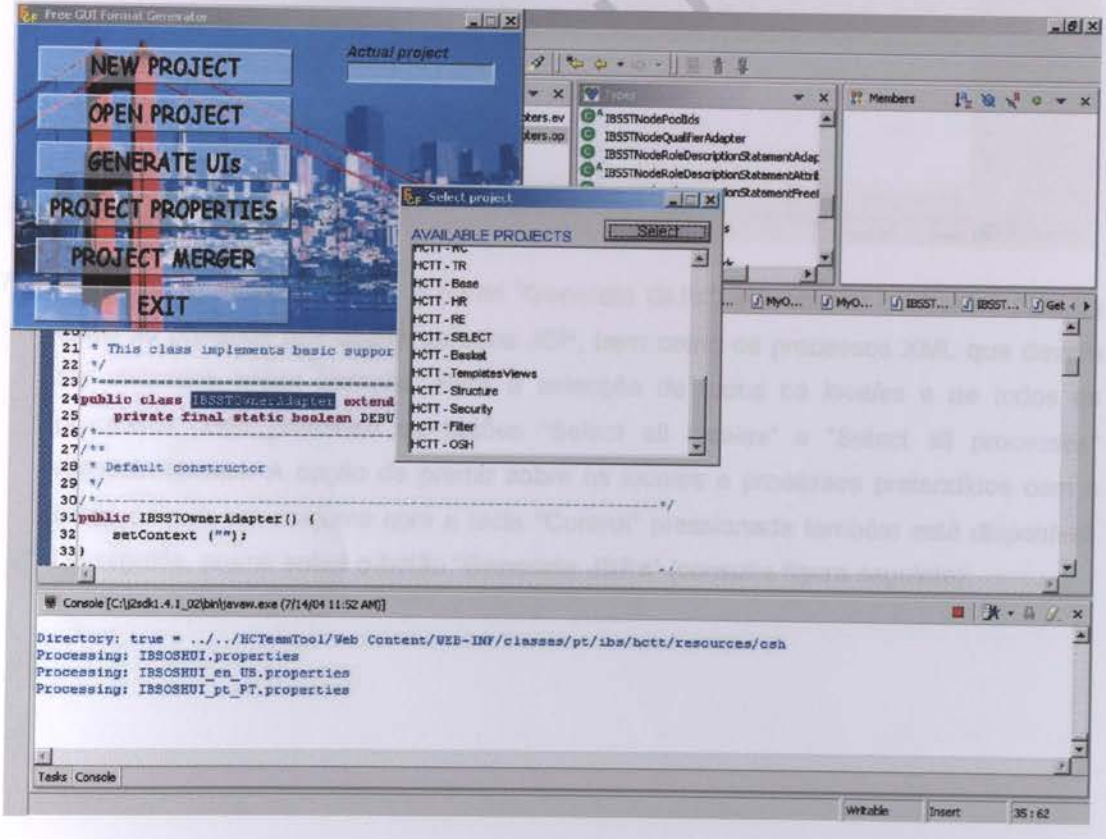
1. O(s) processo(s) XML criado(s) / editado(s) para gerar os JSP pretendidos devem estar em ***HCTeamTool_Webapp\Web Content\etc***.

2. O(s) *bundle*(s) criado(s) / editado(s) para gerar os JSP pretendidos devem estar em ***HCTeamTool\Java Source\pt\ibs\hctt\resources\<nome_projecto>*** e deve actualizar-se o ***HCTTConfig.xml*** com o FILE_NAME e PATH do(s) *bundle(s)* no caso de ser(em) novo(s). Este ficheiro está em ***HCTeamTool_Webapp\Web Content\etc***.

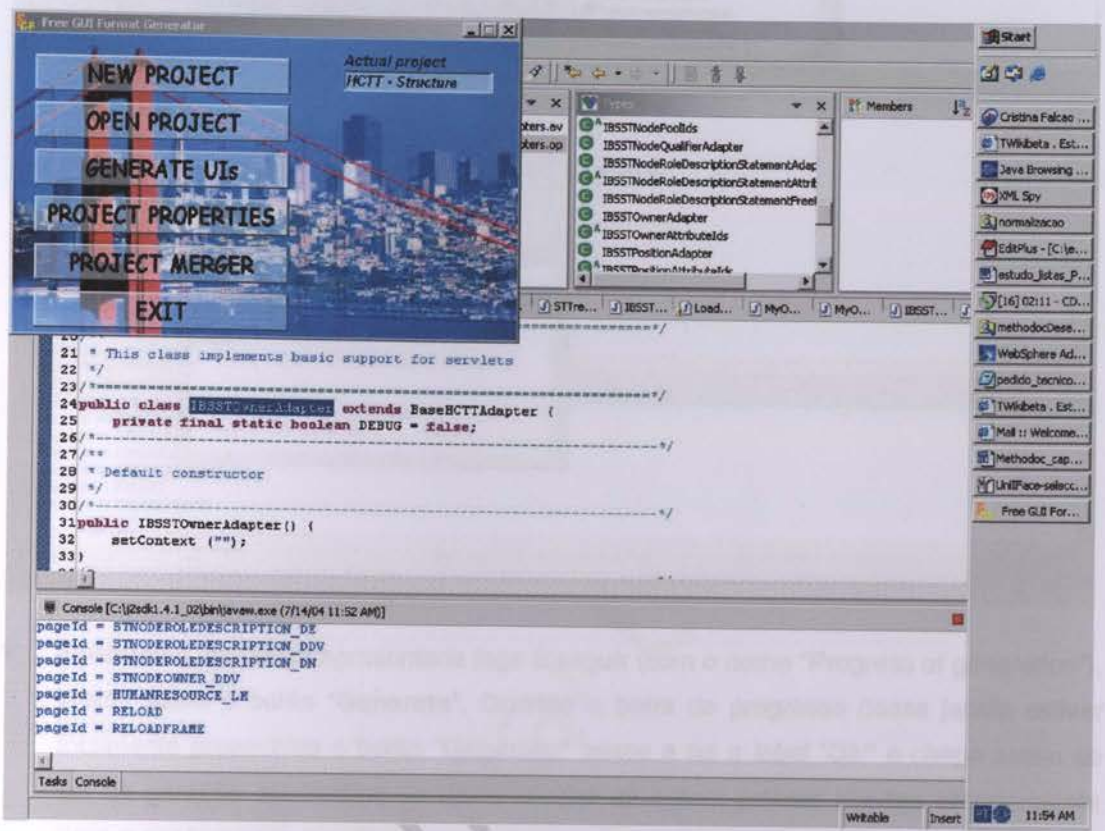
3. Para a geração dos JSP propriamente dita, o(s) processo(s) XML deve(m) ser copiado(s) para ***HCTeamTool_Patterns\<nome_projecto>\etc***, bem como o DTD.

4. Correr a classe ***BundlesCollector*** (que está em ***IBS_Components\src\pt\ibs\bundles***) que irá copiar todos os *bundles* para o local necessário para a geração dos JSP e irá também lançar o executável ***GuiGen*** que permitirá utilizar a *framework UnilFace*.

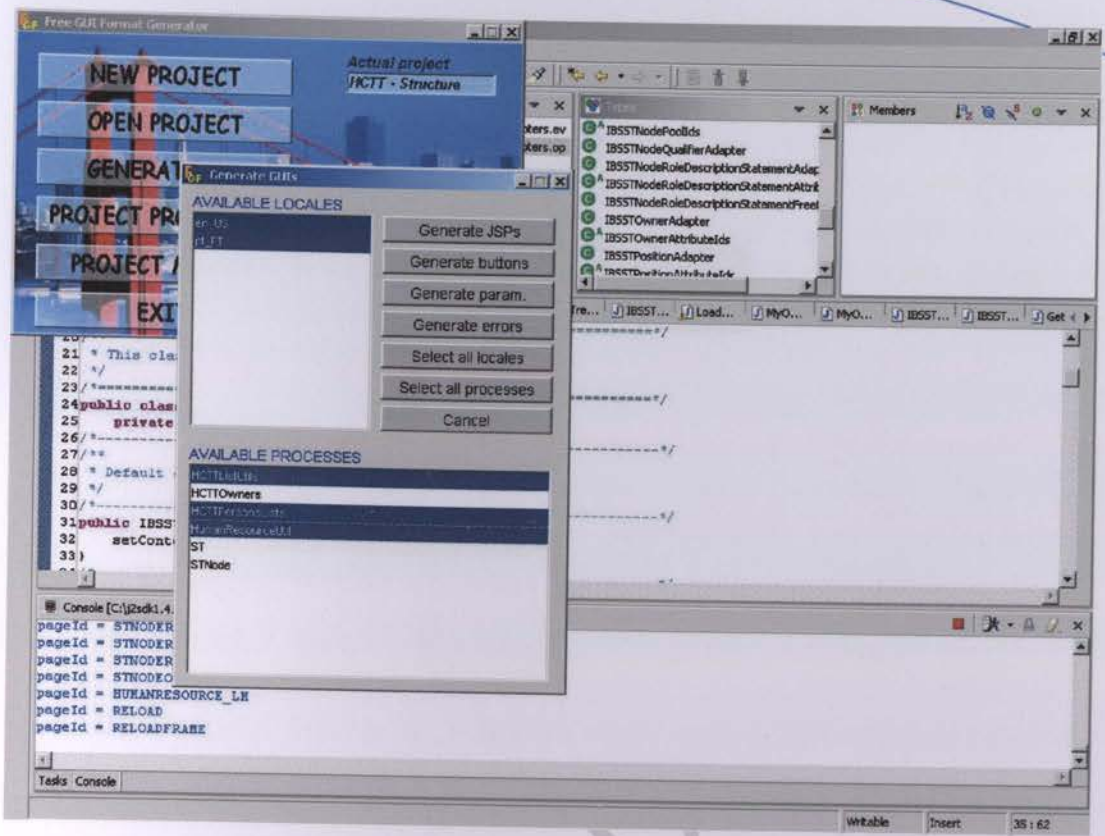
5. São apresentadas duas janelas. Na janela **"Select project"** seleccionar o projecto para o qual o(s) processo(s) XML foi(foram) copiado(s) e, de seguida, premir sobre o botão **"Select"** (ver figura seguinte).



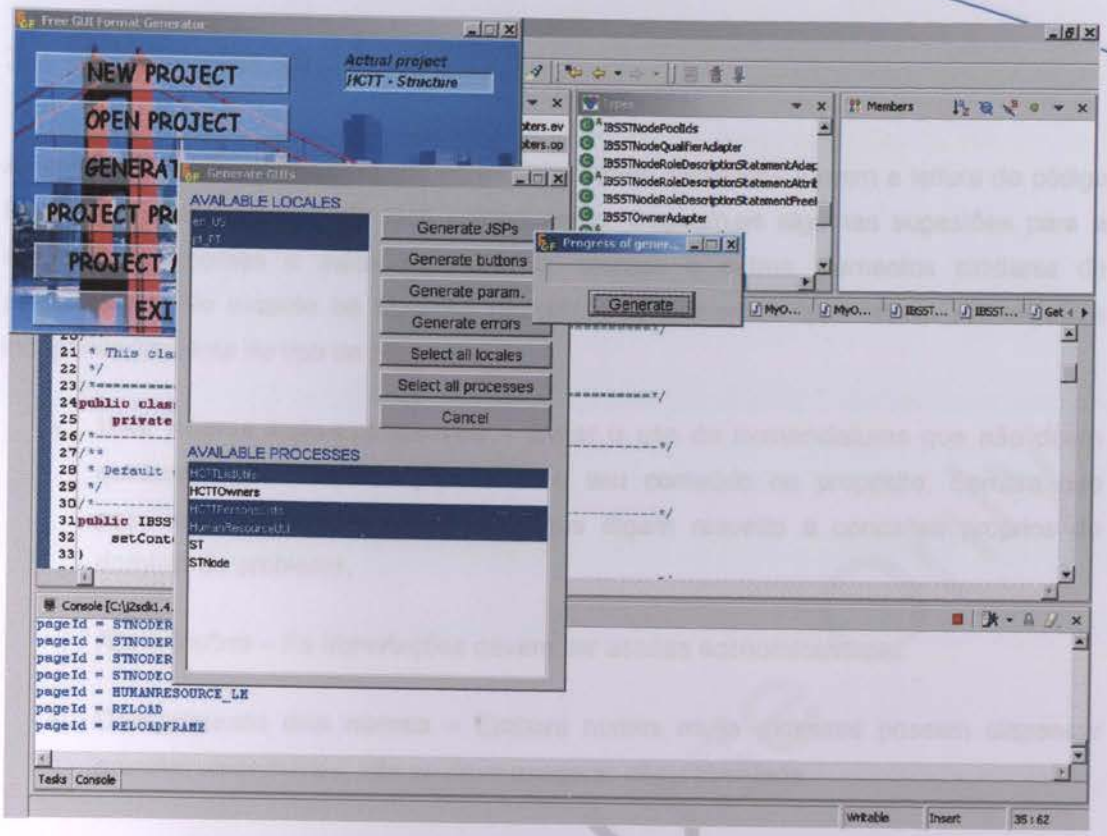
6. Na janela **"Free GUI Format Generator"** o campo "Actual Project" vai ser preenchido (caso não existam erros no(s) XML contidos no projecto seleccionado anteriormente) com o nome do projecto. Premir, então, sobre o botão **"GENERATE Uis"** (figura seguinte):



7. Na janela apresentada (com o nome "Generate GUIs") seleccionar os *locales* para os quais se pretende que sejam gerados JSP, bem como os processos XML que devem ser utilizados nessa geração. Para a selecção de todos os *locales* e de todos os processos pode premir-se os botões "Select all locales" e "Select all processes" respectivamente. A opção de premir sobre os *locales* e processos pretendidos com o botão do rato em conjunto com a tecla "Control" pressionada também está disponível. De seguida, premir sobre o botão **"Generate JSPs"** (consulta figura seguinte):



8. Finalmente, na janela apresentada logo a seguir (com o nome "Progress of generation"), premir sobre o botão "**Generate**". Quando a barra de progresso dessa janela estiver totalmente preenchida o botão "**Generate**" passa a ter a *label* "**Ok**" e chega assim ao fim da geração automática de JSPs. Todas as outras janelas abertas anteriormente devem ser fechadas (ver figura seguinte):



9. O(s) JSP gerado(s) vão estar em ***HCTeamTool_Patterns\<nome_projecto>\output\<nome_Locale>\<nome_processXML>*** e o directório onde este(s) JSP deverá(ão) ser colocado(s) é ***HCTeamTool_Webapp\Web Content\jsp\<nome_Locale>\<nome_processXML>*** (deve criar-se este directório caso ele não exista).

Nota muito importante: quando os ficheiros XML são copiados, no passo 3, através do "Explorer" do *Windows* ou através de outro meio qualquer que não seja o Eclipse, então deve fazer-se *refresh* ao projecto **HCTeamTool_Patterns** no Eclipse antes de se proceder para o passo 4. Isto também se aplica ao último passo (quando os ficheiros JSP são também copiados para outro destino) devendo, neste caso, ser feito *refresh* ao projecto **HCTeamTool_Webapp** após os JSP serem copiados. Se estes *refresh* não forem efectuados os projectos do Eclipse não estarão sincronizados com o sistema de ficheiros do computador.

3.3 Nomenclatura

As convenções adoptadas para dar nomes aos vários elementos tornam a leitura do código fonte muito mais rápida e de fácil entendimento. Seguem-se algumas sugestões para a atribuição de nomes a métodos, variáveis, classes e outros elementos similares de programação. No entanto há algumas directrizes fundamentais que devem ser seguidas independentemente do tipo de elemento:

- **Usar nomes auto-explicativos** – Evitar o uso de nomenclaturas que não dêem qualquer tipo de informação sobre o seu conteúdo ou propósito. Sempre que possível, devem-se aplicar nomes que digam respeito a conceitos próprios do domínio do problema.

- **Abreviações** – As abreviações devem ser usadas economicamente.

- **Comprimento dos nomes** – Embora nomes muito extensos possam dispensar grandes comentários, não se deve exagerar nesta liberdade.

3.3.1 Packages

O nome de um pacote deve seguir as seguintes regras:

- Ser escrito em letra minúscula.

- De acordo com o *standart* ISO 3166 o primeiro domínio é a abreviatura do país. Depois a empresa, o módulo e normalmente para cada módulo são feitos pacotes para cada camada com eventuais sub-divisões especificas a cada caso.

  país.empresa.projecto.modulo.tipo-componente.x_1.x_2.x_n

  pt.ibs.hctt.training.controllers.op

3.3.2 Interfaces

Os nomes para os interface devem respeitar as mesmas regras aplicadas às classes, que serão explicadas de seguida.

3.3.3 Classes

A nomenclatura adoptada é a aconselhada para o Java *standart*. As palavras que constituem os nomes das classes são concatenados e começam por letra maiúscula. O nome deve ser simples, explícito e sempre que possível deve-se evitar abreviaturas, embora por vezes se possa utilizar como é o caso de URL ou HTML.

```
class Raster;
class ImageSprite;
```

3.3.3.1 Tipos de Classes

- **Adapters e AttributeIds**
- **Comandos**
- **Controladores**
- **Entidades VAP**
- **Value Objects**
- **Tipos**

3.3.3.2 Constantes

O nome das constantes devem apresentar-se em letras maiúsculas e para separar cada uma das palavras que constituem o nome utiliza-se o *underscore* "_".

```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
static final int GET_THE_CPU = 1;
```

3.3.3.3 Variáveis

As variáveis Java devem apresentar-se com a primeira letra minúscula e a primeira da cada palavra que forma o nome da variável em letra maiúscula. O nome de variáveis com apenas uma letra pode ser utilizado em variáveis temporárias, como p.e. "i", "j" ... "n" para inteiros e "c", "d" ... "h" para caracteres.

```
int i;
char c;
float myWidth;
```

3.3.3.4 Métodos

Os nomes dos métodos devem ser iniciados por verbos, com as palavras concatenadas da mesma forma que nas classes à excepção da primeira letra do nome que deve ser minúscula.

```
run();
runFast();
getBackground();
```

3.3.4 Outros

A completar.

3.3.4.1 Bundles

A completar.

3.3.4.2 XML

- **Processos XML**
- **Páginas (nomenclatura UnilFace)**
- **Parâmetros**
- **Acções**

A completar.

3.4 Normas de Codificação

Best pratices para cada uma das tecnologias.

3.4.1 Java

3.4.1.1 Reutilização de páginas

- **Views**
- **Páginas genéricas**

3.4.2 JSP

3.4.3 Javascript

3.4.3.1 Funções genéricas

3.4.3.2 Funções de Controlo de Janelas

3.4.4 XML

3.5 Documentação de Classes e Métodos

Importância e objectivo da documentação de código. Definição do estilo de documentação a adoptar.

3.5.1 Javadoc

Particulariedades do JavaDoc.

3.5.2 Exemplo

A completar.

3.6 Implementação de Listagens de Responsáveis e de Colaboradores Activos

Esta secção pretende fornecer um manual para o programador onde é explicado passo-a-passo, e completado com um exemplo prático, como devem ser implementadas as listagens de Responsáveis e/ou de Colaboradores Activos em qualquer página do *Human Capital Team Tool*.

Para tal existem dois processos XML fundamentais, que são **HCTTOwners.xml** e **HCTTPersonsLists.xml**, bem como dois *adapters* essenciais, que são **pt.ibs.hctt.base.gui.adapters.hr.OwnersAdapter** e **pt.ibs.hctt.base.gui.adapters.hr.ActiveHRsAdapter**.

3.6.1 Listagens de Responsáveis

1º passo – Alterar o XML que listará os Responsáveis de uma entidade

É no processo **HCTTOwners.xml** que está definida a página que permitirá listar os Responsáveis de uma qualquer entidade do HCTT, que é a *OWNERS_LM,* bem como as páginas de confirmação de eliminação de um Responsável.

A página *OWNERS_LM* tem definidas várias vistas cuja utilização será, de seguida, explicada e exemplificada.

view_list_maintenance_createOwnersOneAction – uma vista em que a acção de criar novos Responsáveis chama uma página de Colaboradores Activos onde só se adiciona um Responsável de cada vez através de uma acção de *context-menu*. Um **exemplo** da utilização desta vista pode ser encontrado no módulo de **Gestão de Recursos**, nos Responsáveis de uma Estrutura ou de um Nó.

Para utilizar esta vista basta colocar no código XML do *multipart* onde a lista de Responsáveis estará incluída, uma *form* que chame a página *HCTTOwners|OWNERS_LM* onde é também indicada a vista pretendida.

Exemplo:

No processo *OPST.xml* (que define as páginas de cada Estrutura do módulo de Gestão de Recursos) o *multipart* que irá conter a listagem de Responsáveis (que serão, neste caso, os Responsáveis por uma Estrutura) é:

```
<page id="STRUCTURE_DV" url="/jsp/UserLocaleHere/OPST/STRUCTURE_DV.jsp"
default_bundle="IBSOPUI">
        <descriptive_data description="idSTRUCTURE_DV"/>
        <image id="idImage" bundle="IBSOPUI"/>
        <view name="view_multipart"/>
        <form page_id="OPST|STRUCTURE_MDV"/>
        <form page_id="OPST|STROLEDESC_MLM"/>
        <form page_id="HCTTOwners|OWNERS_LM">
                <view name="view_list_maintenance_createOwnersOneAction"/>
        </form>
</page>
```

view_list_maintenance_createOwnersTwoActions - uma vista em que a acção de criar novos Responsáveis chama uma página de Colaboradores Activos onde se podem adicionar múltiplos Responsáveis de cada vez (através da selecção múltipla de Colaboradores Activos e premindo, de seguida, o botão que desencadeia a acção de criação de múltiplos novos Responsáveis) ou, alternativamente, se pode adicionar um Responsável de cada vez através de uma acção de *context-menu*. Um **exemplo** da utilização desta vista pode ser encontrado no módulo de **Recrutamento**, nos Responsáveis por um Processo de Selecção.

Para utilizar esta vista basta colocar no código XML do *multipart* onde a lista de Responsáveis estará incluída, uma *form* que chame a página *HCTTOwners|OWNERS_LM* onde é também indicada a vista pretendida.

Exemplo:

No processo *Recruitment.xml* (que define a maioria das páginas do módulo de Recrutamento) o *multipart* que irá conter a listagem de Responsáveis (que serão, neste caso, os Responsáveis por um Processo de Selecção) é:

```
<page id="RSP_DE_MV" url="/jsp/UserLocaleHere/Recruitment/RSP_DE_MV.jsp"
default_bundle="IBSRCUI">
        <descriptive_data description="idRSP_DE_MV"/>
        <title id="" bundle="IBSRCUI"/>
        <image id="idImage" bundle="IBSRCUI"/>
        <view name="view_multipart"/>
        <form page_id="RSP_DE"/>
        <form page_id="HCTTOwners|OWNERS_LM">
                <view name="view_list_maintenance_createOwnersTwoActions"/>
        </form>
        <form page_id="SEARCHCONF_DE"/>
</page>
```

Ainda no processo **HCTTOwners.xml** está também definida uma página que permite somente listar os Responsáveis de uma qualquer entidade do HCTT, sem fornecer qualquer tipo de acção sobre esta listagem, que é a página *OWNERSVIEW_LM*. Um **exemplo** da utilização desta página pode ser encontrado no módulo de **Estrutura**, nos Responsáveis de uma Estrutura ou de um Nó.

Para utilizar esta página basta colocar no código XML do *multipart* onde a lista de Responsáveis estará incluída, uma *form* que chame a página *HCTTOwners|OWNERSVIEW_LM*.

Exemplo:

No processo *ST.xml* (que define as páginas de cada Estrutura do módulo de Estrutura) o *multipart* que irá conter a listagem de Responsáveis (que serão, neste caso, os Responsáveis por uma Estrutura) é:

```
<page id="STRUCTUREVIEW_DV" url="/jsp/UserLocaleHere/ST/STRUCTUREVIEW_DV.jsp"
default_bundle="IBSSTUI">
        <descriptive_data description="idSTRUCTURE_DV"/>
        <title id="idStructure" bundle="IBSSTUI"/>
        <image id="idImage" bundle="IBSSTUI"/>
        <view name="view_multipart"/>
        <form page_id="STRUCTUREVIEW_MDV"/>
        <form page_id="STROLEDESCRIPTIONVIEW_MLM"/>
        <form page_id="HCTTOwners|OWNERSVIEW_LM" />
</page>
```

Uma vez definido o *multipart* que irá conter a página de listagem de Responsáveis é ainda necessário executar mais dois passos simples.

2º passo - -Colocar os output_parameter's necessários

No processo XML onde o *multipart* referido acima está definido deve pesquisar-se por todas as outras páginas que chamam, numa acção, esse *multipart* (através do atributo `page_id` de uma `action`) e colocar como `output_parameter` 's dessa acção o(s) *id(s)* da entidade e os comandos de criação, eliminação e selecção de Responsáveis da entidade. É necessário tomar alguns cuidados na forma como estes parâmetros são passados e para os explicar será utilizado o exemplo a seguir.

Exemplo:

No processo **Recruitment.xml** foi visto que o *multipart* contendo a listagem de Responsáveis por um Processo de Selecção é RSP_DE_MV. Se for feita uma pesquisa, no processo **Recruitment.xml**, pelas outras páginas que chamam este *multipart* chega-se ao seguinte resultado:

- RSP_LM
- RSPCLOSED_LM
- RSPALL_LM
- RSP_DN
- RSPFREEPOS_LM
- SEARCHCONF_DE
- CHOOSEFUNCTION_DE
- CHOOSEFUNCTIONINEDIT_DE

Em cada `action` destas páginas cujo `page_id` é RSP_DE_MV devem definir-se os seguintes `output_parameter`'s:

```
<output>
        <output_parameter id="SelectionProcessId" />
        <output_parameter id="SelectionProcessId" target="EntityId1"/>
        <output_parameter id="Commands"
value="LoadListCommand=pt.ibs.hctt.rc.cmds.usr.se.GetSelectionProcessOwnersColCmd#Dele
teCommand=pt.ibs.hctt.rc.cmds.usr.se.DltSelectionProcessOwnerCmd#CreateCommand=pt.ibs.
hctt.rc.cmds.usr.se.CrtSelectionProcessOwnerCmd" />
</output>
```

Importa então tomar atenção que:

- Ao *id* do Processo de Selecção foi associado um *alias* com o nome *EntityId1*. Isto acontece porque a página *OWNERS_LM* recebe, como `input_parameter`, um pârametro com este nome. Analisando o código XML desta página, pode verificar-se que ela está preparada para receber, no máximo, 5 *ids*, uma vez que há entidades do HCTT que, para serem identificadas, têm *ids* compostos (por exemplo, um Nó é identificado pelo *id* do Nó e pelo *id* da Estrutura à qual pertence). O mesmo *id* é também passado mas sem o atributo `target` e isto acontece pois o *multipart* RSP_DE_MV contém outras páginas que necessitam do *SelectionProcessId* como pârametro e não o reconheceriam se fosse passado com um *alias EntityId1*.
- O pârametro *Commands* deve ter sempre este nome e através do atributo `value` são passados os comandos de criação, eliminação e selecção de Responsáveis da entidade pretendida. A ordem pela qual os comandos são passados pode ser diferente daquela que é aqui apresentada.

3º passo – Alterar os construtores dos comandos

O construtor das classes dos comandos passados anteriormente é responsável por verificar se o(s) *id(s)* da(s) entidade(s) não é passado com valor *null* ou vazio. No caso da criação, eliminação e selecção de Responsáveis de um Processo de Selecção, é verificado se o *id* do Processo de Selecção é passado correctamente. Uma vez que, para esses comandos, o nome deste *id* não está agora associado a *SelectionProcessId* mas sim a *EntityId1*, deve alterar-se ou implementar-se o construtor das classes de cada um dos comandos mencionados acima para fazerem a verificação ao parâmetro com o nome *EntityId1* e ter também este cuidado em qualquer outro local da classe onde seja pretendido obter esse parâmetro.

Uma alteração semelhante, embora desta vez ao *id* identificador do Responsável, deve ser feita nas classes dos comandos de criação e de eliminação de Responsáveis. O nome do *id* de um Responsável deverá ser sempre *HRId*, independentemente do tipo de entidade do qual é Responsável.

3.6.2 Listagens de Colaboradores Activos

1º passo - Alterar o XML que listará os Colaboradores Activos

É no processo **HCTTPersonsLists.xml** que está definida a página que permitirá listar os Colaboradores Activos do HCTT, que é a *ACTIVEHRS_LM.*

A página *ACTIVEHRS_LM* tem definidas três vistas e o processo **HCTTPersonsLists.xml** tem também definidas três páginas *multipart*, onde cada *multipart* contém uma *form* constituída pela ACTIVEHRS_LM, referenciando uma das suas vistas.

As três vistas definidas para a página ACTIVEHRS_LM são:

view_list_maintenance_addOwnersOneAction - só se adiciona um Colaborador Activo de cada vez, através de uma acção de *context-menu*, como, por exemplo, Responsável de uma entidade. O *multipart* que referencia esta vista é ACTIVEHRS_ONEACTION_MV.

view_list_maintenance_addOwnersTwoActions - pode adicionar-se múltiplos Colaboradores Activos de cada vez (através da selecção múltipla de Colaboradores Activos e premindo, de seguida, o botão que desencadeia a acção de criação de múltiplos novos Colaboradores Activos como, por exemplo, Responsáveis de de uma entidade) ou, alternativamente, se pode adicionar um Colaborador Activo de cada vez através de uma acção de *context-menu*. O *multipart* que referencia esta vista é ACTIVEHRS_TWOACTIONS_MV.

view_list_maintenance_addOwnerByReference – pode passar-se, para um formulário, a referência deste Colaborador Activo (para criar, por exemplo, um novo utilizador do HCTT). O *multipart* que referencia esta vista é ACTIVEHRS_REFERENCED_MV.

A utilização das páginas genéricas de Colaboradores Activos é semelhante à utilização das páginas de Responsáveis.

No processo XML que se está implementar e onde, em dada altura, existe uma página onde uma determinada `action` abre uma página de Colaboradores Activos, basta só colocar como `page_id` dessa `action` o nome de um dos *multiparts* referidos anteriormente – a escolha desse *multipart* é feita pelo programador, conforme aquilo que pretender – e passar os parâmetros necessários, que serão explicados no exemplo a seguir.

Exemplo:

```
<action name="action_create" value="idCreate"
page_id="HCTTPersonsLists|ACTIVEHRS_ONEACTION_MV"    bundle="IBSRCUI" selection="0">
        <hook event_name="onClick">
                <event_method name="hr_list" url="">
                        <event_method_parameter name="this.form"/>
                        <event_method_parameter name="'NEWSTOWNER'"/>
                        <event_method_parameter name="'action_create'"/>
                </event_method>
        </hook>
        <output>
                <output_parameter id="SelectionProcessId" target="EntityId1" />
                <output_parameter id="Commands" value="
CreateCommand=pt.ibs.hctt.rc.cmds.usr.se.CrtSelectionProcessOwnerCmd#"/>
        </output>
</action>
```

Ao *id* do Processo de Selecção foi associado um *alias* com o nome *EntityId1*. Isto acontece porque a página *ACTIVEHRS_LM* recebe, como `input_parameter`, um pârametro com este nome. Analisando o código XML desta página, pode verificar-se que ela está preparada para receber, no máximo, 5 *ids*, uma vez que há entidades do HCTT que, para serem identificadas, têm *ids* compostos (por exemplo, um Nó é identificado pelo *id* do Nó e pelo *id* da Estrutura à qual pertence).

O pârametro *Commands* deve ter sempre este nome e através do atributo `value` é passado o comando de criação, neste exemplo, de um Responsável por um Processo de Selecção.

2º passo – colocar sempre HRId como referência de um Colaborador Activo

Na classe do comando de criação referido no 1º passo deve ter-se atenção que o Colaborador Activo seleccionado na página é referenciado sempre como *HRId*, logo deve ter-se cuidado em sempre utilizar este nome quando se pretende obter o parâmetro de referência ao Colaborador Activo.

3.7 Implementação de tabs (separadores)

Esta secção pretende fornecer um manual para o programador onde é explicado passo-a-passo, e acompanhado de exemplos, como devem ser implementados *tabs* (separadores) em qualquer página do *Human Capital Team Tool*.

Para colocar *tabs* num qualquer local do HCTT são necessários 6 passos muito simples:

1. Criar uma nova `page` num XML por cada página em *tabs* que se pretenda.
2. Nas acções de qualquer página que chamem a página em *tab*, colocar como `page_id` a `page` elaborada no passo anterior.
3. Implementar um novo JSP para os *tabs*.
4. Implementar um *servlet*.
5. Alterar o ficheiro *web.xml*.
6. Inserir os novos *servlets* na *alpha*.

Segue-se uma descrição detalhada de cada passo.

1º passo: Criar uma nova `page` num XML por cada página em *tabs* que se pretenda.

Por cada página em *tabs*, deve implementar-se uma nova `page` no processo XML pretendido.

Esta nova `page` deverá ser como o exemplo a seguir:

```
<page id="RCSELPROCDETAILSTAB"
url="/servlet/RCSelProcDetailsTabServlet?PROCESSES=Recruitment, HCTTOwners,
Recruitment,
void&amp;MAIN_CLASS=Recruitment&amp;BUNDLE=IBSRCUI&amp;PARAMETERS=CourseId&amp;PAGES=R
SP_DE, OWNERSTWOACTIONS_LM, SEARCHCONF_DE,
void&amp;TABS=idTabSelectionProcessEditTitle, idTabOwnerListTitle,
idTabSearchConfigurationsTitle&amp;PAGEJSP=RCSelProcDetailsTab&amp;MODULE=Recruitment&
amp;RESETTAB=True" default_bundle="IBSRCUI">
        <descriptive_data description="idTRCOURSETAB"/>
        <title id="" bundle="IBSTRUI"/>
        <image id="idImage" bundle="IBSTRUI"/>
        <view name="view_detail_view"/>
        <page_data/>
        <adapter class_name="pt.ibs.hctt.rc.gui.adapters.se.SelectionProcessAdapter"
context=""/>
</page>
```

No elemento `page`, no atributo `id` deve colocar-se o nome com que se pretende que esta página seja conhecida.

No atributo `url` deve colocar-se sempre "`/servlet/<nome_do_servlet_criado_para_os_tabs_no_passo_4>?` e de seguida

- `PROCESSES=<listagem_de_processos_XML_separados_por_virgulas>, void&`

  Uma lista dos processos XML onde se encontram as páginas que se pretendem que fiquem em *tabs*.

- `MAIN_CLASS=<nome_do_actual_processo_XML>&`
- `BUNDLE=<nome_do_bundle_que_irá_conter_as_traduções_para_os_nomes_dos_tabs>&`
- `PARAMETERS=<colocar_um_parametro_qualquer>&`

  Este pârametro já não é necessário, contudo irá ocorrer uma excepção no *servlet* criado no passo 4 se não se passar um qualquer parâmetro aqui.

- `PAGES=<nome_das_paginas_pretendidas_dos_XML_passados_em_PROCESSES>,void&`

  Ter atenção que para cada XML passado nos `PROCESSES` só pode associar-se uma só página de `PAGES`. Se se pretender, por exemplo, duas páginas do processo *Recruitment.xml*, então este processo XML deve estar presente duas vezes nos `PROCESSES`.

- `TABS=<ids_dos_títulos_de_cada_tab_que_serão_traduzidos_pelo_bundle_passado_em_BUNDLE>&`
- `PAGEJSP=<nome_do_JSP_criado_para_construir_os_tabs_no_passo_3>&`
- `MODULE=<nome_do_directorio_do_webapp_que_ira_conter_o_JSP_criado_no_passo_3>&`
- `RESETTAB=True"`

2º passo: Nas acções de qualquer página que chamem a página em *tab*, colocar como `page_id` a `page` elaborada no passo anterior.

Exemplo: no processo *Recruitment.xml*, na página *RSP_LM*, a acção `action_details` chamava a página *RSP_DE_MV* numa `condition_page`, passando agora a chamar a página *RCSELPROCDETAILSTAB* do exemplo anterior.

3º passo: Implementar um novo JSP para os *tabs*.

Este é o JSP passado no campo `PAGEJSP` do passo 1 e deve ter o seguinte código:

```
<html>
<%@ include file="/Translation/jsp/Translation.jsp" %>
<%
        boolean showTopFrame = false;
        String tabDefinitions = "";
        boolean showObs = false;
%>
<SCRIPT language="JavaScript" src="<%=
pt.ibs.hctt.cf.control.GlobalInfo.getPathWithURI("/js/IBSGenericScripts.js")
%>"></SCRIPT>
<SCRIPT language="JavaScript" src="<%=
pt.ibs.hctt.cf.control.GlobalInfo.getPathWithURI("/js/Scripts.js") %>"></SCRIPT>
<script language="JavaScript" src="<%=
pt.ibs.hctt.cf.control.GlobalInfo.getPathWithURI("/Window/js/WindowArray.js")
%>"></script>
</script>
<head>
<title><%= translate("idRecruitmentModule","IBSRCUI") %></title>
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Expires" content="-1">
</head>
<body marginheight="0" marginwidth="0" topmargin="0" rightmargin="0" leftmargin="0"
bottommargin="0">
<script language=javascript>
function actionClose() {
}
//top frame size (%)
var     heig1=40;
var//lower frame size inside the table that contains the button tabs
        heig2=100;
//table size (%)
var     heig3=100;
// NOTA: heig1+heig3=100
</script>
<table width="100%" height="100%">
        <tr><td>
                <%@ include file="/Tab/jsp/GenericTab.jsp" %>
        </td></tr>
</table>
<input type=hidden name='hidden' value=0>
<input type=hidden name='tipo' value=0>
</body>
</html>
```

O que poderá ser conveniente mudar é o *id* passado no elemento `<title>` que se refere ao título da janela da página em *tabs*, que será traduzido por um *bundle*. Neste caso, é passado o *id* `idRecruitmentModule`, que será traduzido pelo bundle `IBSRCUI` como "Módulo de Recrutamento".

4º passo: Implementar um *servlet*.

O *servlet* implementado pode servir mais do que uma página em *tab*.

Tome-se como exemplo o *servlet* **pt.ibs.hctt.rc.gui.RCSelProcDetailsTabServlet**

O *servlet* para quaisquer páginas em *tab* deve ser idêntico a este, sendo necessário alterar:

- As declarações das variáveis que irão conter os `input_parameters` das páginas que irão ficar contidas em cada *tab* (linha 204).
- A condição que verifica a variável `mainClass`. Essa condição deverá ser alterada de modo a verificar se esta variável recebe o que foi passado no pârametro MAIN_CLASS do XML, no passo 1. Deve alterar-se também os parâmetros aos quais é feito `getParameter` (linha 213).
- As condições que verificam o que está contido na ArrayList `tabsProcess`. Essa condição deverá ser alterada de modo a verificar se esta variável contém os nomes dos processos XML que foram passados no pârametro PROCESSES no passo 1. Deve alterar-se também os pârametros aos quais é feito `setParameter` (linha 234).

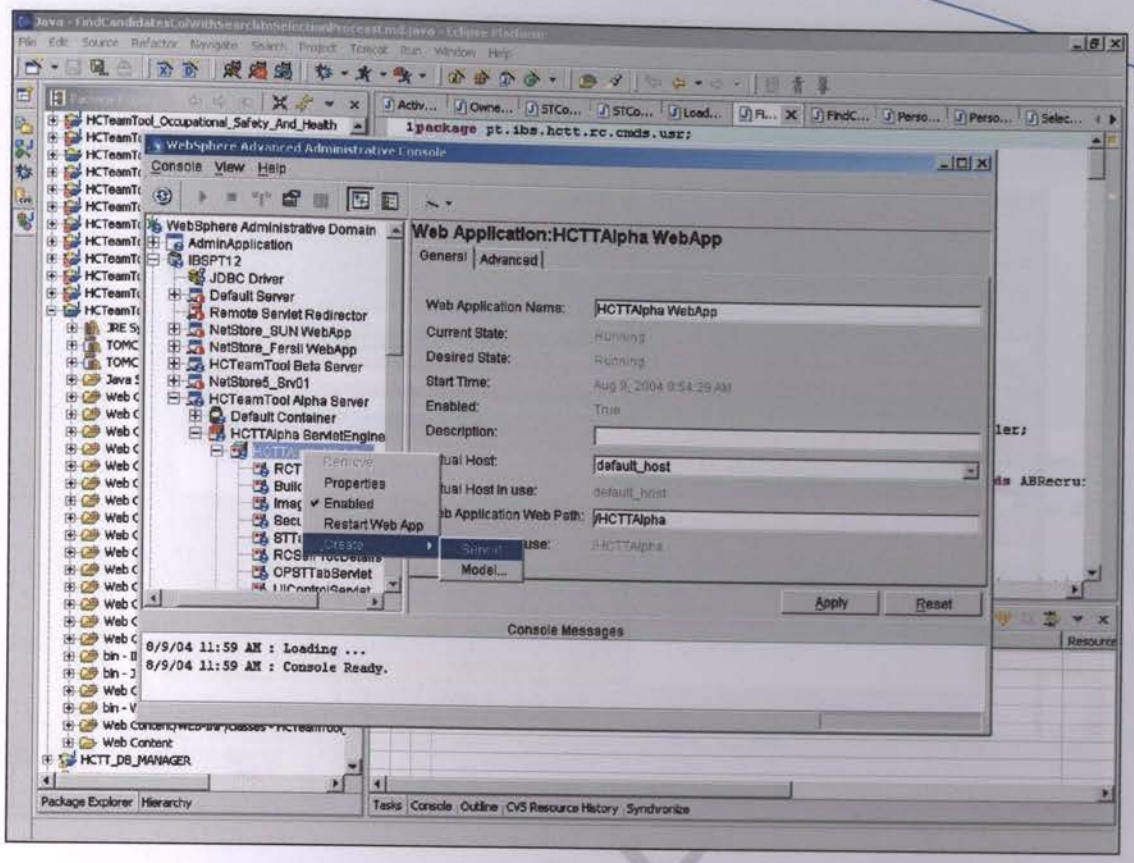
5º passo: Alterar o ficheiro web.xml

O ficheiro *HCTeamTool_Webapp/Web Content/WEB-INF/web.xml* deve ser alterado de modo a incluir o(s) *servlet(s)* criado(s) no passo 4.

6º passo: Inserir o novo *servlet* na *alpha*.

Para inserir cada novo *servlet* na *alpha*, é necessário utilizar a consola de administração do *WebSphere:*

1. *Clickar* com o botão direito do rato em *WebSphere Administrative Domain->IBSPT12-> HCTeamTool Alpha Server->HCTTAlpha Servlet Engine->HCTTAlpha WebApp* e seleccionar ***Create -> Servlet*** (ver figura a seguir).



2. Na janela que é apresentada (ver figura seguinte) inserir os seguintes dados:

   Servlet Name: nome_servlet (exemplo: *RCSelProcDetailsTabServlet*)

   Servlet Class Name: <package>.<nome_servlet> (exemplo: *pt.ibs.hctt.rc.gui.RCTabServlet*)

   Servlet Web Path List: default_host/HCTTAlpha/servlet/<nome_servlet> (exemplo: *default_host/HCTTAlpha/servlet/RCSelProcDetailsTabServlet*)