Universidade do Porto
Faculdade de Engenharia
FEUP

Rui Miguel Monteiro Ferreira

PROCESSO CLÍNICO ELECTRÓNICO, NOVA RELEASE

2005

Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação

Processo Clínico Electrónico, nova *release*
Companhia Portuguesa de Computadores, Healthcare Solutions

Relatório do Estágio Curricular da LEIC 2004/2005

Rui Miguel Monteiro Ferreira

Orientador na FEUP: Prof. Jorge Alves
Orientador na CPCHS: Dr. Alexandre Gomes

Setembro de 2005

Resumo

Este projecto de estágio teve como objectivo a participação no desenvolvimento de uma nova *release* da solução de *Processo Clínico Electrónico* da CPCHS, através da incorporação numa equipa de desenvolvimento criada para o efeito.

Encontrando-se esta aplicação já em ambiente de produção em diversos clientes da CPCHS, o trabalho proposto teve como objectivo último desenvolver uma nova versão da mesma, com uma interface gráfica mais moderna, novas funcionalidades, e recorrendo a um novo paradigma de distribuição centralizado, sob a forma de uma aplicação *web*, contrastando com o modelo *cliente-servidor* convencional da versão actual.

A primeira parte do estágio consistiu no estudo da viabilidade de desenvolvimento de uma aplicação que automatizasse o processo de conversão dos vários componentes da aplicação já existentes para a nova plataforma *web*. A este estudo seguiu-se o desenvolvimento da aplicação propriamente dita, após conclusão favorável da análise realizada. A aplicação, intitulada *Migration Tool*, é capaz de realizar todas as tarefas de conversão passíveis de serem automatizadas, gerando relatórios das intervenções efectuadas, e identificando os casos que podem requerer intervenção manual.

Uma segunda parte do estágio consistiu na implementação de novas funcionalidades na aplicação do *Processo Clínico*, facilitando o tratamento da informação clínica que o sistema existente permite gerir, e tendo sempre em conta a nova plataforma sobre a qual a aplicação vai ser executada.

A par das novas funcionalidades, foram também formuladas algumas directivas para o desenho das novas interfaces gráficas, ao mesmo tempo que se remodelaram as interfaces existentes de acordo com as mesmas directivas.

Findo o período de estágio, conclui-se que o trabalho proposto foi, na sua maior parte, realizado com sucesso. Conseguiu-se dotar a equipa de desenvolvimento da CPCHS de uma ferramenta que automatiza o processo de conversão para a versão *web* na sua quase totalidade. A nova versão da aplicação do *Processo Clínico Electrónico*, para além das novas facilidades de tratamento e consulta de informação, apresenta-se agora ao utilizador com um grafismo bastante mais consistente, agradável e moderno, sendo ainda suficientemente flexível para correr sobre a nova plataforma *web*, ao mesmo tempo que se mantém compatível com o sistema cliente-servidor legado.

Agradecimentos

Gostaria de agradecer, em primeiro lugar, ao Dr. Alexandre Gomes pela oportunidade de estágio que me concedeu e pela orientação prestada na CPCHS. Ao Dr. Paulo Correia e ao Eng. Miguel Henriques, os sinceros agradecimentos pelo apoio diário, pelos sempre prontos e claros esclarecimentos técnicos dados sempre que necessitei, e pela confiança depositada no meu trabalho. Gostava ainda de agradecer aos meus colegas de estágio da FEUP, da Universidade do Minho e da Faculdade de Ciências, pelo companheirismo demonstrado, e pela forma como tornaram mais fácil a adaptação ao novo mundo do trabalho. A todos os outros agradeço também pelo bom ambiente que sempre criaram na CPCHS, proporcionando excelentes condições para que pudesse desenvolver o meu trabalho.

Ao professor Jorge Alves, deixo o meu agradecimento pelo apoio e disponibilidade que desde o início demonstrou, e por me ter feito sentir que tinha sempre alguém a quem recorrer em caso de necessidade. Não queria deixar de agradecer também ao professor Raul Vidal, cujo empenho na organização dos estágios da LEIC deve ser reconhecido.

Um obrigado muito especial à Tânia, por tudo.

Gostaria ainda de referir e agradecer o apoio financeiro do PRODEP.

Índice de Conteúdos

1 Introdução

1.1 Apresentação da Empresa

A Companhia Portuguesa de Computadores, Healthcare Solutions, S.A. (CPCHS) foi criada em 2001 no âmbito de uma reestruturação da empresa Companhia Portuguesa de Computadores, Informática e Sistemas, Lda. (CPCis), actuando exclusivamente na área da Saúde.

A CPCis, a empresa na qual a CPCHS teve origem, foi fundada em 1984, iniciando a sua actividade na área da saúde em 1997. Em 1998 aumentou o seu leque de soluções ao adquirir a divisão de saúde da empresa Medidata, ao mesmo tempo que contratou pessoal com vasta experiência na área. Como resultado de um crescimento contínuo e sustentado, ocorre então em 2001 a divisão da CPCis, aparecendo no mercado a CPCHS, já com várias soluções desenvolvidas e uma posição consolidada no mercado.

A própria CPCHS tem sofrido recentemente um crescimento considerável, possuindo instalações no Porto e em Lisboa, e contando já com mais de 90 colaboradores afectados às áreas de administração, comercial, desenvolvimento e investigação.

A empresa está hoje presente em mais de 80 unidades de saúde, trabalhando com os principais grupos privados, e actua ainda em vários projectos em parceria com o Instituto de Gestão Informática e Financeira da Saúde (IGIF) e a Unidade de Missão para os Hospitais S.A.

1.2 Enquadramento do Projecto na Empresa

O *Processo Clínico* é uma das várias aplicações desenvolvidas pela CPCHS. Inserido na área de Gestão Hospitalar, foi desenvolvido com a ambição de se tornar a aplicação por excelência utilizada pelo pessoal médico.

À data de redacção deste relatório, o *Processo Clínico* encontra-se em utilização efectiva em seis instituições hospitalares:

- Hospital CUF Descobertas (HCD)
- Hospital CUF Infante Santo (HCIS)
- Clínica CUF de Alvalade (CCA)
- Hospitais Universitários de Coimbra (HUC) – algumas áreas
- Hospital da Força Aérea de Lisboa (HFA)
- Centro de Apoio ao Desenvolvimento Infantil (CADin)

As três primeiras instituições (CCA, HCD e HCIS) pertencem ao grupo de hospitais privados José de Mello Saúde (JMS).

Para além destas seis instituições, a CPCHS está actualmente a preparar a instalação do *Processo Clínico* na Santa Casa da Misericórdia de Vila do Conde (SCMVC).

O *Processo Clínico* permite aos médicos gerir toda a informação associada à sua actividade, ao mesmo tempo que acedem a resumos de todos os dados relevantes acerca dos pacientes, facultando a estes profissionais uma visão ampla e completa de todo o histórico existente.

Tratando-se de profissionais bastante exigentes, com horários bastante preenchidos e, portanto, com pouca disponibilidade para grandes períodos de adaptação à utilização de uma aplicação nova, o *Processo Clínico* viu aqui uma considerável resistência à sua introdução no dia a dia dos médicos. Um dos desafios deste projecto de estágio é precisamente dotar esta aplicação de novas interfaces gráficas, mais modernas, simples e intuitivas, tentando torná-las ainda mais práticas e funcionais para os utilizadores a que se destinam.

À semelhança de todas as aplicações de Gestão Hospitalar da CPCHS, o *Processo Clínico Electrónico* assenta sobre a plataforma *Oracle Forms 6i*, uma arquitectura cliente-servidor com base de dados centralizada que disponibiliza um *runtime* para as aplicações criadas. Esta arquitectura tem a grande desvantagem de obrigar à instalação de vários componentes em todas as máquinas clientes, exigindo ainda vários passos de configuração manual das mesmas, representando um esforço considerável num processo de actualização. É neste contexto que é definido um segundo objectivo para este projecto de estágio, o de fazer com que a nova versão do *Processo Clínico*, e mesmo de toda a Gestão Hospitalar, seja executada sobre uma nova arquitectura, totalmente centralizada, a plataforma *Oracle Forms 9i*. Esta permite a utilização de *thin clients*, de modo a que as máquinas clientes necessitem apenas de um *browser web* para aceder a todas as aplicações. As actualizações necessárias à instalação de novas versões serão então efectuadas num único ponto central.

1.3 Organização do Estágio

Todo o trabalho efectuado no âmbito deste estágio foi levado a cabo nas instalações da CPCHS, no Porto. À semelhança de todos os estágios oferecidos pela CPCHS, este começou com um período de formação, no qual foram apresentados ao estagiário as metodologias de trabalho da empresa, as ferramentas e tecnologias utilizadas, e os produtos já desenvolvidos, numa perspectiva de dotar o estagiário de algum conhecimento da área de negócio. Este período, que teve a duração total de 3 semanas, incluiu formação na utilização das ferramentas *Oracle Forms Builder 6i, Oracle Reports Builder 6i*, assim como uma introdução à linguagem de programação utilizada nestas ferramentas, o PL/SQL.

Seguiu-se um período de introdução ao processo de conversão dos *forms* para a versão *web*, onde foi apresentado ao estagiário o trabalho e investigação já efectuados nesta área. Este incluiu uma fase de conversão manual de vários *forms* existentes, com o objectivo de apreender todos os conceitos envolvidos no processo, como preparação para a fase seguinte.

Seguidamente, iniciou-se um período de estudo das tecnologias disponíveis para o desenvolvimento da aplicação de migração de *forms* (*Migration Tool*) que, complementado com o estudo do processo de migração, permitiu concluir que seria viável o desenvolvimento da aplicação.

A fase seguinte consistiu no desenvolvimento da *Migration Tool*. O protótipo funcional implementado permitiu, no final desta fase, efectuar a conversão automática das várias centenas de *forms* existentes passíveis de serem convertidos desta forma, identificando ao mesmo tempo aqueles que necessitariam de intervenção manual.

A fase seguinte consistiu na reestruturação do *Processo Clínico*, tendo sido desenvolvidas novas funcionalidades a fim de tornar a aplicação mais consistente e fácil de utilizar, e criado um novo estilo para as interfaces gráficas, tendo este sido implementado em muitos dos *forms* do *Processo Clínico*. É de salientar o facto de este processo de reestruturação ter sido efectuado de forma a tornar o *Processo Clínico Electrónico* compatível com a plataforma *Oracle Forms 6i*, ao mesmo tempo que o tornava disponível na nova versão *web*.

1.4 Organização do Relatório

Neste relatório, começa-se por descrever os problemas a resolver, com algum detalhe, no capítulo 2. Nesse capítulo justifica-se a necessidade de desenvolver a *Migration Tool* e de proceder aos novos desenvolvimentos no *Processo Clínico*, apresentando ainda as interligações entre os dois problemas.

O capítulo 3 apresenta as tecnologias utilizadas, nomeadamente aquelas usadas no desenvolvimento do protótipo da *Migration Tool*.

No capítulo 4, é feita a especificação da *Migration Tool*. Os desenvolvimentos do *Processo Clínico* não são ainda abordados neste capítulo.

O capítulo 5, à semelhança do anterior, refere-se unicamente à *Migration Tool*, descrevendo desta vez o protótipo implementado, analisando os problemas encontrados e respectivas soluções.

O capítulo 6 consiste na avaliação de resultados, onde se apresentam os resultados dos testes efectuados com a *Migration Tool*, antecipando já algumas conclusões em relação aos mesmos.

As conclusões de todo o trabalho encontram-se no capítulo 7. São apresentadas aqui conclusões acerca da *Migration Tool*, e também acerca do trabalho desenvolvido na aplicação do *Processo Clínico Electrónico*, indicando os pontos onde, futuramente, o trabalho pode ser melhorado.

No Anexo A é feita uma apresentação à ferramenta de desenvolvimento *Forms Builder*, descrevendo-se com algum detalhe a estruturação dos *forms*. A leitura deste anexo é muito importante se o leitor não estiver familiarizado com os conceitos inerentes ao desenvolvimento de aplicações em *Oracle Forms Builder*.

A leitura do Anexo B é obrigatória, pois descreve detalhadamente o problema da migração de *forms* para a plataforma *Oracle Forms 9i*, independentemente do facto de haver uma ferramenta a automatizar o processo ou não. É aqui que são justificados os vários passos que, nos capítulos anteriores, se diz ser necessário a *Migration Tool* realizar.

O trabalho efectuado no *Processo Clínico Electrónico* é dividido em duas partes. No Anexo C apresentam-se as directivas formuladas para o desenho das novas interfaces gráficas, com vários exemplos do trabalho feito. O Anexo D descreve as funcionalidades do *Processo Clínico*, apresentando o trabalho efectuado nesta área durante o estágio.

Finalmente, o Anexo E contém um pequeno guia de instalação da *Migration Tool*.

2 Análise do Problema

Este capítulo começa por apresentar uma análise detalhada do contexto técnico das soluções desenvolvidas na CPCHS, sendo esta informação útil para se compreender na sua totalidade o problema aqui tratado.

O projecto de estágio é aqui claramente dividido em duas partes, correspondendo cada uma delas a um problema mais ou menos independente, pelo menos até certo ponto. Este capítulo descreve cada um desses problemas detalhadamente, deixando clara a necessidade deste projecto de estágio. Por fim, tece algumas considerações acerca da interligação entre os dois problemas analisados.

2.1 Apresentação geral das técnicas de desenvolvimento e distribuição da CPCHS

A CPCHS desenvolve a generalidade das suas aplicações sobre a plataforma *Oracle Forms*, utilizando actualmente a versão 6i da mesma. Esta plataforma disponibiliza, entre outros, uma ferramenta de desenvolvimento do tipo RAD (*Rapid Application Development*) denominada Oracle *Forms Builder 6i*[1], e um ambiente de execução para os programas criados, o *Oracle Forms Runtime*. Os programas criados não são, portanto, ficheiros executáveis nativos de determinado sistema operativo, mas sim ficheiros binários interpretados e executados pelo *Forms Runtime* (ficheiros com extensão *fmx*).

Num ambiente de produção, existe um repositório central onde são colocados todos os componentes executáveis da aplicação (ficheiros *fmx* correspondentes a *forms*, entre outros), e um servidor com a base de dados. O repositório de ficheiros executáveis e a base de dados podem, eventualmente, estar localizados na mesma máquina. Cada um dos postos de trabalho possui uma instalação do *Forms Runtime 6i*, e tem *mapeada* uma *network drive* que permite aceder aos ficheiros executáveis do servidor de uma forma transparente para a aplicação. Quando a aplicação é executada por um utilizador, determinado ficheiro executável é acedido através da *drive* mapeada, sendo depois interpretado pelo ambiente de *runtime* local. Este esquema permite ter um único repositório central de ficheiros executáveis, que podem facilmente ser substituídos por novas versões sempre que necessário.

Infelizmente, apenas novas versões das aplicações da CPCHS que *correm* sobre a mesma versão do *runtime* podem ser actualizadas desta forma. Havendo necessidade de desenvolver novas versões das aplicações que exijam uma nova versão do *runtime* da Oracle, o processo de actualização torna-se muito moroso. Na eventualidade de uma situação como a descrita, seria necessário, no mínimo, instalar em todas as máquinas clientes a nova versão do *Forms Runtime*, de uma só vez, para que todos os clientes fossem capazes de interpretar correctamente o novo formato dos ficheiros executáveis que então passariam a estar disponíveis no repositório. Este cenário leva a uma resistência muito grande por parte da CPCHS em actualizar o *runtime* instalado nas máquinas clientes, cujo número é da ordem das várias centenas. Entre outros inconvenientes, isto obriga a equipa de desenvolvimento a ter

[1] A unidade de programação principal do *Forms Builder* é o *form*, que corresponde a um ecrã da aplicação, com todo o código que implementa a sua lógica de funcionamento. Os ficheiros fonte do *Forms Builder* têm a extensão *fmb*, correspondendo cada ficheiro *fmb* a um *form*. Quando compilados, os *forms* existem sob a forma de ficheiros *fmx*.

em conta algumas falhas existentes na versão do *runtime* utilizada, falhas essas entretanto corrigidas pela Oracle em versões posteriores, disponibilizadas sob a forma de *patches* que não podem ser facilmente instalados. Mais ainda, a aquisição das aplicações da CPCHS por parte de um eventual novo cliente (aqui, "cliente" refere-se uma entidade que compra serviços ou produtos a outra) exige um esforço enorme de configuração, já que é necessário instalar o *runtime* em todas as máquinas clientes (aqui, "cliente" refere-se a uma máquina que faz parte de um sistema informático distribuído numa arquitectura cliente-servidor, usufruindo dos serviços disponibilizados pela máquina que desempenha o papel de servidor), e proceder a vários passos de configuração manual em cada uma delas.

2.2 A migração para a plataforma *Oracle Forms 9i*

É neste cenário que surge a necessidade de se ter um sistema completamente centralizado, onde todas as actualizações possam ser feitas com segurança num único local. Quer se trate de actualizações aos ficheiros executáveis desenvolvidos pela CPCHS que correm sobre o mesmo *runtime* já usado em produção, quer se trate mesmo de actualizações ao ambiente *runtime* disponibilizadas pela Oracle.

A solução para este problema está já parcialmente encontrada. A própria Oracle disponibiliza na versão 9i do *Forms* a possibilidade de *correr* as aplicações sem qualquer componente local instalado, para além de um convencional *browser web*[2]. Na nova versão desta ferramenta, é possível configurar o *runtime* numa máquina servidora (e não em cada uma das máquinas clientes) e, através do *Internet Application Server* (IAS), o servidor aplicacional da Oracle, disponibilizar o acesso das máquinas clientes aos componentes da aplicação[3]. Neste novo cenário, é possível actualizar os executáveis das aplicações da CPCHS facilmente, e proceder a quaisquer actualizações ao ambiente de *runtime* com a mesma facilidade. Se, por exemplo, a Oracle lançar uma actualização para o *Forms 9i* com pequenas correcções, esta só tem que ser instalada no servidor, com grande probabilidade de não ser necessária qualquer outra alteração, já que a nova versão manter-se-á compatível com os executáveis dos *forms* existentes.

Começar imediatamente a utilizar o *Oracle Forms 9i* não é, contudo, uma opção viável para a CPCHS e seus clientes. Enquanto que a transição para uma nova versão do *Forms* é pacífica quando se trata de uma versão com pequenas alterações de correcção, a transição da versão 6i para a 9i é um salto enorme. São muitas as funcionalidades que deixaram de ser suportadas, ou substituídas por outras tecnologias. A título de exemplo, a CPCHS recorreu a alguns controlos OCX nas suas aplicações, como forma de implementar funcionalidades mais avançadas não disponíveis na plataforma. Este tipo de controlos já não é suportado na versão 9i, devendo agora ser utilizados os *JavaBeans* como forma de adicionar componentes externos.

---

[2] Na verdade, é necessário ter instalada localmente uma máquina virtual Java compatível com os *Applets* disponibilizados pelo servidor, e um *plugin* para que o *browser utilizado* seja capaz de os executar. Mas, se ainda não o tiver sido, esta instalação é feita automaticamente aquando da primeira execução da aplicação, bastando para isso que o utilizador dê permissão para tal. Na CPCHS, é disponibilizado no servidor *web* o *JInitiator*, que inclui a máquina virtual Java e o *plugin* para o Internet Explorer.

[3] O *Oracle Forms 9i* é baseado na plataforma Java. Os ficheiros com o código fonte são compilados em ficheiros *fmx* para a versão 9. Quando estes são invocados para execução, o que o *runtime* faz é, na verdade, criar classes Java que constituem um Java *Applet* a partir dos executáveis (*fmx*), sendo este *Applet* disponibilizado normalmente através do servidor aplicacional IAS.

Desenvolver substitutos para todos os componentes que já não são suportados na nova plataforma é um processo que leva muito tempo, e o desenvolvimento dos *forms* não pode parar à espera que todo o processo de migração para a nova versão esteja concluído. Simultaneamente, muitos clientes actuais não estão interessados em migrar para a nova versão *web*, necessitando apenas de pequenas adições e/ou correcções aos *forms* que já utilizam.

Este é o problema que levou a CPCHS a procurar uma forma gradual de proceder à migração para a nova versão, preferencialmente com uma ferramenta que automatizasse o processo.

Numa primeira fase, deveria ser estudada a possibilidade de desenvolver tal ferramenta, tendo em conta que esta deveria ser capaz de modificar os ficheiros *fmb* que constituem o código fonte dos *forms*, atendendo ao facto de estes ficheiros se encontrarem em formato binário (o que torna muito difícil a sua manipulação directa). Preferencialmente, a aplicação deveria também ser capaz de compilar os *forms* para as duas plataformas em utilização, o *Forms 6i* e o *Forms 9i*.

Outro requisito desta ferramenta é a flexibilidade de configuração. Apesar de o *Processo Clínico* ser a aplicação que impulsionou a migração para a nova plataforma *web*, e de o desenvolvimento da ferramenta de migração estar inserido no projecto do novo *Processo Clínico*, esta ferramenta deverá ser facilmente adaptável às restantes aplicações da CPCHS, principalmente a toda a restante área de Gestão Hospitalar, para a qual está também prevista a migração para a plataforma 9i.

2.3 As novas funcionalidades do *Processo Clínico Electrónico*

Outro problema do *Processo Clínico*, na sua versão à data de início deste estágio, era a falta de um *look-and-feel*[4] transversal a toda a aplicação. Por se tratar de uma aplicação muito grande, cuja implementação foi sendo feita ao longo de muito tempo, por vários programadores diferentes, sem que existissem regras bem definidas acerca de qual deveria ser o aspecto dos ecrãs, o tipo de controlos gráficos a utilizar em cada situação, os ícones a usar para cada função dos botões, etc., o resultado era uma imagem inconsistente do *Processo Clínico*, que necessitava de ser uniformizada. Um caso simples mas bastante ilustrativo deste problema é o aspecto das *tooltips*[5] dos botões. Vários botões utilizam cores diferentes para o fundo da caixa em que a descrição aparece, assim como o próprio tipo de letra das *tooltips* varia de botão para botão.

Para além disso, o simples facto de os *forms* terem um grafismo não muito moderno apelava à necessidade de uma remodelação (ver figura 1).

Em termos de funcionalidade, havia também falhas a tratar, como a forma de navegação inconsistente e nem sempre fácil ou intuitiva; inconsistência da informação apresentada em locais diferentes da aplicação; fraca integração entre os vários componentes da aplicação; etc.

---

[4] O *look-and-feel* refere-se aos aspectos do desenho de uma interface gráfica, em termos de cores, formas, disposição dos componentes, tipo de letra, etc. (o "*look*"); e ao comportamento dos elementos dinâmicos, tais como botões, caixas de texto, menus, etc. (o "*feel*").

[5] A *tooltip* é uma pequena descrição da função de um controlo gráfico (um botão, por exemplo), que é apresentada ao utilizador sobre a interface gráfica quando este coloca o ponteiro do rato sobre o respectivo controlo.

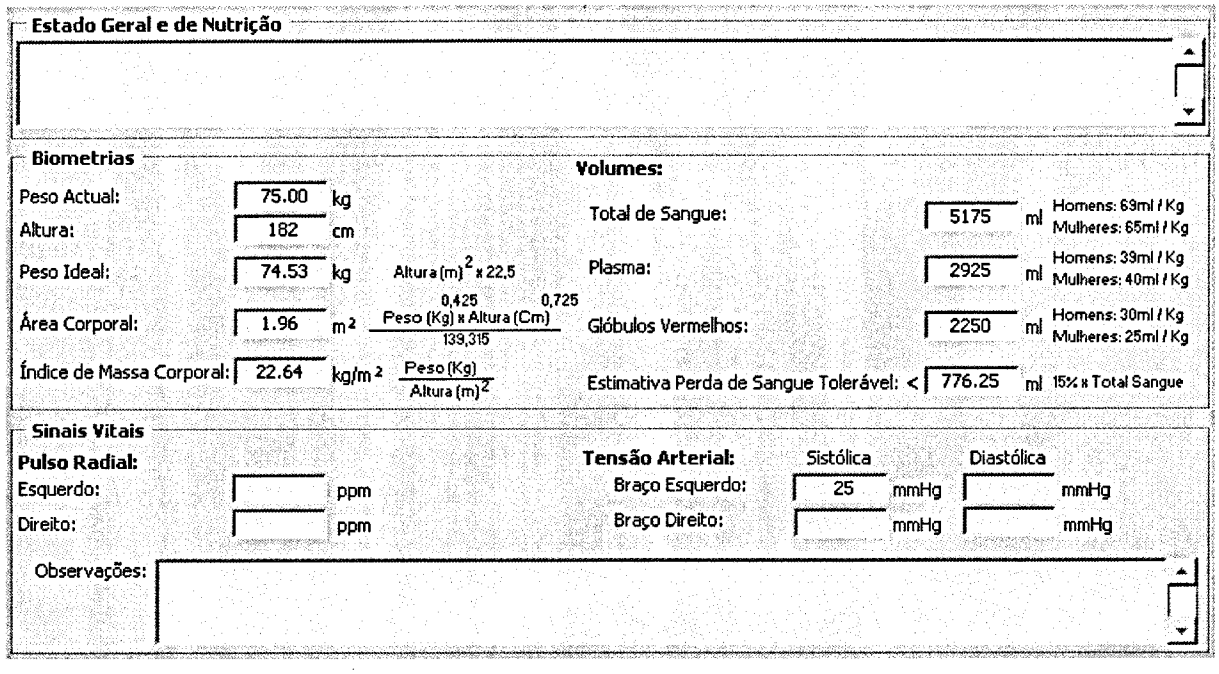


Fig.1 *– Exemplo de ecrã cuja interface gráfica, bastante simples e completa em termos de funcionalidade (os campos que apresentam uma descrição a cor azul são calculados automaticamente quando os valores do peso actual e altura são alterados), tem um aspecto pouco moderno e apelativo.*

2.4 O problema da migração gradual a par de novos desenvolvimentos

Finalmente, havia a necessidade de considerar sempre os dois grandes problemas (a migração para a plataforma *web* e a reestruturação do *look-and-feel e funcionalidades*) em conjunto, e nunca isoladamente.

Por um lado, a ferramenta de migração deveria ser capaz de converter todos os *forms* para a plataforma *Oracle Forms 9i*, mas tendo sempre em conta que os *forms* já convertidos iriam continuar a ser reestruturados. Não é viável manter duas versões dos *forms* depois destes terem sido convertidos (uma versão para a plataforma 6i e outra para 9i). Por isso, a aplicação de migração deve fazer alterações às fontes dos *forms* de modo a que estes continuem compatíveis com a versão 6i, para que as alterações entretanto efectuadas possam ser utilizadas pelos clientes que ainda usam essa versão, ao mesmo tempo que ficam disponíveis na plataforma *web*.

Por outro lado, a reestruturação deve ser feita de forma consciente, tendo sempre em conta que as novas funcionalidades implementadas e alterações efectuadas devem ser suportadas nas duas plataformas, para evitar a necessidade de ter novos controlos específicos de cada uma das versões (isto só deve acontecer para os controlos que já estão desenvolvidos e que não são suportados em *Oracle Forms 9i*).

3 Revisão Tecnológica

A migração da plataforma *Oracle Forms 6i* para a plataforma *Oracle Forms 9i,* apesar de ser um problema com o qual outros se hão-de ter deparado, é um para o qual não existem soluções concretas desenvolvidas. Isto porque se trata de um problema demasiado específico de cada caso. Enquanto que a migração de *forms* simples poderia ter uma solução genérica, a realidade das empresas impede que isto aconteça em aplicações como o *Processo Clínico* (já para não falar de toda a Gestão Hospitalar).

Seguindo uma estratégia de reutilização de código, todos os *forms* da CPCHS são baseados num outro *form* (temp.*fmb*), que serve o propósito de disponibilizar controlos e respectivo código comuns a todos os *forms*, desempenhando o papel de uma *template.* Muitas bibliotecas de funções e procedimentos foram também desenvolvidas para promover a reutilização de código. O processo de conversão de *forms* na CPCHS deve ter isto em conta. Converter um *form*, entre outras alterações, implica substituir a *template* utilizada, recorrendo a outra desenvolvida especificamente para a plataforma 9i, e substituir muitas das bibliotecas de funções às quais cada *form* faz referência. Muitas, mas não todas, porque algumas necessitavam apenas de pequenas alterações, continuando compatíveis com a plataforma 6i. Outras, contudo, sofreram alterações de tal ordem que as tornaram incompatíveis com a plataforma 6i, obrigando à criação de novas bibliotecas.

Este problema é apenas um dos muitos encontrados no processo de migração, e é aqui descrito com o único propósito de demonstrar o quanto este processo é específico da realidade de cada empresa.

Justificado o facto de não existirem soluções para o problema, vão ser analisadas na secção seguinte as tecnologias disponíveis para implementar a aplicação pretendida.

A secção 3.2 analisará com algum detalhe a tecnologia escolhida para implementação da aplicação *Migration Tool*, a JDAPI.

3.1 Tecnologias utilizadas

Uma abordagem possível ao problema da conversão para a plataforma *web* seria desenvolver uma aplicação, em qualquer linguagem de programação que permitisse o acesso a ficheiros, que lesse o conteúdo dos ficheiros *fmb*, procedesse às respectivas modificações, e guardasse as alterações efectuadas. A compilação do código fonte (o ficheiro *fmb*) poderia ser levada a cabo recorrendo às ferramentas de linha de comandos disponíveis no *Forms Builder 6* e *Forms Builder 9*. No entanto, os ficheiros *fmb* encontram-se guardados num formato binário, e não em formato de texto. Para além disso, a especificação do formato não é pública. Isto torna esta solução praticamente inviável. Mais ainda, considerando a alternativa existente, descrita nos parágrafos seguintes, esta abordagem foi imediatamente posta de parte.

A plataforma *Oracle Forms*, principalmente na versão 9i, recorre em grande parte à tecnologia Java. O ambiente de *runtime* através da Internet que esta plataforma disponibiliza, por exemplo, é totalmente baseado nesta tecnologia, nomeadamente em *Java Applets*. Na mesma filosofia de ligação com Java, a Oracle disponibiliza nas instalações do *Forms Builder 9i* uma versão do Java 2 SDK, até porque vários componentes desta ferramenta utilizam-na. Também como parte integrante do *Forms Builder 9i*, é disponibilizada uma API (*Application*

Programming Interface), denominada JDAPI (***Java Design-time API***), que permite ler, criar, manipular, guardar e compilar aplicações em *Oracle Forms*. Esta API não é mais do que uma interface (*wrapper*) para a *Forms C API*, a API utilizada pelo próprio *Forms Builder* para manipular os *forms*. A JDAPI apresenta-se assim como uma poderosa ferramenta para manipular os *forms* existentes, permitindo realizar praticamente todas as tarefas que o próprio *Forms Builder* é capaz.

A JDAPI disponível numa instalação do *Forms Builder 9i* permite o acesso a *forms* criados em versões anteriores do *Forms Builder*, nomeadamente aqueles desenvolvidos na CPCHS em *Forms Builder 6i*. Com esta versão da API é possível abrir esse *forms*, alterá-los, guardá-los e compilá-los. Infelizmente, ao guardar os ficheiros *fmb* alterados com esta API, estes deixam de ser compatíveis com o *Forms Builder 6i*. Isto não é viável, pois o desenvolvimento deve continuar a ser feito em *Forms Builder 6i*, de modo a que as alterações e novas funcionalidades entretanto desenvolvidas continuem a poder ser utilizadas em todos os clientes que possuem a plataforma 6i.[6]

A versão do *Forms Builder 6i* utilizada na CPCHS, a 6.0.8.13.0, não disponibiliza esta API. Existe, contudo, outra versão do *Forms Builder 6i*, a 6.0.8.21.3, disponível através de um *patch* que faz o *upgrade*, que já a suporta (assim como qualquer outra versão posterior a esta).

A possibilidade de utilização destas duas versões da JDAPI (a que é instalada com o *Forms 9i* e a versão disponibilizada com o *patch* referido para o *Forms 6i*) constitui a base tecnológica necessária que levou à decisão de avançar com o desenvolvimento da *Migration Tool*.

A secção seguinte deste documento apresenta esta API com mais algum detalhe. Note-se que grande parte do que aqui é descrito é fruto da investigação levada a cabo na análise da viabilidade de desenvolvimento da *Migration Tool*. Trata-se, portanto, de material resultante do projecto de estágio, e não informação que estava disponível para consulta à priori (à excepção da especificação da API).

3.2 JDAPI

Existem diferenças evidentes no funcionamento das duas versões da API. A versão 9i guarda os ficheiros *fmb* e cria ficheiros *fmx* em formatos suportados pela plataforma *Oracle Forms 9i*, enquanto a versão 6i cria ficheiros *fmb* compatíveis com ambas as plataformas (porque o *Forms Builder* 9i é retro compatível), e ficheiros *fmx* suportados apenas pela plataforma 6i (o *Forms Runtime 9i* não é compatível com os *fmx* da versão 6i).

No entanto, as diferenças existem apenas ao nível do comportamento da API acima descrito. Ambas as versões são distribuídas num ficheiro JAR (Java Archive)[7] com exactamente a mesma estrutura interna, sendo a assinatura das classes que constituem a API a mesma. Isto significa que uma aplicação desenvolvida com uma das versões pode facilmente passar a utilizar a outra, bastando para isso que seja compilada de novo, desta vez fazendo referência ao ficheiro JAR da nova versão da API. Este ficheiro tem o nome *f60jdapi.jar* na versão 6i, e *f90jdapi.jar* na versão 9i.

---

[6] À data de redacção deste relatório, todos os clientes do *Processo Clínico* e Gestão Hospitalar utilizam a plataforma *Oracle Forms 6i*. A migração para a nova plataforma *web* ainda não está concluída.

[7] O formato de ficheiros JAR é utilizado para distribuir aplicações ou extensões para serem usadas por outras aplicações Java. No caso da JDAPI, um único ficheiro JAR é utilizado para distribuir facilmente toda a API. Basta fazer referência a este ficheiro aquando da compilação de qualquer aplicação que utilize JDAPI.

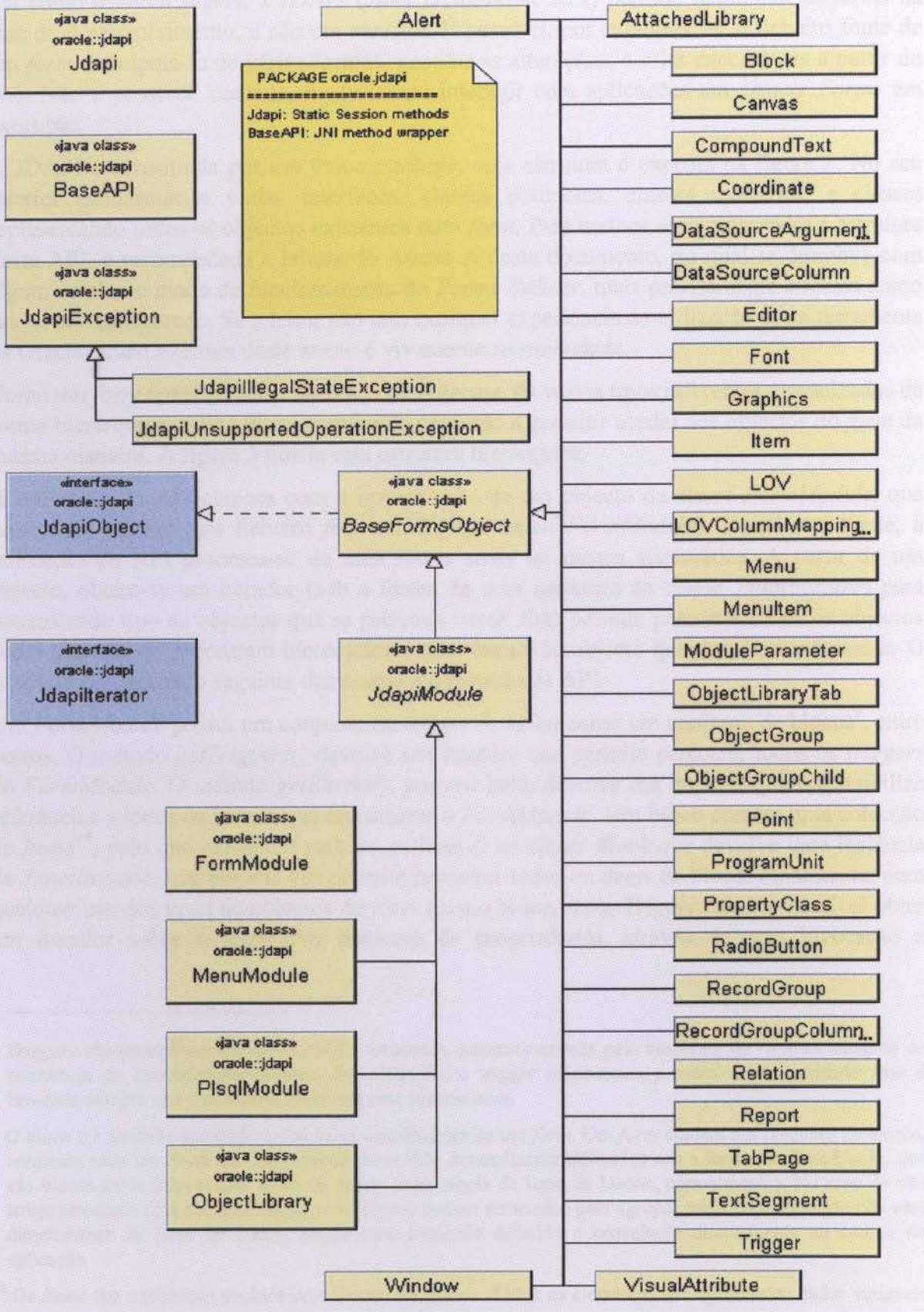


Fig.2 *– Diagrama de classes da JDAPI. Este diagrama expõe a estrutura do package que constitui a JDAPI.*

Tal como o nome sugere, a JDAPI (*Java Design-time API*) permite manipular os *forms* na fase de desenvolvimento, e não em *runtime*. É possível, por exemplo, ler o ficheiro fonte de um *form*, manipulá-lo de várias formas, guardar as alterações, e criar executáveis a partir do *fmb*. Não é possível, contudo, controlar ou interagir com aplicações em *Oracle Forms* em execução.

A JDAPI é constituída por um único *package*, cuja estrutura é exposta na figura 2. No seu interior encontram-se várias interfaces, classes abstractas, classes utilitárias, e classes representando todos os objectos existentes num *form*. Para melhor se compreender a estrutura desta API, é recomendada a leitura do Anexo A deste documento, no qual se descreve com algum detalhe o modo de funcionamento do *Forms Builder*, mais precisamente a forma como um *form* é estruturado. Se o leitor não tem qualquer experiência de utilização desta ferramenta da Oracle, então a leitura deste anexo é vivamente recomendada.

Como um *form* consiste numa colecção de objectos, de vários tipos diferentes, organizados de forma hierárquica, a JDAPI foi construída de modo a permitir aceder aos objectos do *form* da mesma maneira. A figura 3 ilustra esta estrutura hierárquica.

A utilização da API começa com a instanciação de um objecto da classe *FormModule*, que representa o *form* (um ficheiro *fmb* contém um único *FormModule*). Daqui em diante, a utilização da API processa-se de uma forma mais ou menos sistemática. A partir de um objecto, obtém-se um iterador (sob a forma de uma instância da classe *JdapiIterator*) para determinado tipo de objectos que se pretenda tratar. Este permite percorrer todos os objectos desse tipo que se encontram hierarquicamente abaixo do objecto que devolveu o iterador. O exemplo do parágrafo seguinte demonstra a utilização da API.

Um *FormModule* possui um conjunto de *triggers*[8], assim como um conjunto de blocos[9], entre outros. O método *getTriggers()* devolve um iterador que permite percorrer todos os *triggers* do *FormModule*. O método *getBlocks()*, por seu lado, devolve um iterador que disponibiliza referências a todos os blocos que constituem o *FormModule*. Um bloco contém uma colecção de *Items*[10], pelo que existe um método *getItems()* na classe *Block* que devolve uma instância de *JdapiIterator*, que por sua vez permite percorrer todos os *items* do bloco. Finalmente, para qualquer um dos tipos de objectos do *form* (como bloco, *Item*, *Trigger, etc.),* é possível obter um iterador sobre o respectivo conjunto de propriedades, através de uma invocação a

---

[8] *Triggers* são procedimentos em PL/SQL, invocados automaticamente pelo ambiente de *runtime* aquando da ocorrência de determinados eventos. Por exemplo, o trigger *when-mouse-pressed* de determinado *item* é invocado sempre que o utilizador pressiona esse mesmo *item*.

[9] O bloco é a unidade agrupadora dos *items* constituintes de um *form*. Um *form* contém um conjunto de blocos, contendo cada um deles um conjunto de *items*. São normalmente utilizados sob a forma de *DataBlocks*, que são blocos associados a uma fonte de dados (uma tabela da Base de Dados, normalmente). No caso de não terem associada uma fonte de dados, estes blocos podem ser usados para agrupar *items* cujo conteúdo não vem directamente da Base de Dados, sendo eses conteúdo definido e consultado directamente no código da aplicação.

[10] Os *Items* são a principal unidade constituinte dos *forms*. Todos os elementos que apresentam dados variáveis num *form* são *items*, podendo aparecer sob a forma de caixas de texto editáveis (*items* do tipo *Text Item*), caixas de texto não editáveis (*Display Items*), listas de elementos (*List Items*), caixas de texto com *drop-down list* (*List Items* do tipo *Combo-Box*), *radio buttons*, *check boxes*, etc. Os *items* podem estar associados a um campo de uma fonte de dados (uma coluna de uma tabela, tipicamente), permitindo automaticamente apresentar o conteúdo da fonte de dados, alterá-lo e inserir novos dados (quando a fonte de dados é uma tabela, estas operações correspondem, respectivamente, a comandos *SELECT*, *UPDATE* e *INSERT*).

getMetaProperties(). Os objectos percorridos por este iterador são instâncias de *JdapiMetaProperty*. Desta forma, consegue-se aceder e modificar qualquer propriedade de qualquer objecto que faça parte da constituição de um *form*.

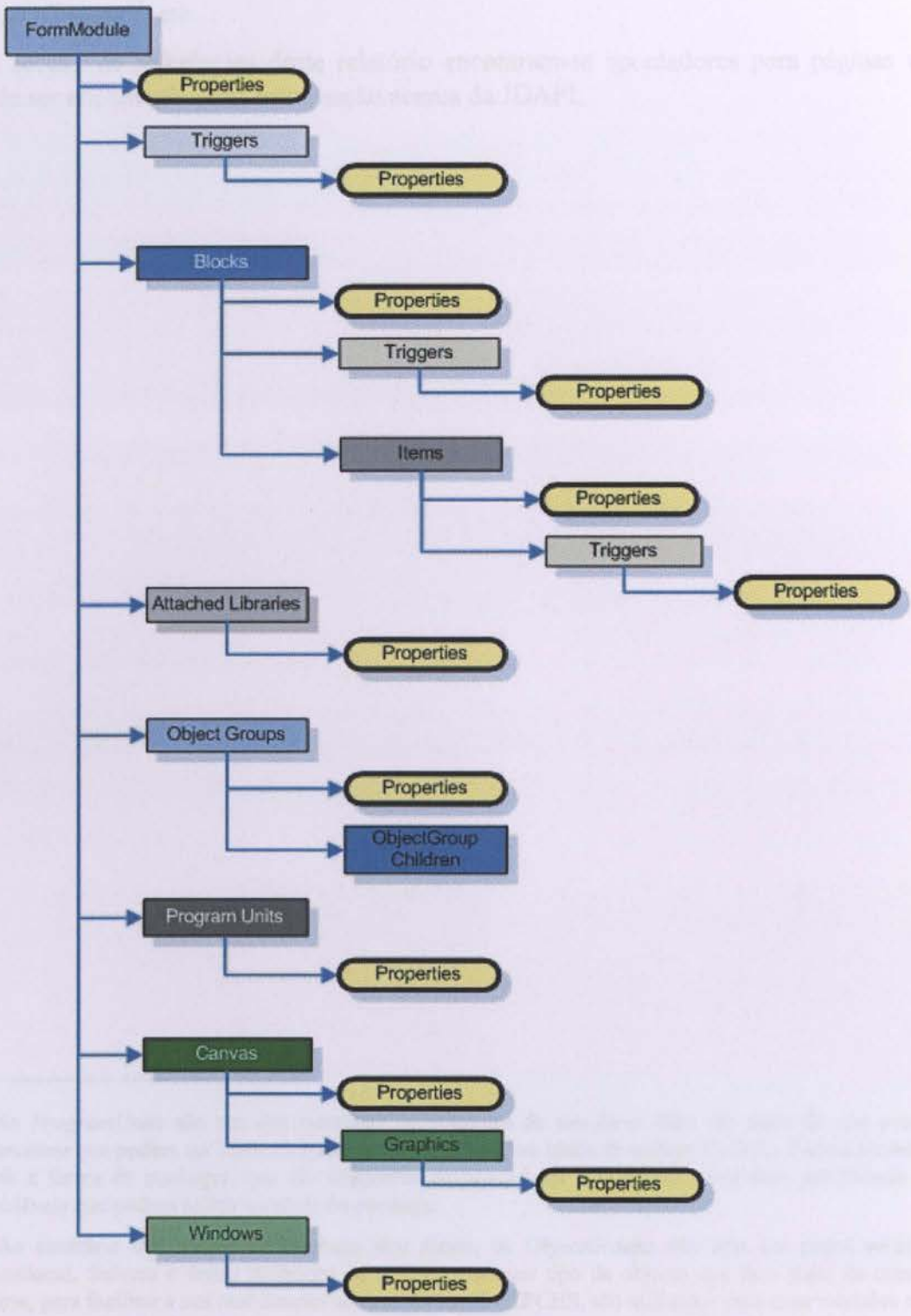


Fig.3 *– Hierarquia lógica de classes da JDAPI, ilustrando as relações existentes entre alguns dos diferentes constituintes de um form. Note-se que este esquema não inclui todos os conceitos suportados pela JDAPI, mas apenas aqueles mais ilustrativos da sua estrutura hierárquica.*

Naturalmente, existem outros conceitos associados aos *forms*, e a JDAPI não deixa qualquer um deles de parte. É possível, por exemplo, remover e adicionar bibliotecas de funções associadas aos *forms*; consultar e modificar o código PL/SQL de *ProgramUnits*[11] (assim como o código PL/SQL dos *triggers* de qualquer objecto); gerir o conteúdo de *ObjectGroups*[12]; etc.

Na secção de referências deste relatório encontram-se apontadores para páginas *web* onde pode ser encontrada mais informação acerca da JDAPI.

---

[11] As *ProgramUnits* são um dos possíveis constituintes de um *form*. Não são mais do que *procedures* ou *functions* que podem ser invocados livremente em qualquer bloco de código PL/SQL. Podem também aparecer sob a forma de *packages*, que são conjuntos de *procedures* e *functions* agrupados, partilhando o acesso a variáveis que podem existir ao nível do *package*.

[12] Ao contrário dos outros constituintes dos *forms*, os *ObjectGroups* não têm um papel verdadeiramente funcional. Servem o único propósito de agrupar qualquer tipo de objecto que faça parte da constituição do *form*, para facilitar a sua reutilização noutros *forms*. Na CPCHS, são utilizados para criar módulos reutilizáveis da seguinte forma: é criado um *form* com todos os objectos que se pretende reutilizar, incluindo o código com todas as funcionalidades; ainda no mesmo *form*, cria-se um *ObjectGroup*, ao qual são adicionados todos esses objectos; sempre que se pretenda reutilizar esse componentes em determinado *form*, basta abrir esse *form no Forms Builder* a par do *form* que constitui o módulo reutilizável, e arrastar o ObjectGroup para o novo *form*. É possível indicar que os objectos do *ObjectGroup* devem ser simplesmente copiados para o novo *form*, ou que deve ser mantida uma referência aos objectos originais, de modo a que sempre que estes sejam modificados, as alterações se propaguem ao *form* que reutiliza o *ObjectGroup*.

4 Especificação das Soluções

4.1 A *Migration Tool*

Tal como já se tornou evidente no capítulo anterior, a aplicação de conversão de *forms* da plataforma 6i para 9i, aqui denominada por *Migration Tool*, foi desenvolvida tendo como base tecnológica a plataforma Java, recorrendo à JDAPI para interagir com os *forms*. A escolha recaiu sobre esta tecnologia por dois motivos: pelas enormes possibilidades que ela oferece para a resolução do problema em questão; e pelo facto de não existir uma alternativa viável.

A secção 4.1.1 apresenta detalhadamente os vários requisitos da *Migration Tool*. Na secção 4.1.2 são descritos os casos de utilização, sendo apresentados os actores do sistema. Na secção 4.1.3 é feita uma descrição do vocabulário do domínio da aplicação. Finalmente, na secção 4.1.4, é apresentada a arquitectura de alto nível do sistema.

Se o leitor não estiver familiarizado com a programação em *Oracle Forms Builder*, recomenda-se a leitura do anexo A antes de continuar. A leitura do Anexo B é também aconselhada, pois vai ajudar a compreender os requisitos da *Migration Tool*.

4.1.1 Especificação de Requisitos

São de seguida listados e explicados os requisitos funcionais obrigatórios e opcionais, e os requisitos não funcionais da aplicação.

Os requisitos da aplicação são apresentados nesta secção inicial, e não associados aos casos de utilização. Isto acontece porque a maioria dos requisitos é partilhada por todos os casos de utilização, e aqueles que não o são podem ser facilmente associados ao caso respectivo.

Requisitos Funcionais

- Efectuar ligação à base de dados – a generalidade dos *forms* interage de alguma forma com uma base de dados Oracle, fazendo consultas, alterações de registos ou criações de novos registos, e também invocando procedimentos e funções definidos na própria base de dados. Para que estes *forms* possam ser compilados com sucesso, é necessário ter uma ligação activa à base de dados.
- Ler ficheiros *fmb*, e proceder às necessárias alterações:
    - Remover e adicionar associações a bibliotecas.
    - Remover e "carregar" de novo a *template* (e, consequentemente, redefinir propriedades que possam ter sido perdidas neste processo).
    - Reposicionar *DataBlocks* e *Canvases* aquando da criação do ficheiro executável para a plataforma 9i - a *template* utilizada na plataforma 9i inclui dois *DataBlocks* e dois *Canvases* adicionais, que são colocados automaticamente no início da lista de *DataBlocks* e *Canvases*, respectivamente. Estes objectos devem ser movidos para o final dessas listas, caso contrário esse posicionamento poderá influenciar o comportamento dos *forms* na plataforma *web*.

  - Procurar no código PL/SQL chamadas a determinadas funções e procedimentos e registar a ocorrência das mesmas – eram já conhecidos, à partida, determinados procedimentos e funções que poderiam trazer problemas aquando da execução na plataforma 9i, e para os quais não foi ainda implementada uma alternativa viável. A identificação dos *forms* que as utilizam é importante para mais tarde se saber exactamente onde intervir caso as aplicações evidenciem comportamentos incorrectos na plataforma web (ver Anexo B).
  - Substituir determinadas porções de código PL/SQL nos *forms*, nomeadamente invocações a procedimentos e funções – este passo consiste essencialmente em substituir chamadas a funções e procedimentos que não funcionam correctamente na plataforma *web* por outras versões dos mesmos compatíveis com ambas as plataformas (6i e 9i); nos casos mais simples, apenas o nome da função e/ou do *package* onde esta reside é alterado, mas há casos onde é necessário efectuar o *parsing* dos argumentos passados, para os adaptar à assinatura dos novos métodos (ver Anexo B).
  - As alterações referidas no ponto anterior devem ser efectuadas em todos os constituintes do *form* onde possa existir código PL/SQL:
    - *Program Units* – (funções e procedimentos simples, ou *packages* que contêm funções e/ou procedimentos);
    - *Triggers* (ao nível do *form*, dos blocos e dos *items*);
  - As substituições no código só devem ser efectuadas quando o objecto ao qual este pertence não o tiver herdado (caso contrário, perde-se a relação de herança para a propriedade do código PL/SQL do objecto).
- Guardar as alterações efectuadas aos *forms* única e exclusivamente no formato de ficheiro do *Forms Builder* 6i (para que exista, sempre que possível, uma única versão de cada *form*, e não uma para cada plataforma).
- Compilar o *form* já alterado, criando executáveis para as plataformas 6i e 9i.
- Processar vários *forms* de uma vez (*batch mode*).
- Gerar relatórios (*logs*) de todo o processo de conversão, apresentando estatísticas acerca do número de *forms* convertidos com sucesso e com erro, alterações efectuadas, etc.
- Identificar claramente os *forms* cuja conversão não foi efectuada com sucesso, separando-os dos restantes.
- Criar um *backup* do ficheiro *fmb* original.
- Permitir a configuração do processo de conversão:
  - Definição do nome das bibliotecas que devem ser sempre removidas; das que devem ser sempre adicionadas; e das que devem ser substituídas por outras, caso sejam utilizadas.
  - Indicação da localização do ficheiro *temp.fmb* (a *template* base de todos os *forms*).

  - Possibilidade de definir qual a base de dados a que se efectua a ligação, assim como o nome de utilizador e respectiva *password* (já que existem várias bases de dados, e cada utilizador tem acesso a objectos diferentes).

- Garantir que um *form* que tenha sido convertido previamente possa ser convertido novamente sem qualquer problema. O seguinte exemplo ilustra este requisito. Uma das tarefas da aplicação é substituir todas as chamadas a funções no *package TEXT_IO* por invocações às funções homónimas no *package CPC_TEXT_IO*. A aplicação deve procurar o texto "TEXT_IO" no código PL/SQL e substituir todas as ocorrências encontradas por "CPC_TEXT_IO". Contudo, deve ter o cuidado de não substituir "CPC_TEXT_IO" por "CPC_CPC_TEXT_IO", caso contrário um ficheiro com esta invocação já corrigida não poderia ser submetido à conversão através da *Migration Tool*.

Requisitos Funcionais Opcionais

- Gerar estatísticas acerca da ocorrência e número de invocações a determinadas funções e procedimentos. Existe já um requisito obrigatório semelhante a este, residindo a diferença no facto de este (o opcional) se referir a quaisquer funções e procedimentos para as quais o utilizador pretenda obter estatísticas, enquanto que o requisito obrigatório atrás referido se refere a procedimentos e funções relevantes para o processo de conversão, contemplando métodos conhecidos como possíveis pontos de falha aquando da execução na plataforma *web*.
- Disponibilizar uma interface gráfica para o utilizador iniciar o processo de conversão, assim como configurar as várias opções dadas pela aplicação.
- Possibilidade de guardar num ficheiro as configurações efectuadas pelo utilizador, de modo a reutilizá-las na invocação seguinte da aplicação (localização do ficheiro *temp.fmb*, assim como os parâmetros de ligação à base de dados, etc.).
- Para além do ficheiro de *log* referido num dos requisitos obrigatórios, criar um outro ficheiro, num formato compatível com o ficheiro utilizado aquando da realização do processo de conversão manual no início do projecto de estágio (um ficheiro Excel).
- Apresentar na interface gráfica uma estimativa do tempo restante para terminar o processo actual de conversão (útil quando o processo de conversão inclui muitos *forms*).
- São necessários vários passos de configuração numa máquina para que esta possa executar a *Migration Tool*:
  - a variável de ambiente *CLASSPATH* deve apontar para o ficheiro *f60jdapi.jar* aquando da execução das alterações ao ficheiro *fmb* na plataforma 6i.
  - para a JDAPI da versão 6i funcionar, é necessário ter na variável de sistema *PATH* a localização do directório *bin* da instalação do *Forms Builder 6i* com o *patch* para suportar esta API.
  - a compilação do *form* em 6i, que não pode ser feita através da API por questões de compatibilidade, necessita que a variável *PATH* aponte para o directório *bin* de uma instalação do *Forms Builder 6i* sem o patch instalado.

- para se poder aceder à JDAPI para 9i, é necessário que a variável *PATH* aponte para o directório *bin* da instalação do *Forms Builder 9i*, para além da *CLASSPATH* apontar para o ficheiro f90jdapi.jar.

Por ser tão complexa a configuração necessária para a aplicação funcionar correctamente, pretende-se que esta seja capaz de executar o maior número possível destes passos automaticamente, a fim de facilitar a sua instalação.

Requisitos Não Funcionais

- Desempenho – pretende-se que o processo de conversão através da *Migration Tool* seja suficientemente rápido, de modo a que demore consideravelmente menos que o processo de conversão manual. Este (o processo de conversão manual), dependendo da complexidade do *form* a converter, pode demorar desde 15 a mais de 30 minutos.
- Consumo de recursos – a utilização de memória deve ser controlada, de forma a evitar que durante um processo de conversão de várias dezenas de *forms* a utilização de memória cresça demasiado, tornando o sistema demasiado lento.

4.1.2 Casos de Utilização

Foi identificado um único actor para a aplicação a desenvolver, tendo-lhe sido associados três casos de utilização distintos. Na figura 4 é apresentado o modelo de casos de utilização da *Migration Tool*.

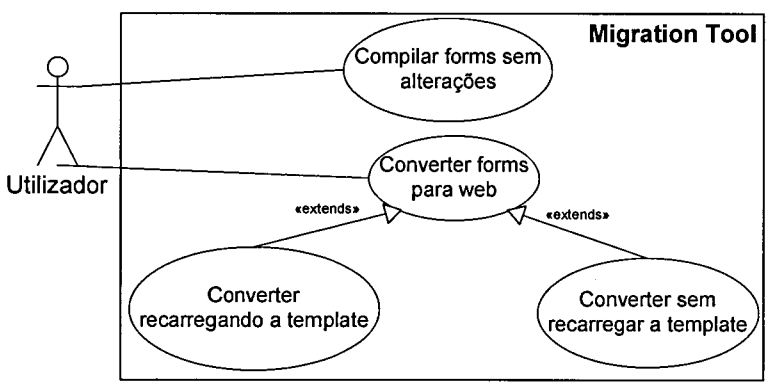


***Fig.4** – Modelo de Casos de Utilização da Migration Tool.*

O único actor do sistema é o utilizador da aplicação, o programador que pretende efectuar a conversão de *forms*.

Os vários casos de utilização são descritos detalhadamente de seguida, sendo apresentada a sequência de operações que constitui cada um deles, através de diagramas de actividade. São evidenciadas as acções levadas a cabo pelo utilizador para cada um dos casos, assim como as operações internas realizadas pelo sistema. São também incluídos protótipos das interfaces gráficas da aplicação, para melhor ilustrar a forma de interacção do utilizador com o sistema.

4.1.2.1 Compilar forms sem alterações

Este caso de utilização consiste na compilação de um ou mais *forms* (ficheiros *fmb* criados com a versão 6i do *Forms Builder*) para as plataformas cliente-servidor (6i) e *web (9i)*. O ficheiro com a *source* do *form* não sofre quaisquer alterações, sendo apenas gerados os ficheiros executáveis e o relatório das operações.

Sequência de Funcionamento

O diagrama de actividade da figura 5 ilustra os passos que o utilizador deve seguir na execução deste caso de utilização, ao mesmo tempo que apresenta a sequência de operações realizadas internamente pelo sistema para satisfazer o pedido.

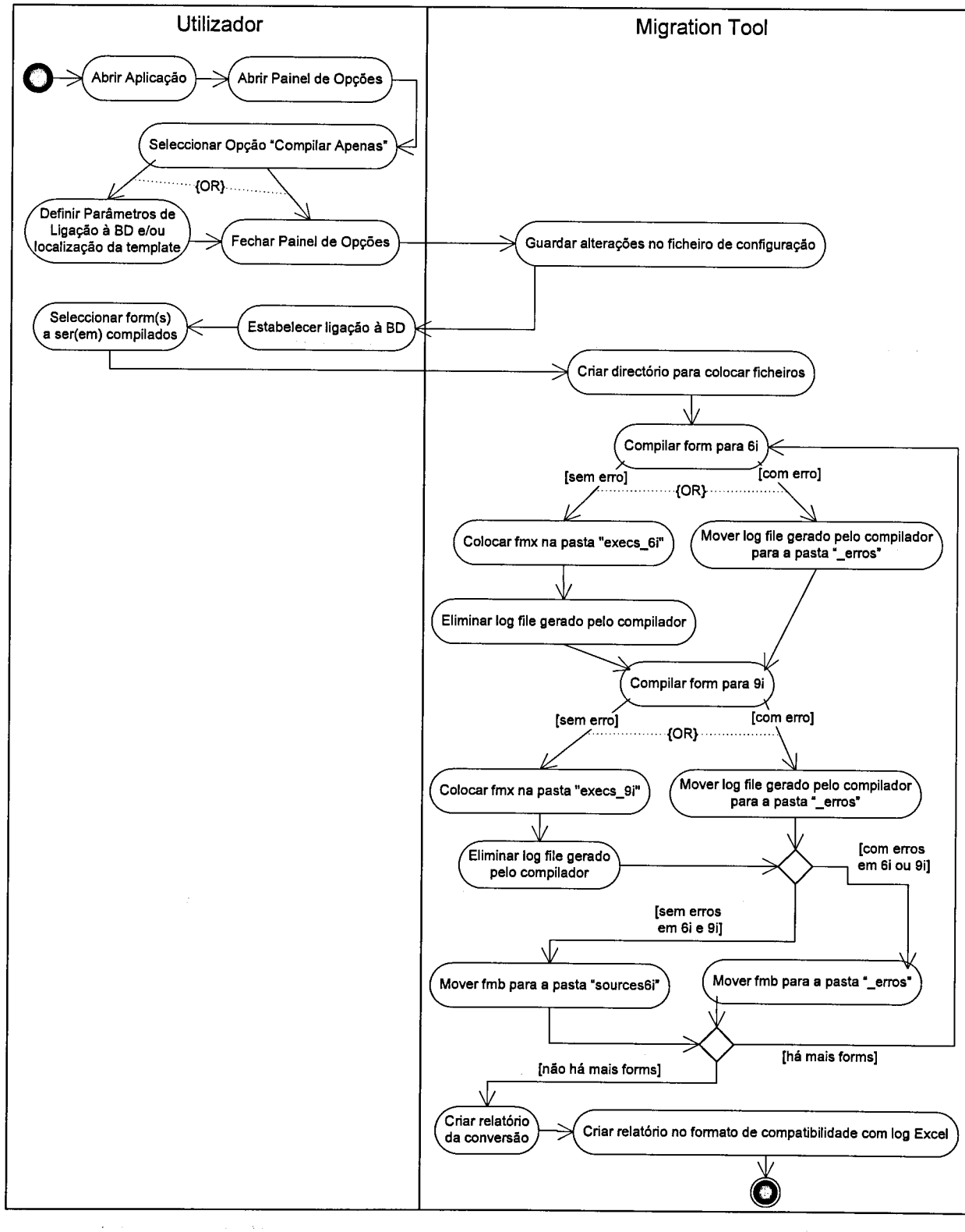


Fig.5 *– Diagrama de actividade do caso de utilização "Compilar forms sem alterações".*

Este caso de uso inicia-se quando o utilizador pretende compilar um número arbitrário de *forms*. Nesta situação, nenhuma alteração é efectuada ao ficheiro fonte, para além do estritamente necessário para a conversão para a plataforma 9i[13]. Este caso de uso será desencadeado tipicamente quando o utilizador souber que os *forms* que precisa de compilar são já compatíveis com a plataforma 9i, porque foram criados tendo esta plataforma já em conta, ou então foram já submetidos à conversão através da *Migration Tool*. Nesta situação, indicar à aplicação que não necessita de pesquisar todo o código de todos os objectos do *form* traz grandes ganhos no tempo necessário para completar a compilação, principalmente se se estiver a processar um número elevado de *forms*.

Para indicar à aplicação que pretende efectuar apenas a compilação dos *forms* sem proceder a quaisquer verificações ou alterações, o utilizador deve, depois de abrir a aplicação, aceder ao painel de opções, e seleccionar a opção "Compilação apenas", em detrimento da opção "Migração dos *forms*". Ainda neste painel de opções, é dada ao utilizador a possibilidade de definir quais são os parâmetros de ligação à Base de Dados, assim como indicar qual é a localização do ficheiro da *template*. Este passo de configuração é opcional, pois a aplicação guarda os valores definidos numa utilização anterior.

Apesar de, por simplificação, isto não ser evidente no diagrama, o passo da ligação à Base de Dados é opcional, e não tem necessariamente que ser executado naquela posição do fluxo de acções. É possível que o utilizador feche o painel de opções e imediatamente a seguir seleccione os ficheiros que pretende compilar, sem estabelecer a ligação. Se não o fizer, a *Migration Tool* pergunta-lhe se pretende efectuar essa ligação. E ainda aqui o utilizador pode dizer que não[14].

O passo seguinte que o utilizador deve dar é o de seleccionar o(s) *form(s)* que pretende compilar. Faz isto através de uma janela de selecção de ficheiros com possibilidade de selecção múltipla. Depois deste passo, o utilizador deve esperar que o processo de compilação esteja completado, já que todas as actividades seguintes são da responsabilidade da *Migration Tool*.

Esta começa por, no mesmo directório onde se encontram os ficheiros *fmb* que vai processar, criar um novo sub directório, no qual colocará todos os ficheiros resultantes do processo de compilação. O nome dado a este directório é "migracao_", seguido da data e hora em que o processo de compilação teve início. Isto permite que um novo processo de compilação iniciado para um mesmo ficheiro no mesmo directório não elimine os ficheiros criados aquando da compilação/conversão anterior. A aplicação copia para esse directório o ficheiro *fmb* que vai processar, para evitar danificar o original, caso algum problema ocorra.

O passo seguinte consiste em abrir o ficheiro *fmb* e, já que nenhuma alteração ou verificação deve ser efectuada, proceder à compilação do mesmo. O compilador (externo à *Migration Tool*) gera um ficheiro de *log* com o resultado da compilação. Se a compilação for concluída

---

[13] Note-se que estas alterações necessárias para a compilação em 9i são efectuadas apenas para que se possa proceder à compilação, sendo descartadas de seguida. Isto acontece porque se as alterações fossem guardadas, o ficheiro *fmb* desse *form* deixaria de ser compatível com o *Forms Builder 6i*.

[14] A possibilidade de não estabelecer ligação à BD pode ser útil quando os *forms* que se pretende compilar ou converter não necessitarem dela. Naturalmente, um *form* que não precisa da Base de Dados pode também ser compilado/convertido com a ligação activa. Mas se se considerar a hipótese remota de não ser possível estabelecer a ligação à BD, esta opção vai então permitir que a aplicação proceda à conversão de um destes *forms* sem problemas.

com sucesso, o conteúdo deste ficheiro é irrelevante, já que vai conter apenas uma mensagem indicando o sucesso da operação. Neste caso, este ficheiro de *log* do compilador é eliminado, e o ficheiro executável criado, de extensão *fmx*, é colocado num sub directório chamado "execs_6i". Se ocorrerem erros de compilação, este ficheiro de *log* é guardado para mais tarde ser analisado pelo utilizador. Este ficheiro é movido para a pasta "_erros", criada assim que o primeiro erro de compilação ocorre. Havendo erros de compilação, é pouco provável que o ficheiro executável seja criado, mas é possível que seja. Neste caso, esse é copiado para a pasta "execs_6i".

O ficheiro *fmb* é de seguida aberto e compilado para a plataforma 9i, através da *JDAPI*. Quando o *form* é aberto com esta versão da API, utilizando a *template* específica para a plataforma 9i, são efectuados pequenos ajustes no posicionamento de alguns constituintes do *form*, e este é compilado. Os passos levados a cabo depois da compilação em 6i são agora repetidos para a compilação em 9i, sendo que os ficheiros de *log* da compilação e o executável eventualmente criado recebem o mesmo tratamento. Desta feita, o directório utilizado para colocar os ficheiros executáveis chama-se "execs_9i".

Depois da compilação do *form* para ambas as plataformas estar concluída, a aplicação verifica se ocorreu algum erro. Não tendo ocorrido nenhum, o ficheiro *fmb* colocado na raiz do directório do projecto no início do processo é movido para a pasta "sources6i", criada na primeira vez que é utilizada. É utilizado o nome "sources6i" pois os ficheiros aqui colocados, mesmo quando são alterados, são sempre ficheiros fonte do *Forms Builder 6i*. No caso de ter ocorrido algum erro na compilação para 6i ou 9i, o ficheiro *fmb* é movido para a pasta "_erros".

Os passos até aqui descritos correspondem a uma iteração de todo o processo, sendo repetidos para cada um dos *forms*. No final, é criado um relatório com a indicação do resultado do processo para cada um deles, contendo ainda estatísticas globais. Antes do processo de conversão dos *forms* ser efectuado pela *Migration Tool*, estes eram convertidos manualmente, sendo mantido um ficheiro com uma folha de cálculo do Excel, onde eram registadas as alterações efectuadas aos *forms*, assim como as verificações que se efectuavam. Por questões de compatibilidade, a *Migration Tool* gera ainda um outro relatório, num formato que permite a fácil importação desses dados para a folha de Excel utilizada. Este relatório, contudo, não contém tanta informação como o anterior. Os ficheiros com os relatórios, denominados "log.txt" e "logExcel.txt", são colocados na raiz do directório do projecto.

Interface com o Utilizador

Nas figuras 6, 7 e 8 são apresentados protótipos das interfaces com o utilizador disponibilizadas pela *Migration Tool*. Nesta secção estas interfaces são descritas com algum detalhe, considerado necessário para melhor se compreender o funcionamento da aplicação.

Na figura 6 é visível o painel de opções da aplicação, com a opção de "Compilação apenas" seleccionada. Esta é a opção correspondente ao caso de utilização "Compilar *forms* sem alterações". Neste painel é possível definir os parâmetros de ligação à Base de Dados, caso tal seja necessário. Este passo é opcional, porque estes parâmetros podem já estar correctamente definidos, não sendo necessário alterá-los. O mesmo se passa com a localização do ficheiro da *template*, *temp.fmb*. Note-se que é ainda possível neste painel seleccionar a opção de não recarregar a *template*, sendo que nesse caso esse ficheiro não é utilizado.

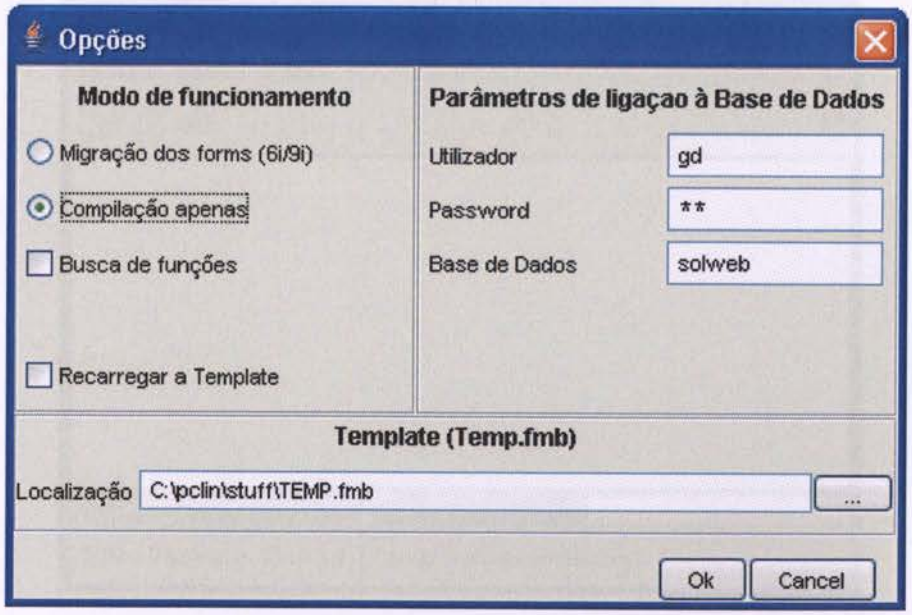


***Fig.6** – Painel de opções da Migration Tool. Aqui, o utilizador pode definir os parâmetros de ligação à Base de Dados, a localização do ficheiro temp.fmb, e ainda indicar se deve ser feita a migração completa ou se os forms devem ser compilados sem alterações.*

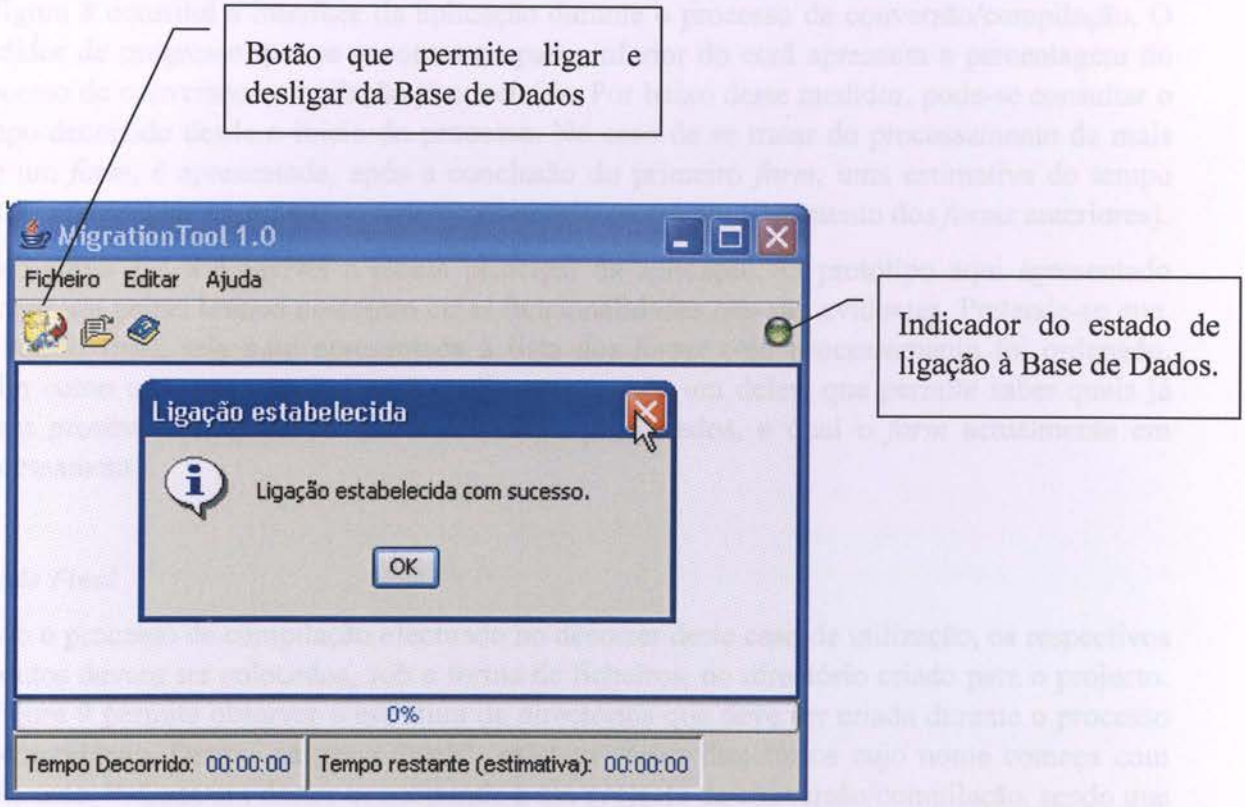


***Fig.7** – Mensagem apresentada ao utilizador depois da ligação à Base de Dados ter sido efectuada com sucesso. O indicador de estado fica vermelho quando não está estabelecida uma ligação, e verde caso contrário.*

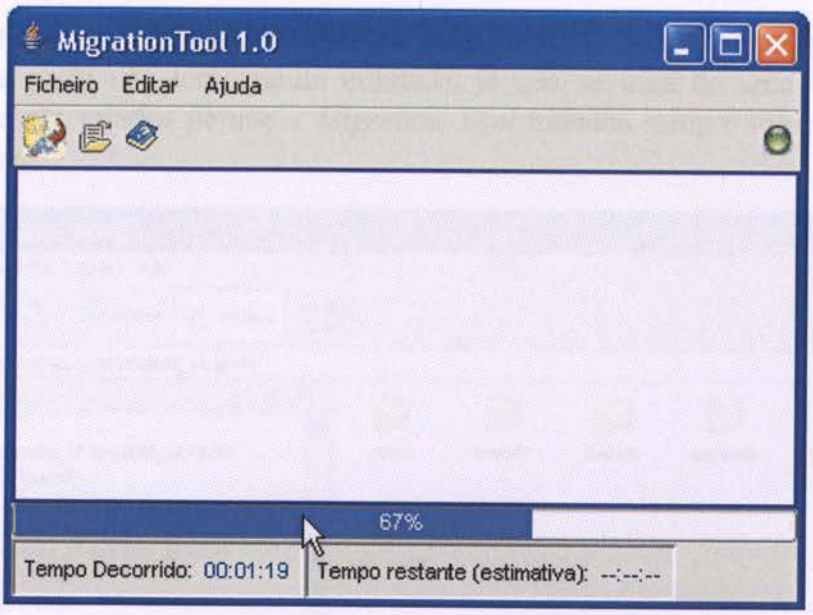


***Fig.8** – Interface apresentada ao utilizador durante o processo de conversão/compilação.*

A figura 7 mostra a mensagem apresentada ao utilizador depois da ligação à Base de Dados ter sido estabelecida com sucesso. Se, por outro lado, o estabelecimento da ligação falhar por algum motivo, o utilizador deve ser alertado do facto com uma mensagem de erro.

A figura 8 constitui a interface da aplicação durante o processo de conversão/compilação. O medidor de progresso que se encontra na parte inferior do ecrã apresenta a percentagem do processo de conversão/compilação já concluída. Por baixo deste medidor, pode-se consultar o tempo decorrido desde o início do processo. No caso de se tratar do processamento de mais que um *form*, é apresentada, após a conclusão do primeiro *form*, uma estimativa do tempo restante (calculada com base no tempo necessário para o processamento dos *forms* anteriores).

Nas figuras 7 e 8 é visível a janela principal da aplicação. O protótipo aqui apresentado contém um painel branco no centro cujas funcionalidades não são evidentes. Pretende-se que, na versão final, seja aqui apresentada a lista dos *forms* cujo processamento foi ordenado, assim como uma pequena indicação, adjacente a cada um deles, que permite saber quais já foram processados, quais os que ainda serão processados, e qual o *form* actualmente em processamento.

Estado Final

Findo o processo de compilação efectuado no decorrer deste caso de utilização, os respectivos produtos devem ser colocados, sob a forma de ficheiros, no directório criado para o projecto. A figura 9 permite observar a estrutura de directórios que deve ser criada durante o processo de compilação. Dentro da pasta "web", existem vários directórios cujo nome começa com "migracao_". Cada um destes corresponde a um projecto de conversão/compilação, sendo que cada projecto consiste no processamento de um ou mais *forms*. O resto do nome dos directórios consiste na data e hora em que o processo de conversão/compilação foi iniciado.

Num processo onde não ocorram erros de compilação, devem ser criados três directórios: "Execs6i", "Execsi9" e "Sources6i". Nos dois primeiros são colocados os ficheiros executáveis gerados para cada uma das plataformas. Em "Sources6i" é colocada uma cópia de

cada um dos ficheiros fonte (*fmb*). Neste caso de uso, "Compilar *forms* sem alterações", o conteúdo desta pasta não tem grande utilidade, já que se trata de uma cópia dos *forms* originais. Estes são criados porque a *Migration Tool* trabalha sempre sobre cópias e nunca sobre os originais.

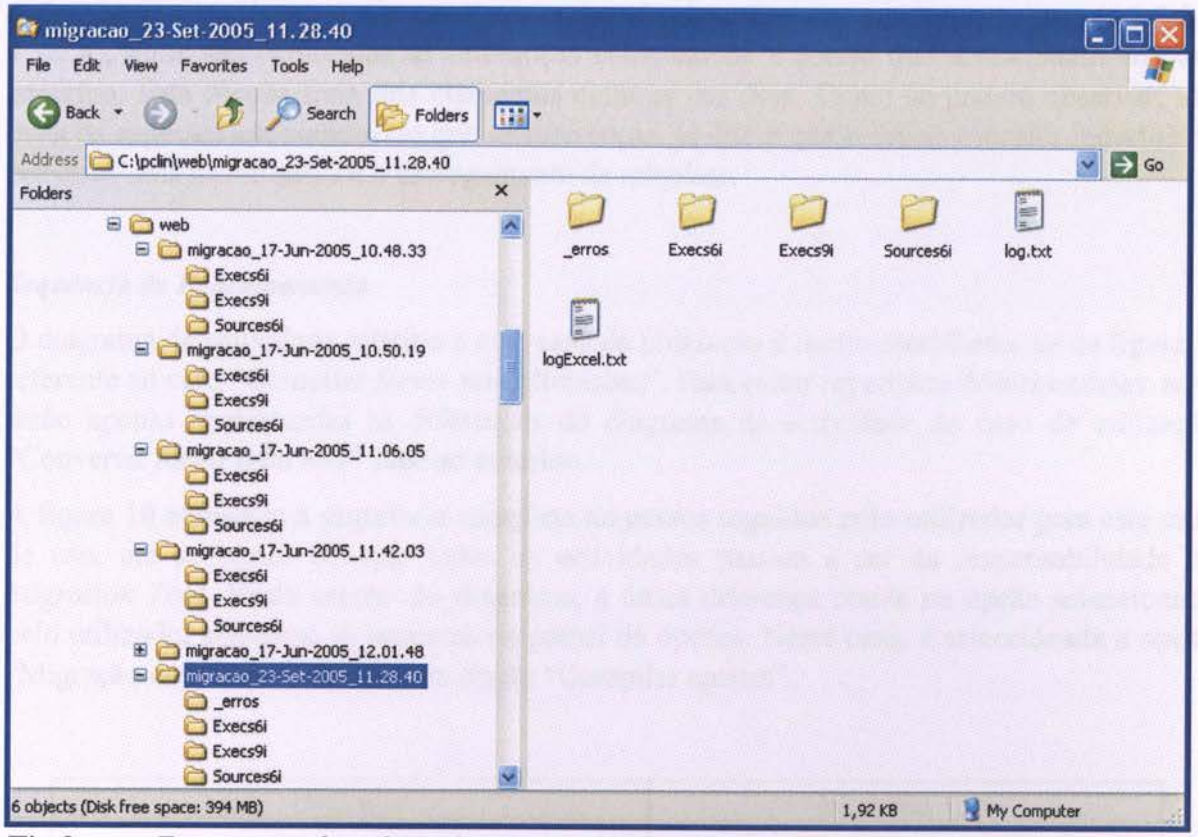


***Fig.9** – Estrutura de directórios e ficheiros criados durante o processo de compilação/conversão.*

Se ocorrer algum erro de compilação, deve ser criado um outro directório, com o nome "_erros", onde são colocados os ficheiros *fmb* correspondentes aos *forms* com erro, assim como os ficheiros gerados pelos compiladores com a descrição dos erros ocorridos. Neste caso, as cópias dos ficheiros *fmb* ficam distribuídas pelos directórios "Sources6i" e "_erros".

Na raiz do directório do projecto, e ao mesmo nível dos directórios "Execs6i", "Execs9i", "Sources9i" e "_erros", são criados os ficheiros de *log* do processo. Neste caso de utilização, onde nenhuma alteração ou verificação é efectuada, estes ficheiros contêm apenas a indicação de que a compilação ocorreu com ou sem erros.

4.1.2.2 Converter forms para web

Neste caso de utilização, os *forms* são processados pela *Migration Tool* antes de serem compilados. Este caso inicia-se quando o utilizador pretende converter um número arbitrário de *forms* para a plataforma 9i. Esta conversão implica alterações às fontes dos *forms* (ficheiros *fmb*) para que estas fiquem compatíveis com ambas as plataformas, e a compilação dos *forms* para as mesmas.

Os passos efectuados durante o processamento a que os *forms* são sujeitos são descritos na secção 4.1.1. Tudo o que foi referido no caso de utilização anterior (secção 4.1.2.1) aplica-se

também a este, residindo as diferenças nos passos de processamento adicionais efectuados no caso "Converter *forms* para *web*", e no facto de os ficheiros de *log* criados no âmbito deste caso de uso incluírem descrições das várias alterações e verificações efectuadas.

Note-se que "Converter *forms* para *web*" trata-se de um caso de utilização genérico, como se pode observar na figura 4. Os casos concretos são descritos nas secções seguintes (4.1.2.3 e 4.1.2.4), onde são explicadas as diferenças entre ambos e aquilo que acrescentam ao caso genérico. Esta secção trata dos elementos comuns aos dois. Como se poderá observar, são mais os aspectos em comum do que as diferenças, já que o que o único conceito introduzido por estes dois novos casos é o carregamento da *template*.

Sequência de Funcionamento

O diagrama de actividade relativo a este caso de utilização é muito semelhante ao da figura 5, referente ao caso "Compilar *forms* sem alterações". Para evitar repetições desnecessárias, aqui serão apenas apresentadas as diferenças do diagrama de actividade do caso de utilização "Converter *forms* para *web*" face ao anterior.

A figura 10 apresenta a sequência completa de passos seguidos pelo utilizador para este caso de uso, até ao ponto em que todas as actividades passam a ser da responsabilidade da *Migration Tool*. Nesta secção do diagrama, a única diferença reside na opção seleccionada pelo utilizador enquanto se encontra no painel de opções. Neste caso, é seleccionada a opção "Migração dos *forms*", ao invés da opção "Compilar apenas".

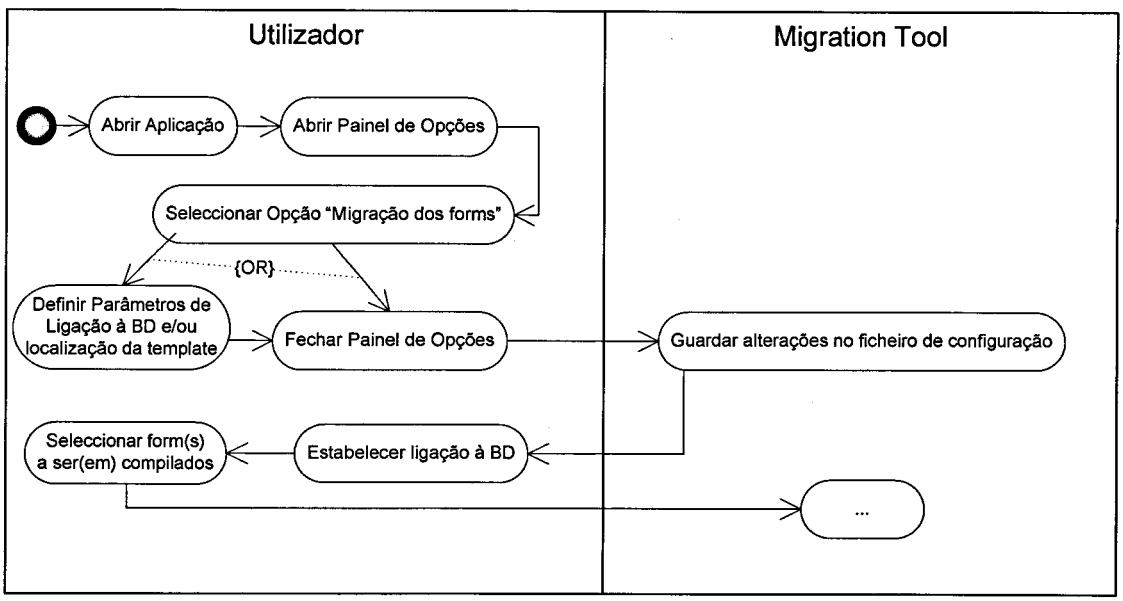


***Fig.10** – Diagrama de actividade parcial do caso de utilização "Converter forms para web". Os passos levados a cabo exclusivamente pela Migration Tool, já sem a intervenção do utilizador, não estão aqui contemplados.*

Os passos que surgem no seguimento do diagrama da figura 10, apresentados no diagrama parcial da figura 11, são os mesmos que aparecem na figura 5. A única diferença a notar é a existência de novas actividades entre "Criar directório para colocar ficheiros..." e "Compilar

form para 6i" (as actividades realçadas a amarelo). Por isto, a figura 11 contempla estas novas actividades, mas não repete as restantes, que surgem a seguir a "Compilar *form* para 6i" no fluxo de actividades.

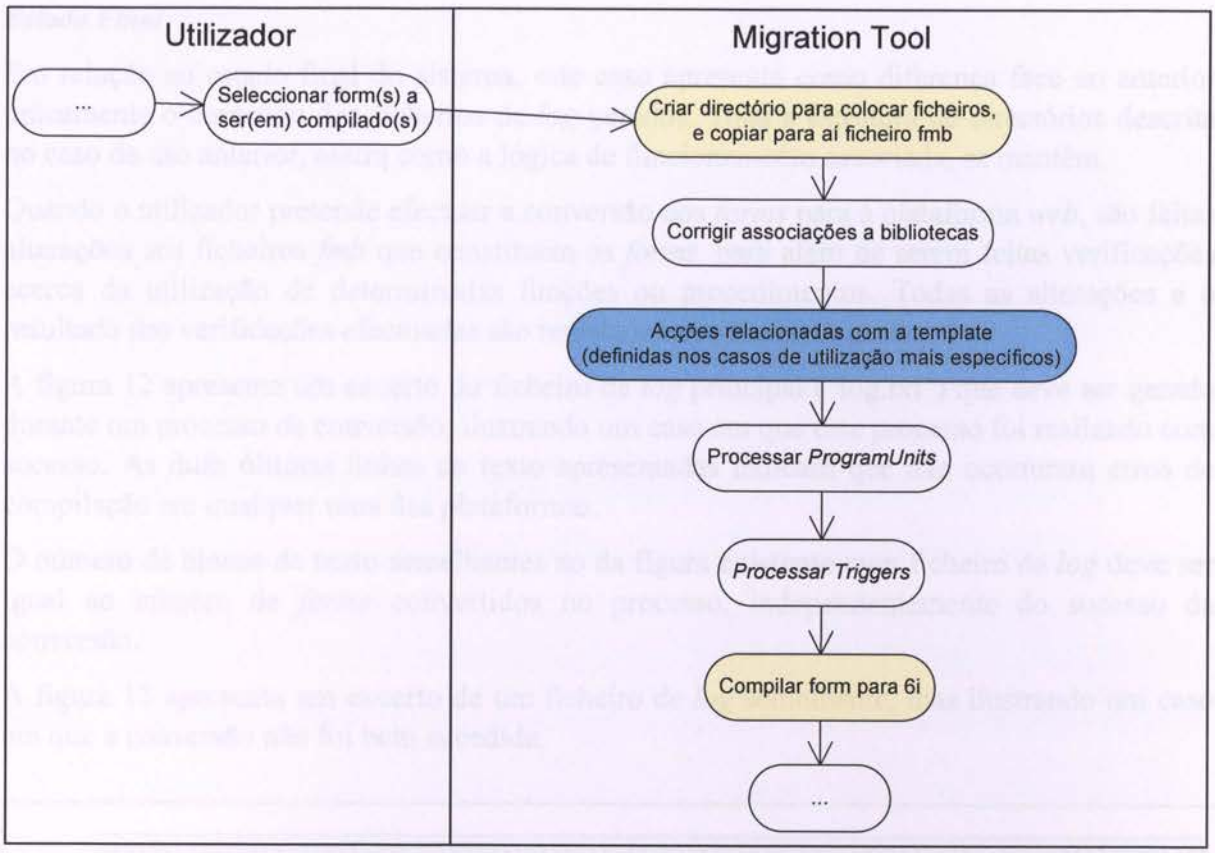


***Fig.11** – Diagrama de actividade parcial do caso de utilização "Converter forms para web". Os passos apresentados neste diagrama surgem no seguimento dos da figura anterior.*

Ainda no diagrama da figura 11, aparece uma actividade realçada a azul. Esta surge aqui como uma actividade abstracta, que será definida apenas nos casos de utilização que são especializações deste, descritos nas secções 4.1.2.3 e 4.1.2.4.

A actividade "Corrigir associações a bibliotecas" corresponde à eliminação de bibliotecas de funções e procedimentos obsoletas, e sua substituição, sempre que necessário, por novas versões das mesmas. "Processar *ProgramUnits*" corresponde às alterações que devem ser efectuadas no código PL/SQL das *ProgramUnits* dos *forms*, e ainda às verificações que devem ser feitas. A actividade "Processar *Triggers*" consiste exactamente no mesmo processamento, sendo que, desta feita, os alvos do processamento são todos os *triggers* do *form* (o que inclui os *triggers* directamente no nível do *form*, os *triggers* dos vários *DataBlocks*, e os *triggers* dos vários *items* dos vários *DataBlocks*).

Interface com o Utilizador

As interfaces com o utilizador associadas a este caso de utilização são em quase tudo semelhantes àquelas apresentadas para o caso de uso anterior. Relativamente a este assunto,

basta apenas referir que o utilizador, depois de abrir o painel de opções, deve seleccionar a opção "Migração dos *forms*" ao invés de "Compilação apenas".

Estado Final

Em relação ao estado final do sistema, este caso apresenta como diferença face ao anterior unicamente o conteúdo dos ficheiros de *log* gerados. Toda a estrutura de directórios descrita no caso de uso anterior, assim como a lógica de funcionamento associada, se mantêm.

Quando o utilizador pretende efectuar a conversão dos *forms* para a plataforma *web*, são feitas alterações aos ficheiros *fmb* que constituem os *forms*, para além de serem feitas verificações acerca da utilização de determinadas funções ou procedimentos. Todas as alterações e o resultado das verificações efectuadas são registados nos ficheiros de *log*.

A figura 12 apresenta um excerto do ficheiro de *log* principal ("log.txt") que deve ser gerado durante um processo de conversão, ilustrando um caso em que este processo foi realizado com sucesso. As duas últimas linhas de texto apresentadas indicam que não ocorreram erros de compilação em qualquer uma das plataformas.

O número de blocos de texto semelhantes ao da figura existente num ficheiro de *log* deve ser igual ao número de *forms* convertidos no processo, independentemente do sucesso da conversão.

A figura 13 apresenta um excerto de um ficheiro de *log* semelhante, mas ilustrando um caso em que a conversão não foi bem sucedida.

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Form ghpa7370.fmb
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

--> Template foi novamente carregada;
--------------------------------------------------
--> Bibliotecas Removidas:
        --> HUC;
--> Bibliotecas Adicionadas:
        --> REPORTS_2004;
        --> WINHELP;
--------------------------------------------------
--> Nº de  chamadas a Host   detectadas e corrigidas = 1 --> (1 reports + 0 outros)
--> Nº de  get_file_name     detectados e corrigidos = 0
--> Nº de  tool_env          detectados e corrigidos = 0
--> Nº de  text_io           detectados e corrigidos = 8
--> Nº de  run_product       detectados = 0
--> Nº de  mouse triggers    detectados = 0 --> (0 when-mouse-enter + 0 when-mouse-leave)
--> Nº de  create_timer      detectados = 0
--------------------------------------------------
--> Nº de  win_api_environment     detectados e corrigidos = 0
--> Nº de  Outros win_api           detectados (Total) = 0
--------------------------------------------------
Compilação em 6i - SEM erros;
Compilação em 9i - SEM erros;
--------------------------------------------------
```

***Fig.12** – Excerto de um ficheiro de log referente a um form cuja conversão foi efectuada com sucesso, conforme se pode verificar pela informação contida nas duas últimas linhas de texto, e pela ausência da palavra "ERRO" repetida a seguir ao nome do form.*

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Form ghpa7700.fmb        --->      ERRO   ERRO   ERRO   ERRO   ERRO   ERRO   ERRO
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
--> Template foi novamente carregada;
----------------------------------------------------------
--> Bibliotecas Removidas:
        --> HUC;
--> Bibliotecas Adicionadas:
        --> REPORTS_2004;
        --> WINHELP;
----------------------------------------------------------
--> Nº de  chamadas a Host   detectadas e corrigidas = 3 --> (1 reports + 2 outros)
--> Nº de  get_file_name     detectados e corrigidos = 1
--> Nº de  tool_env          detectados e corrigidos = 0
--> Nº de  text_io           detectados e corrigidos = 10
--> Nº de  run_product       detectados = 0
--> Nº de  mouse triggers     detectados = 0 --> (0 when-mouse-enter + 0 when-mouse-leave)
--> Nº de  create_timer      detectados = 0
----------------------------------------------------------
--> Nº de  win_api_environment    detectados e corrigidos = 0
--> Nº de  Outros win_api         detectados (Total) = 0
----------------------------------------------------------
--> Utiliza ActiveX (procurar built-in 'Dispatch_Event')
----------------------------------------------------------
Compilação em 6i - SEM erros;
Compilação em 9i - COM erros;
----------------------------------------------------------
```

***Fig.13** – Excerto de um ficheiro de log referente a um form cuja conversão não foi efectuada com sucesso. A repetição da palavra "ERRO" na linha do nome do form permite identificar facilmente os forms com erro quando se percorre um ficheiro de log de uma conversão de muitos forms. A leitura das últimas linhas do bloco permite saber que a compilação em 6i foi efectuada com sucesso, tendo os erros ocorrido na compilação para a plataforma 9i.*

Depois do nome do *form*, surge no ficheiro de *log* uma linha referente ao carregamento da *template*. Esta será explicada nas secções 4.1.2.3 e 4.1.2.4, já que está directamente relacionada com os casos de uso que têm este como mais genérico (ver figura 5)

A informação que surge a seguir no ficheiro de *log* prende-se com as bibliotecas que estão associadas ao *form*. Aqui estão registados os nomes das bibliotecas que foram adicionadas e removidas.

Os restantes dados indicam se determinadas invocações de procedimentos ou funções são utilizadas nos *forms*, ou se, para além de serem utilizadas, foram corrigidas por invocações às novas versões dos mesmos procedimentos ou funções. As invocações dos métodos *Host* e *get_file_name*, quando encontradas, são substituídas por invocações dos métodos *cpc_hosts.cpc_host* e *cpc_get_file_name*, respectivamente. Qualquer invocação a métodos dos *packages tool_env* e *text_io* deve ser substituída por invocações aos métodos homónimos nos *packages cpc_tool_env* e *cpc_text_io*, respectivamente. As chamadas aos procedimentos *run_product* e *create_timer* são contabilizadas, assim como a existência de *triggers when-mouse-enter e when-mouse-leave* associadas aos *items* do *form*. A existência de invocações a métodos do *package win_api_environment* deve ser registada, assim como as chamadas a métodos de todos os outros *packages* cujo nome começa com *win_api*. A utilização de controlos *ActiveX* por parte do *form* deve também ser registada.

No final do ficheiro de *log* aparecem estatísticas de todo o processo de conversão, onde é referido o número de *forms* processados, número e percentagem de *forms* convertidos com sucesso e com erros, sendo ainda descriminado os *forms* com erros em 6i, em 9i ou em ambas. A figura 14 apresenta um exemplo desta porção do ficheiro de *log*.

```
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
||Estatísticas: 4 forms convertidos
||
||        --> 3 (75.0%) com sucesso
||        --> 1 (25.0%) com erros
||                --> 0 (0.0%) só com erros em 6i
||                --> 0 (0.0%) só com erros em 9i
||                --> 1 (100.0%) com erros em 6i e em 9i
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

***Fig.14** – Excerto de um ficheiro de log* apresentando as estatísticas finais.

Para além deste ficheiro *log.txt*, é criado ainda o ficheiro *logExcel.txt*. A informação aqui contida não é tão completa quanto aquela colocada no ficheiro *log.txt*, limitando-se a conter o nº de ocorrências de funções e procedimentos encontradas, separadas por caracteres *TAB*, de modo a que, quando copiadas para uma folha de cálculo Excel, cada valor apareça numa coluna diferente.

4.1.2.3 Converter recarregando a template

Este caso de utilização é uma extensão do caso anterior, "Converter *forms* para *web*". O que acrescenta ao caso do qual descende é o comportamento relativamente ao carregamento da *template*. Optou-se por considerar casos de utilização específicos para a conversão com e sem carregamento da *template* devido ao facto de esta opção poder ter um impacto muito grande nos resultados obtidos.

O diagrama de actividade apresentado para o caso de uso anterior na figura 11 é perfeitamente válido para este. Aqui, a actividade genérica "Acções relacionadas com a *template*", realçada a azul, deve ser substituída por "Recarregar a *template*". O nome da actividade é suficiente para compreender o que significa.

Relativamente aos ficheiros produzidos durante o processo de conversão, é digna de relevo a linha do ficheiro *log.txt* referente ao carregamento da *template*. Neste caso de utilização, essa linha toma a forma "*Template* foi novamente carregada", que é exactamente a que é apresentada nas figuras 12 e 13.

Esta informação é muito importante para uma análise posterior dos *forms* caso estes revelem algum comportamento anómalo aquando da sua execução na plataforma *web*, já que o carregamento da *template* não será sempre livre de problemas (ver Anexo B).

4.1.2.4 Converter sem recarregar a template

À semelhança do anterior, este caso de uso é uma extensão de "Converter *forms* para *web*", e o que acrescenta é o comportamento relativamente ao carregamento da *template*. Neste caso, a actividade a cor azul do diagrama da figura 11, "Acções relacionadas com a *template*", deve ser substituída por "NULL", já que aqui não é realizada nenhuma operação relacionada com a *template*.

A opção de considerar a existência deste caso de utilização é discutível, já que não acrescenta absolutamente nada a "Converter *forms* para *web*". Poder-se-ia ter optado por criar apenas o caso de uso "Converter *forms* para *web*", que consistiria na conversão sem o carregamento da *template*, e que deixaria de ser um caso de utilização abstracto, passando a concreto. O caso de utilização "Converter recarregando a *template*" seria uma extensão desse, acrescentando a actividade de recarregar a *template*. Neste cenário, este terceiro caso de utilização não existiria.

Contudo, optou-se por esta abordagem devido ao facto de se pretender que o ficheiro de *log* apresentasse sempre a indicação de que a *template* foi carregada ou não. Não faria sentido, portanto, que no caso de utilização "Converter *forms* para *web*", que não deveria ter a noção da existência da *template*, se fizesse uma referência à mesma, indicando que esta não foi recarregada no ficheiro de *log*.

4.1.3 Classes do Domínio

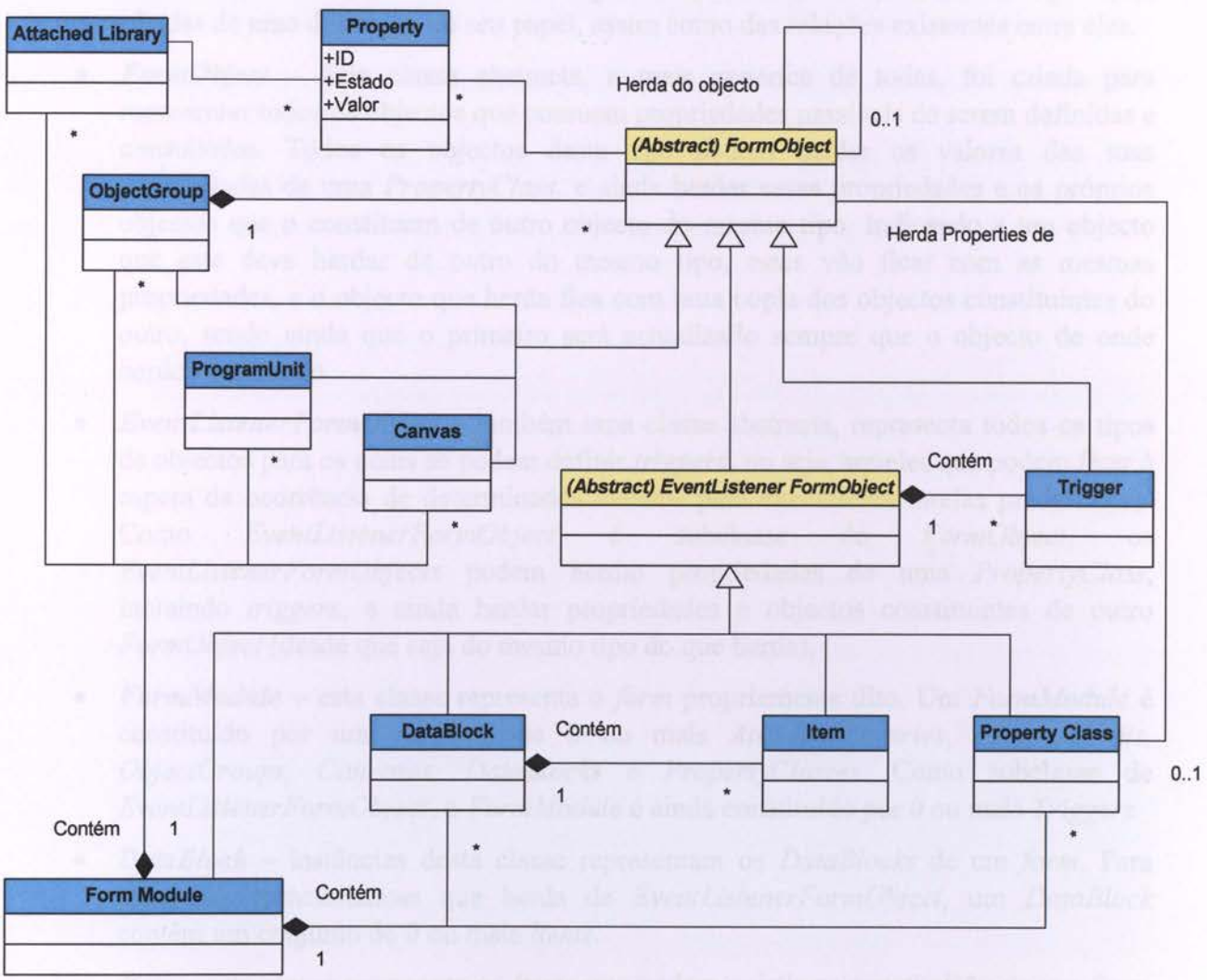


Fig.15 *– Diagrama de classes do domínio da aplicação. Note-se que apenas os conceitos abordados pela Migration Tool são aqui representados.*

O diagrama da figura 15 apresenta os conceitos do domínio da aplicação, evidenciando as relações existentes entre as várias classes.

Como os *forms* são o centro da aplicação a desenvolver, os conceitos modelados mantêm uma relação muito próxima com a constituição dos mesmos. Esta relação pode ser facilmente observada através da comparação do modelo de classes do domínio da figura 15 com o diagrama da figura 3. Ignore-se o facto de a figura 3 listar de uma forma mais exaustiva os conceitos tratados pela JDAPI (que são os mesmos existentes no desenvolvimento de aplicações no *Forms Builder*), já que o diagrama de classes de domínio contempla unicamente os conceitos relevantes para o desenvolvimento da *Migration Tool*. O diagrama da figura 15, contudo, evidencia relações que não é possível depreender analisando unicamente a figura 3.

As classes concretas deste diagrama, realçadas com a cor azul, correspondem directamente a objectos dos *forms* que podem ser tratados quer através da JDAPI, quer através do *Forms Builder* (ver capítulo 3.2 e anexo A). Os conceitos abstractos, realçados a amarelo, foram criados para agrupar do ponto de vista lógico as restantes classes, que partilham características entre si. São listadas, de seguida, as classes do domínio da aplicação, acompanhadas de uma descrição do seu papel, assim como das relações existentes entre elas.

- ***FormObject*** – Esta classe abstracta, a mais genérica de todas, foi criada para representar todos os objectos que possuem propriedades passíveis de serem definidas e consultadas. Todos os objectos deste tipo podem herdar os valores das suas propriedades de uma *PropertyClass,* e ainda herdar essas propriedades e os próprios objectos que o constituem de outro objecto do mesmo tipo. Indicando a um objecto que este deve herdar de outro do mesmo tipo, estes vão ficar com as mesmas propriedades, e o objecto que herda fica com uma cópia dos objectos constituintes do outro, sendo ainda que o primeiro será actualizado sempre que o objecto de onde herdou é mudado.
- ***EventListenerFormObject*** – também uma classe abstracta, representa todos os tipos de objectos para os quais se podem definir *triggers*, ou seja, aqueles que podem ficar à espera da ocorrência de determinados eventos para executarem tarefas predefinidas. Como *EventListenerFormObject* é subclasse de *FormObject*, os *EventListenerFormObjects* podem herdar propriedades de uma *PropertyClass*, incluindo *triggers*, e ainda herdar propriedades e objectos constituintes de outro *FormObject* (desde que seja do mesmo tipo do que herda).
- ***FormModule*** – esta classe representa o *form* propriamente dito. Um *FormModule* é constituído por um conjunto de 0 ou mais *AttachedLibraries*, *ProgramUnits, ObjectGroups, Canvases, DataBlocks* e *PropertyClasses*. Como subclasse de *EventListenerFormObject*, o *FormModule* é ainda constituído por 0 ou mais *Triggers.*
- ***DataBlock*** – instâncias desta classe representam os *DataBlocks* de um *form*. Para além das características que herda de *EventListenerFormObject*, um *DataBlock* contém um conjunto de 0 ou mais *Items*.
- ***Item*** – esta classe representa os *items* que podem existir na constituição de um *form.* Modelando o comportamento dos *forms* no *Forms Builder*, os *Items* existem sempre associados a um *DataBlock*, e possuem propriedades e *triggers* que podem ser herdados de uma *PropertyClass* ou de outro *Item*.

- ***PropertyClass*** – as instâncias desta classe não são mais do que conjuntos de propriedades e *triggers* definidos pelo utilizador, que servem o único propósito de poderem ser referenciadas por instâncias de *FormObject*, ficando esses objectos a herdar as propriedades e/ou *triggers* definidos na *PropertyClass*. Como *PropertyClass* é subclasse de *EventListenerFormObject*, uma *PropertyClass* pode herdar propriedades e *triggers* de outra *PropertyClass*. Quando um *FormObject* herda propriedades de uma *PropertyClass*, obtém unicamente os valores das propriedades que esse *FormObject* possui. As restantes propriedades que possam eventualmente existir na *PropertyClass* são ignoradas. No *Forms Builder*, as *PropertyClasses* podem ser usadas, por exemplo, para definir as propriedades visuais (como cor, tamanho, etc.) de determinado tipo de *items* (como botões). Indicando a esses *items* que devem herdar as propriedades da *PropertyClass* criada, todos eles vão então ter o mesmo aspecto gráfico, e serão todos alterados se a *PropertyClass* o for também.
- ***ProgramUnit*** – representa as *ProgramUnits* de um *form*. Estes objectos são uns dos vários constituintes dos *forms*, tratando-se de procedimentos e funções PL/SQL, ou *packages* de funções e procedimentos. Podem ser do tipo *Procedure, Function, PackageSpec ou PackageBody (*um *PackageSpec* contém a definição de variáveis e assinatura dos métodos implementados no *PackageBody* que podem ser acedidos do exterior do *package)*. Como realização directa da classe abstracta *FormObject*, *ProgramUnit* possui um conjunto de *Properties*, que podem ser herdadas de uma *PropertyClass* ou de outra *ProgramUnit*, mas não possui *triggers* nem, consequentemente, a capacidade de os herdar. Se a *PropertyClass* a partir da qual uma *ProgramUnit* herda as suas propriedades incluir a definição de *triggers* para além de *Properties*, os *triggers* são ignorados, sendo herdadas apenas as propriedades relevantes para a *ProgramUnit*.
- ***ObjectGroup*** – a funcionalidade dos *ObjectGroups* é de certa forma semelhante à das *PropertyClasses*. Estes servem de repositório de objectos, não tendo qualquer carácter funcional. Existem para facilitar a reutilização de objectos. Dentro de um *ObjectGroup* de determinado *form* pode ser colocado qualquer *FormObject* do mesmo *form*. Sendo o próprio *ObjectGroup* um *FormObject*, este pode herdar as suas propriedades de uma *PropertyClass*, e herdar os objectos que agrupa de outro *ObjectGroup*. Existe, contudo, uma limitação que não foi modelada no diagrama de classes da figura 20 e que deve ser realçada: dentro de um *ObjectGroup* pode ser colocado qualquer *FormObject* à excepção de outro *ObjectGroup* (um *ObjectGroup* não pode conter outro *ObjectGroup*). Os objectos que se encontram dentro de um *ObjectGroup* existem obrigatoriamente no mesmo *form*. Para se poder adicionar um objecto a um *ObjectGroup*, esse objecto deve ser criado em primeiro lugar no *form*. Depois de um *ObjectGroup* estar criado, pode ser utilizado para copiar todos os objectos que contém para outro *form*, através de dois métodos diferentes: cópia simples, ou herança. Em qualquer um dos casos, o *ObjectGroup* passa a existir também no novo *form*, assim como todos os objectos que contém. Se o *ObjectGroup* contiver um *canvas*, por exemplo, esse *canvas* fica automaticamente visível na lista de *canvases* do *form*. Se o método usado for o de cópia simples, o que acontece é que os objectos são todos copiados normalmente, não se mantendo qualquer relação entre os objectos no novo *form* e os objectos originais. Se se utilizar o método de herança, o *ObjectGroup* criado no novo *form* mantém uma referência ao *ObjectGroup* original, mantendo-se

sincronizado com este. Se o *ObjectGroup* no *form* original for alterado, as alterações vão-se repercutir no *form* que utiliza a versão herdada do mesmo.

- ***Canvas*** – como subclasse de *FormObject*, *Canvas* possui um conjunto de propriedades que pode herdar de uma *PropertyClass* ou de outro *canvas*. Num *form*, um *canvas* tem o papel de servir de superfície de desenho, tendo todos os componentes visuais que estar inseridos dentro de um *canvas*. Existem mais conceitos relacionados com os *canvas* e as interfaces gráficas dos *forms*, como *Windows ou Graphics*, mas estes não são aqui explicados por não terem relevância para a *Migration Tool*. Os objectos do tipo *canvas,* contudo, são tratados pela aplicação de migração.
- ***AttachedLibrary*** – ao contrário das restantes classes que possuem um conjunto de *Properties*, esta não constitui uma realização da classe abstracta *FormObject*. Isto deve-se ao facto de, apesar de uma biblioteca (instância de *AttachedLibrary*) ter propriedades que podem ser manipuladas (na verdade, apenas uma propriedade de *AttachedLibrary* pode ser alterada, e trata-se apenas de um comentário que o programador possa eventualmente associar à biblioteca; as restantes propriedades da biblioteca podem apenas ser consultadas), estas não podem ser herdadas de uma *PropertyClass* ou de outra *AttachedLibrary*. Uma instância de *AttachedLibrary* representa uma biblioteca de funções e procedimentos externa que pode ser associada a um *form*, para que este possa utilizar os métodos nela definidos.
- ***Property*** – as *Properties* definem as características dos *forms* e dos seus constituintes. Todos os tipos de objectos aqui modelados possuem um conjunto de propriedades que podem ser definidas e consultadas. E, à excepção das *AttachedLibraries*, estes objectos podem ainda herdar os valores destas propriedades de uma *PropertyClass* ou de outros objectos do mesmo tipo. Estas propriedades possuem um valor que pode ser de várias naturezas, como sequência de caracteres, número inteiro ou decimal, data, cor no formato RGB, etc. E todas elas possuem um valor por defeito, que é utilizado se nenhum outro for definido. Existe ainda a noção de "estado" de uma propriedade, que está relacionada com o valor que a *Property* assume. As propriedades encontram-se sempre num de 4 estados possíveis:
    - *Default Value* – o valor da propriedade não foi definido, e assume o seu valor por defeito.
    - *Inherited Value* – o objecto ao qual a *Property* pertence herdou os valores das suas propriedades de uma *PropertyClass* ou de outro objecto do mesmo tipo, e a propriedade em questão assume esse valor (o valor não foi alterado pelo utilizador, mas não corresponde ao valor por defeito).
    - *Overriden Default Value* – o valor da propriedade, que era inicialmente o valor por defeito, foi alterado (pelo utilizador em *design-time*, ou no código da aplicação em *runtime*).
    - *Overriden Inherited Value* – o objecto ao qual a *Property* pertence herdou os valores das suas propriedades de uma *PropertyClass* ou de outro objecto do mesmo tipo. Contudo, esse valor foi alterado (pelo utilizador em *design-time*, ou no código da aplicação em *runtime*).

A JDAPI prevê ainda a possibilidade de um estado *Unknown*, mas não foi possível esclarecer as circunstâncias em que uma propriedade assume este estado, até porque este não existe no *Forms Builder*.

- ***Trigger*** – os *triggers* são procedimentos escritos em PL/SQL associados a determinados tipos de objectos. Em termos de implementação, são em quase tudo semelhantes às *ProgramUnits*. As diferenças residem no facto de os *triggers* não possuírem valor de retorno (daí não assumirem a forma de funções, mas apenas de procedimentos), e no facto de estes constituírem sempre procedimentos simples, enquanto que as *ProgramUnits* podem assumir a forma de *packages* de procedimentos ou funções. As classes de objectos que possuem *triggers* são aquelas que descendem da classe abstracta *EventListenerFormObject*, nomeadamente *FormModule, DataBlock, Item* e *PropertyClass*. Existe uma importante diferença na forma como estas classes utilizam os *triggers*. As classes *FormModule, DataBlock* e *Item* são verdadeiros e*vent listeners*, ficando atentas, em *runtime*, à ocorrência dos eventos para os quais têm *triggers* definidos, invocando esses mesmos *triggers* sempre que um desses eventos ocorra. Por outro lado, a *PropertyClass*, como já foi atrás referido, não passa de um repositório de *Properties* e *Triggers*, servindo o único propósito de permitir que outras classes herdem os valores dessas propriedades e *triggers*.

Nesta listagem e descrição das classes do domínio da aplicação foram referidos três tipos de herança, que se podem resumir da seguinte forma:

1. Herança de propriedades – todos os *FormObjects* podem referenciar uma *PropertyClass*, herdando os valores das propriedades comuns a esse *FormObject* e à *PropertyClass*.
2. Herança de objectos – todos os *FormObjects* podem referenciar outro objecto da mesma classe, herdando os valores das propriedades desse objecto, assim como todos os objectos (e respectivas propriedades) que façam parte da constituição do mesmo.
3. Herança através de um *ObjectGroup* – um *ObjectGroup* e todos os objectos que este contém podem ser copiados para outro *form*, mantendo uma relação de herança com os objectos originais, de modo a que as alterações efectuadas nos objectos originais sejam repercutidas nos objectos do novo *form*.

4.1.4 Arquitectura

Nesta secção será apresentada a arquitectura da aplicação, expondo a estrutura de alto nível do sistema. A secção 4.1.4.1 descreve a arquitectura lógica, e na secção 4.1.4.2 é apresentada a arquitectura física. Sempre que necessário, são dadas explicações acerca das decisões tomadas ao nível da arquitectura.

4.1.4.1 Arquitectura Lógica

A abordagem seguida para apresentar esta arquitectura consiste em começar por uma visão de alto nível sem grandes detalhes, descendo-se de seguida aos pormenores. É feita apenas uma decomposição horizontal, sendo deixada de parte a divisão vertical, porque não faz sentido dividir a *Migration Tool* em grupos funcionais menores que o todo que a constitui.

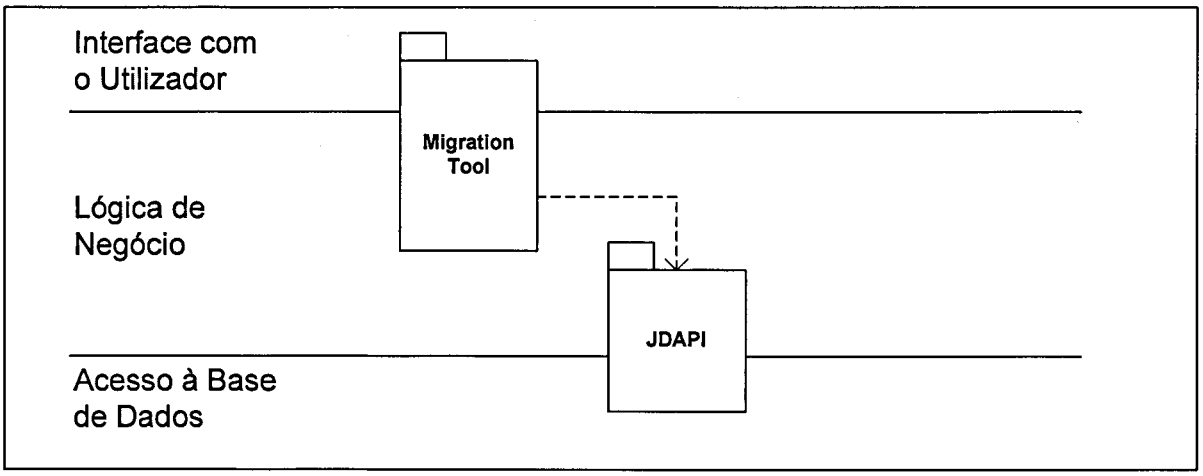


***Fig.16** – Diagrama de pacotes lógicos, ilustrando a arquitectura de alto nível do sistema. É visível a decomposição em camadas horizontais.*

O diagrama da figura 16 ilustra a decomposição horizontal do sistema em três camadas de implementação: "Interface com o Utilizador", "Lógica de Negócio" e "Acesso à Base de Dados". Note-se que a *Migration Tool* propriamente dita não é constituinte da camada de acesso à Base de Dados. Esta responsabilidade é delegada na totalidade para a JDAPI. A *Migration Tool* disponibiliza a interface com o utilizador, implementando a lógica de negócio sobre as funcionalidades disponibilizadas pela JDAPI. Os acessos à Base de Dados são efectuados por esta última de forma totalmente transparente para a *Migration Tool*.

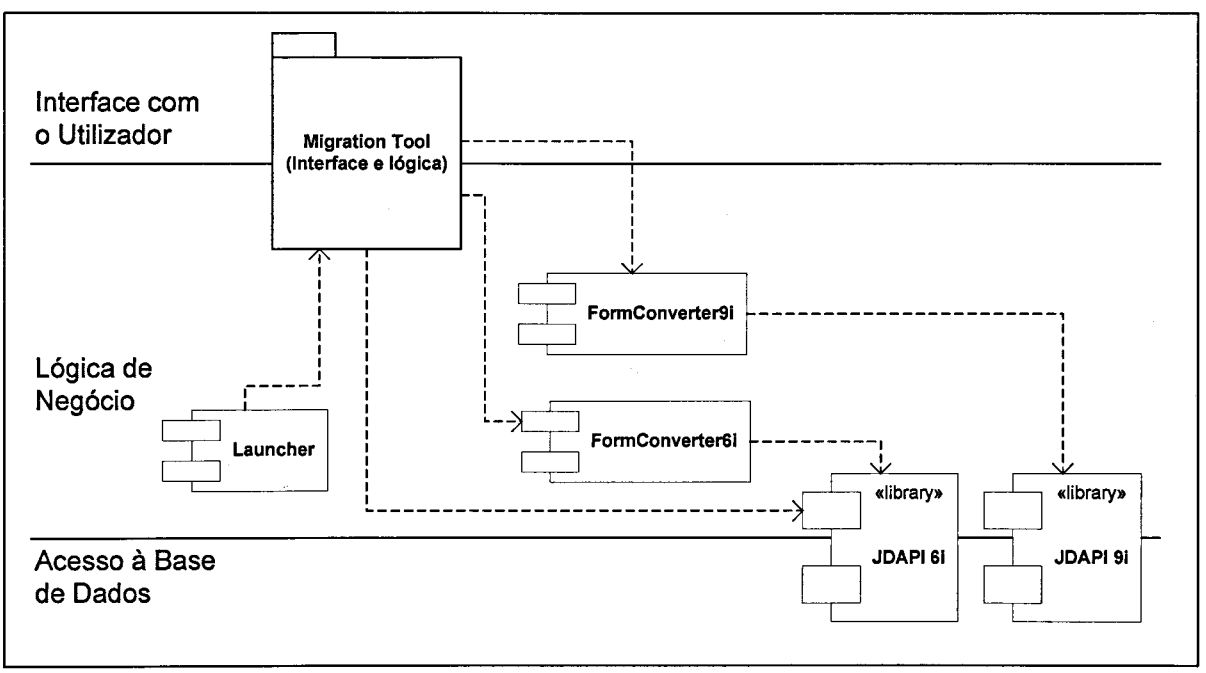


***Fig.17** – Visão detalhada da arquitectura lógica do sistema, ilustrando a sua decomposição em camadas horizontais.*

Na figura 17 é visível a mesma arquitectura apresentada na figura anterior, desta vez com um maior nível de detalhe. Neste diagrama continua a evidenciar-se a decomposição horizontal

do sistema em 3 camadas, tendo cada um dos pacotes do diagrama da figura 16 sido divido em vários dos seus constituintes. A JDAPI surge agora sob a forma de dois componentes distintos, deixando já antever a sua estrutura física. Faz sentido, contudo, que esta divisão seja aqui feita porque são de facto utilizadas duas versões da JDAPI, independentemente da forma como esta é distribuída fisicamente.

A *Migration Tool* aparece dividida em 4 componentes. "Launcher" é o responsável por iniciar a *Migration Tool* propriamente dita, preparando as variáveis de ambiente necessárias para o correcto funcionamento da aplicação. A necessidade deste componente é explicada com mais detalhe no capítulo 5.

O pacote identificado por "*Migration Tool* (Interface e lógica)" representa o corpo principal da aplicação. Este pacote contém a interface gráfica, assim como a lógica mais directamente relacionada com esta (por exemplo, o ciclo que percorre os vários *forms* seleccionados através da interface gráfica), sendo o centro da aplicação.

"FormConverter6i" e *"FormConverter9i"* são os componentes que efectuam a conversão e compilação dos *forms* para as plataformas cliente-servidor e *web*, respectivamente. São estes que interagem mais directamente com a JDAPI, cada qual com a versão que lhe corresponde.

Note-se que existe uma relação entre o pacote principal da *Migration Tool* e o componente *"JDAPI 6i"*. Esta existe porque, através da interface gráfica, é possível estabelecer directamente uma ligação à BD, sem se ter ainda qualquer *form* em processamento. Esta ligação à Base de Dados é efectuada através da versão 6i da JDAPI. Esta questão é explicada no capítulo 5.

4.1.4.2 Arquitectura Física

Nesta secção é documentada a arquitectura física de alto nível do sistema. Esta arquitectura é ilustrada no diagrama da figura 18. O sistema é composto por duas máquinas: o servidor com a Base de Dados Oracle à qual os *forms* acedem, e a máquina cliente, onde a *Migration Tool* e a JDAPI se encontram instaladas. Para que a execução da *Migration Tool* seja possível, é também necessária uma instância da *Java Virtual Machine* na máquina cliente.

É necessário que, na máquina servidora, exista uma instância de um Sistema de Gestão de Bases de Dados Oracle, que permita o acesso à Base de Dados sobre a qual os *forms* a serem processados serão executados.

Fisicamente, uma distribuição da *Migration Tool* é constituída por dois ficheiros *jar* correspondentes às duas versões da JDAPI, um ficheiro *jar* que consiste no pacote principal da *Migration Tool ("migracao.jar")* contendo várias classes Java, e ainda dois componentes que consistem também em classes Java, "*Launcher.class*" e "*FormConverter9i.class*". Estes dois últimos são distribuídos fora do package "*migracao.jar*".

Note-se que a figura 18 não constitui um diagrama de distribuição da *Migration Tool* completo. Para que esse diagrama pudesse ser considerado dessa forma, seria necessário acrescentar-lhe a pasta *bin* da instalação do *Oracle Forms Builder 6i* com o *patch* para suportar a JDAPI. Este directório, com todo o seu conteúdo, deve estar localizado no directório raiz da instalação da *Migration Tool*. Sem ele, não seria possível utilizar a JDAPI 6i, pois o ficheiro "*f60jdapi.jar*" utiliza bibliotecas e ficheiros executáveis existentes nesse directório. O único componente desse directório que é referido no diagrama é o executável "*ifcmp60.exe*", pois é o único que é utilizado directamente pela *Migration Tool* (os restantes são utilizados apenas pela JDAPI).

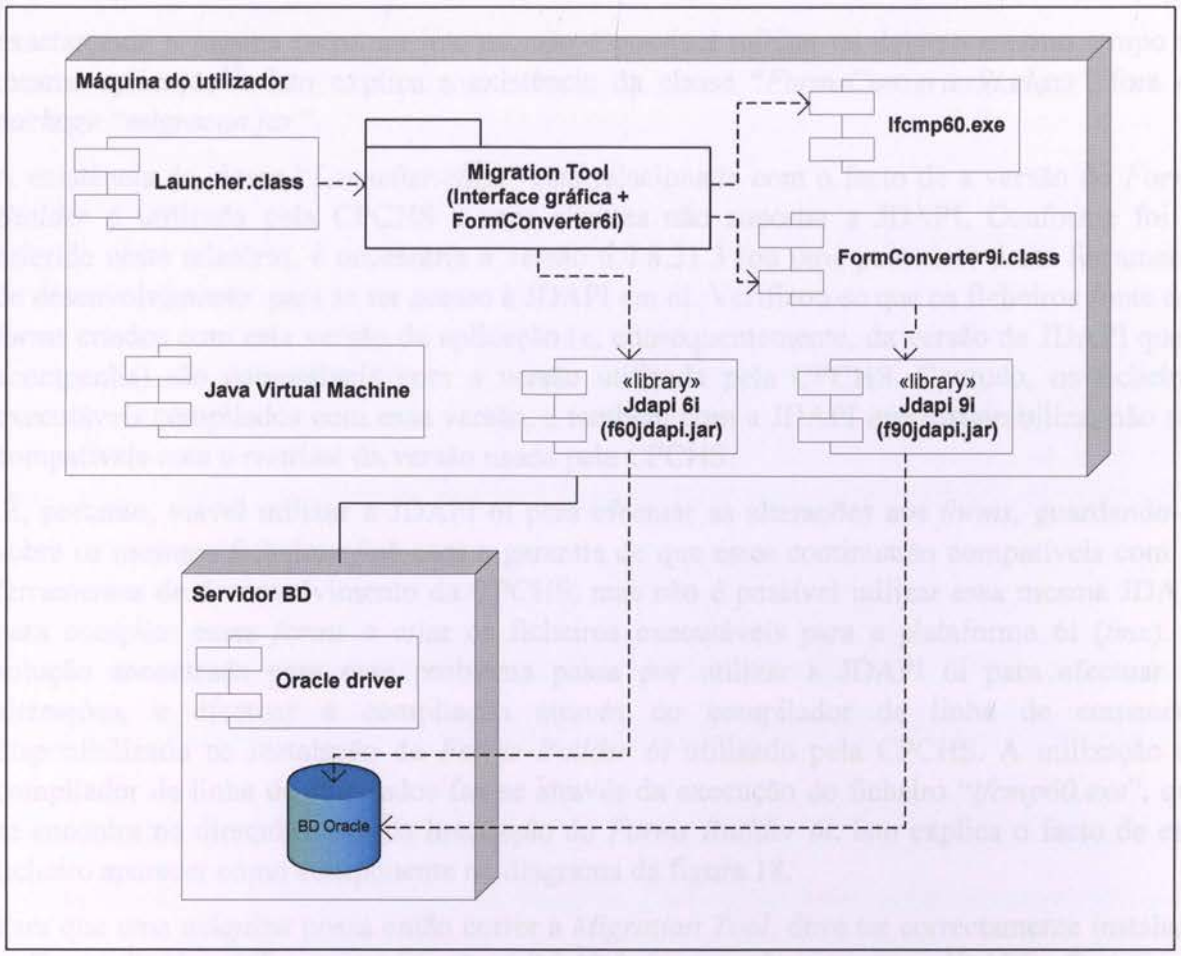


***Fig.18** – Estrutura física de alto nível do sistema.*

O diagrama da arquitectura lógica apresentado na figura 17 é já uma boa aproximação à estrutura física da *Migration Tool.* O componente *"FormsConverter6i"* referido nesse diagrama, contudo, não é contemplado nesta visão da arquitectura física. Isto acontece porque, apesar de *"FormsConverter6i"* ter um correspondente físico, que consiste numa classe Java, este encontra-se dentro do pacote *"migracao.jar"*.

A opção de se ter as classes *"Launcher.class"* e *"FormConverter.class"* como classes isoladas, e não como integrantes do *package* principal da *Migration Tool,* deve-se ao facto de estes terem que ser executados como aplicações distintas, cada qual na sua própria instância da *JavaVirtual Machine*.

Durante uma execução da *Migration Tool* são lançadas, portanto, três instâncias da *Java Virtual Machine*: uma quando o utilizador inicia a aplicação, através da execução da classe *"Launcher.class"*; outra iniciada logo de seguida por esta classe, onde é executado o conteúdo do *package "migracao.jar"*, que contém a interface gráfica; e, finalmente, uma outra instância da máquina virtual quando o processo de compilação em 9i é iniciado, sendo executado o ficheiro *"FormConverter9i.class"*. Note-se que as alterações aos *forms* e a compilação em 6i são efectuadas na mesma instância em que corre a interface gráfica, a do *package "migracao.jar"*.

Esta situação é motivada pela necessidade de se executar as alterações aos *forms* usando a versão 6i da JDAPI, e a compilação em 9i usando a versão 9i da JDAPI. Como estes dois *packages* (JDAPI 6i e JDAPI 9i) são distribuídos sob a forma de ficheiros *jar* com

exactamente a mesma estrutura interna, não é possível utilizar os dois ao mesmo tempo na mesma aplicação[15]. Isto explica a existência da classe "*FormsConverter9i.class*" fora do *package "migracao.jar"*.

A existência da classe "*Launcher.class*" está relacionada com o facto de a versão do *Forms Builder 6* utilizada pela CPCHS e seus clientes não suportar a JDAPI. Conforme foi já referido neste relatório, é necessária a versão 6.0.8.21.3 (ou uma posterior) desta ferramenta de desenvolvimento para se ter acesso à JDAPI em 6i. Verificou-se que os ficheiros fonte dos *forms* criados com esta versão da aplicação (e, consequentemente, da versão da JDAPI que a acompanha) são compatíveis com a versão utilizada pela CPCHS. Contudo, os ficheiros executáveis compilados com essa versão, e também com a JDAPI que disponibiliza, não são compatíveis com o *runtime* da versão usada pela CPCHS.

É, portanto, viável utilizar a JDAPI 6i para efectuar as alterações aos *forms*, guardando-as sobre os mesmos ficheiros *fmb* com a garantia de que estes continuarão compatíveis com as ferramentas de desenvolvimento da CPCHS, mas não é possível utilizar essa mesma JDAPI para compilar esses *forms* e criar os ficheiros executáveis para a plataforma 6i (*fmx*). A solução encontrada para esse problema passa por utilizar a JDAPI 6i para efectuar as alterações, e efectuar a compilação através do compilador de linha de comandos disponibilizado na instalação do *Forms Builder 6i* utilizado pela CPCHS. A utilização do compilador de linha de comandos faz-se através da execução do ficheiro "*ifcmp60.exe*", que se encontra no directório *bin* da instalação do *Forms Builder 6i*. Isto explica o facto de este ficheiro aparecer como componente no diagrama da figura 18.

Para que uma máquina possa então correr a *Migration Tool*, deve ter correctamente instalado o *Forms Buider 6i*[16], *na sua versão* 6.0.8.13.0 (a que não suporta a JDAPI). O correcto funcionamento deste *Forms Builder 6i* obriga a que o seu directório *bin* conste da variável de ambiente "*PATH*". Contudo, também a execução da JDAPI exige que o directório *bin* da versão 6.0.8.21.3 (a que suporta a JDAPI) se encontre na variável de ambiente "*PATH*". E não é possível que os dois directórios constem ao mesmo tempo no "*PATH*" do sistema[17]. Não é também possível optar por colocar um directório ou outro, pois haverá sempre uma versão do *Forms Builder 6i* que não irá funcionar correctamente.

A solução encontrada para este problema consiste então em não alterar a variável "*PATH*" do sistema, ficando lá o directório *bin* da versão do *Forms Builder 6i* utilizado para o desenvolvimento na CPCHS (o que não suporta a JDAPI), aí colocado no processo de intalação da ferramenta. Assim, essa máquina pode ser utilizada para desenvolvimento, tal como seria possível se a *Migration Tool* não estivesse instalada.

Os ficheiros necessários ao funcionamento da JDAPI 6i encontram-se na sua totalidade no directório *bin* da versão 6.0.8.21.3 do *Forms Builder*. Fez-se então uma instalação dessa versão do *Forms Builder* numa máquina de testes, com o único propósito de se copiar todo o

---

[15] A inclusão dos dois ficheiros *jar* no *CLASSPATH* da máquina virtual onde a aplicação fosse executada faria com que todas as referências às classes de qualquer um dos *packages* fossem redireccionadas para aquele cujo ficheiro *jar* aparecesse em primeiro lugar no *CLASSPATH*.

[16] O correcto funcionamento da *Migration Tool* exige também que o *Forms Buider 9i* esteja instalado, caso contrário não é possível utilizar a JDAPI 9i.

[17] Na verdade, é possível que ambos os directórios apareçam na variável de ambiente "*PATH*". Contudo, será sempre utilizado aquele que aparecer em primeiro lugar, e nunca o outro.

conteúdo da pasta *bin* dessa instalação para o directório da *Migration Tool.* E, ao invés de se iniciar a *Java Virtual Machine* com acesso à variável "*PATH*" do sistema operativo, criou-se a classe "*Launcher.class*" que, ao invés de invocar directamente uma nova instância da *Virtual Machine* para correr a aplicação, cria uma instância da linha de comandos (através do comando *cmd.exe*), na qual define um novo valor para a variável de ambiente "*PATH*", já a apontar para o directório *bin* da versão com suporte para a JDAPI. Só aí, e ainda dentro dessa instância da linha de comandos, é que é iniciada a *Migration Tool.*

Desta forma, esta "vê" a variável "*PATH*" aqui definida, sem que o "*PATH*" do sistema tenha sido alterado. Fica então garantida a possibilidade de utilizar a JDAPI 6i, já que as classes do *package* "*f60jdapi.jar*" não vão ter problemas em encontrar os ficheiros de que precisam (que se encontram no directório *bin* com suporte para a JDAPI), ao mesmo tempo que se mantém inalterado o "*PATH*" do sistema, que continua a apontar para o directório *bin* da versão sem suporte para a JDAPI.

O único cuidado a ter com este esquema é o de repor a variável de ambiente "*PATH*" a apontar para o *bin* da versão antiga na instância da linha de comandos criada para efectuar a compilação dos *forms* em 6i. Isto porque, se a *Java Virtual Machine* "vê" o "*PATH*" alterado, os processos por si criados (como é o caso do que for criado para efectuar a compilação através da linha de comandos) "verão" esse mesmo "*PATH*" também.

4.2 O novo *Processo Clínico Electrónico*

O trabalho desenvolvido no âmbito do *Processo Clínico Electrónico* não teve uma estrutura tão bem definida como o da *Migration Tool.* Os objectivos a atingir nesta fase não eram muito específicos, dado que não era objectivo deste estágio ter a nova versão do *Processo Clínico Electrónico* concluída, mas apenas participar no seu desenvolvimento.

Contudo, a criação do estilo visual para as novas interfaces gráficas ficou à exclusiva responsabilidade do estagiário, sendo esta a área sobre a qual incidirá principalmente a descrição que é feita neste relatório. Relativamente aos restantes desenvolvimentos, estes serão apresentados sob a forma de listagem das novas funcionalidades do *Processo Clínico Electrónico.*

Pelos motivos descritos nos parágrafos anteriores, este assunto é tratado em anexos a este documento, onde é possível fazê-lo com uma estruturação menos rígida, mais adequada ao trabalho desenvolvido. O novo estilo gráfico (*look-and-feel*) do *Processo Clínico Electrónico* é abordado no Anexo C, enquanto que as funcionalidades implementadas na nova versão da aplicação são descritas no Anexo D.

5 Documentação do Protótipo

Foi desenvolvido um protótipo funcional da *Migration Tool*, cuja documentação é apresentada neste capítulo. A secção 5.1 apresenta uma descrição geral das classes que consituem os vários componentes da aplicação. A secção 5.2 documenta detalhadamente cada um dos componentes referidos na secção 5.1, do ponto de visto externo e interno (especificação e implementação).

5.1 Descrição geral

Todos os casos de utilização descritos anteriormente utilizam a totalidade das classes implementadas. A figura 19 apresenta os componentes desenvolvidos, descriminando as várias classes que os constituem. Cada uma destas classes corresponde de facto a uma classe Java. Este diagrama contempla não só os componentes desenvolvidos no projecto de estágio, mas também os componentes com os quais estes interagem, nomeadamente os dois *packages* da JDAPI, assim como o compilador de linha de comandos *ifcmp60.exe*

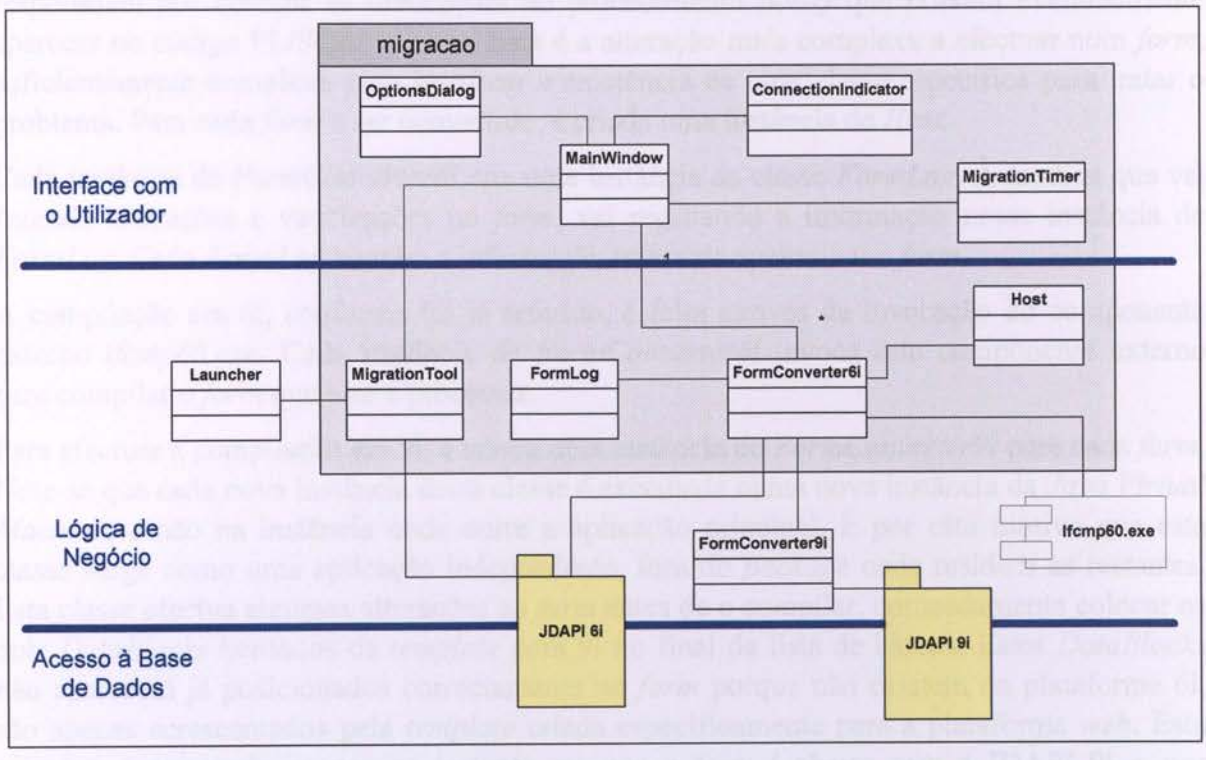


***Fig.19** – Diagrama de classes do protótipo implementado. Cada classe do diagrama corresponde a uma classe Java, e os pacotes JDAPI 6i e JDAPI 9i correspondem aos ficheiros jar que implementam essas bibliotecas. As classes estão organizadas de forma a evidenciar a camada lógica a que cada uma pertence.*

O pacote *migracao* constitui o componente principal da aplicação, "atravessando" as camadas de interface com o utilizador e lógica de negócio.

A camada de interface com o utilizador é constituída na sua totalidade por classes deste pacote. *MainWindow* é a classe principal da interface com o utilizador, constituindo a janela principal da aplicação. *OptionsDialog* constitui a janela do painel de opções. A classe

ConnectionIndicator é um componente gráfico que implementa o indicador de ligação à Base de Dados existente na janela principal da aplicação. E *MigrationTimer* é outro componente gráfico, utilizado para implementar o medidor de tempo que existe também na janela principal. Este componente é composto pelo medidor de tempo decorrido, e pelo indicador da estimativa do tempo restante.

As restantes classes do pacote *migracao* pertencem à camada de lógica de negócio. Tratam-se das classes *MigrationTool, FormConverter6i, Host* e *FormLog*. *MigrationTool* é a classe principal da aplicação, daí ter o nome da mesma. A aplicação é iniciada através da instanciação desta classe, sendo ela a responsável por despoletar a criação da interface gráfica apresentada ao utilizador. É também ela que gere a ligação à Base de Dados, e disponibiliza a maioria dos serviços utilizados pelas outras classes (à excepção da conversão dos *forms* propriamente dita).

Na classe *FormConverter6i* são implementadas as funcionalidades de conversão dos *forms*. É criada uma instância desta classe para cada *form* a processar. Esta é responsável por efectuar as alterações, por guardá-las novamente no ficheiro *fmb*, e por iniciar a compilação.

O propósito da classe *Host* é o de efectuar uma das várias alterações necessárias. Esta é responsável por corrigir as invocações ao procedimento *host()* que possam eventualmente aparecer no código PL/SQL do *form*. Esta é a alteração mais complexa a efectuar num *form*, suficientemente complexa para justificar a existência de uma classe específica para tratar o problema. Para cada *form* a ser convertido, é criada uma instância de *Host*.

Cada instância de *FormConverter6i* cria uma instância da classe *FormLog*. À medida que vai fazendo alterações e verificações no *form*, vai registando a informação nessa instância de *FormLog*. Cada *FormLog* contém a informação referente apenas a um *form*.

A compilação em 6i, conforme foi já referido, é feita através da invocação do componente externo *ifcmp60.exe*. Cada instância de *FormConverter6i* invoca este componente externo para compilar o *form* que está a processar.

Para efectuar a compilação em 9i, é criada uma instância de *FormConverter9i* para cada *form*. Note-se que cada nova instância desta classe é executada numa nova instância da *Java Virtual Machine*, e não na instância onde corre a aplicação principal. É por este motivo que esta classe surge como uma aplicação independente, fora do *package* onde residem as restantes. Esta classe efectua algumas alterações ao *form* antes de o compilar, nomeadamente colocar os dois *DataBlocks* herdados da *template* para 9i no final da lista de blocos. Estes *DataBlocks* não aparecem já posicionados correctamente no *form* porque não existem na plataforma 6i, são apenas acrescentados pela *template* criada especificamente para a plataforma *web*. Esta *template* é carregada apenas no instante em que o *form* é aberto com a JDAPI 9i e, por defeito, os novos objectos são colocados no início da lista de objectos do mesmo tipo do *form*. Esta alteração, contudo, não pode ser guardada, e deve ser feita sempre que se abre o *form* com a JDAPI 9i.

Também fora do *package migracao* e ainda na camada de lógica de negócio, existe a classe *Launcher*. Esta constitui uma aplicação independente, executada numa *Java Virtual Machine* diferente daquela em que corre a aplicação principal, à semelhança da classe *FormsConverter9i*. Esta classe é o ponto de entrada para a *Migration Tool*, sendo a classe executada quando o utilizador inicia a aplicação, e responsável por iniciar a instância da *Java Virtual Machine* onde será executada a aplicação principal.

As duas versões da JDAPI servem de base à *Migration Tool*, sendo estas as responsáveis por gerir os acessos à Base de Dados.

As interfaces com o utilizador disponibilizadas pela *Migration Tool* são muito simples. Limitam-se à janela principal (ver figuras 7 e 8), e ao painel de opções (ver figura 6). Estas interfaces foram já apresentadas no capítulo de especificação de requisitos. O funcionamento da aplicação passa por abrir o painel de opções, para se definirem os parâmetros de ligação à Base de Dados e a localização do ficheiro da *template*, caso se opte por carregá-la novamente. Estes parâmetros não têm que ser definidos todas as vezes que se pretenda utilizar a aplicação, pois são guardados de umas invocações para as outras num ficheiro de configuração.

Depois de se sair do painel de opções, o utilizador deve *clicar* no botão "Abrir" da barra de ferramentas da janela principal e, na janela de selecção de ficheiros que lhe é apresentada, seleccionar os ficheiros *fmb* correspondentes aos *forms* que pretende converter. A aplicação pergunta-lhe de seguida se pretende estabelecer a ligação à Base de Dados. Depois de responder a esta pergunta (normalmente com uma resposta afirmativa), o utilizador só tem que aguardar que o processo de conversão fique concluído. Uma variante deste processo consiste em estabelecer a ligação à Base de Dados manualmente antes de abrir os *forms*. Neste caso, o utilizador *clica* no botão da barra de ferramentas que efectua a ligação, e só depois selecciona os *forms* a converter.

5.2 Documentação dos componentes

Esta secção documenta detalhadamente os vários componentes, apresentando a sua especificação, e também vários detalhes de implementação.

5.2.1 MigrationTimer

extends JPanel

implements ActionListener

Descrição: Componente gráfico que apresenta o tempo decorrido desde o início do processo de conversao, assim como uma estimativa do tempo restante, com base no tempo gasto para converter os *forms* anteriores.

Constructor Detail

```
public MigrationTimer()
```

Construtor *default* para esta classe.

Method Detail

```
public void actionPerformed(ActionEvent evt)
```

Método definido na interface *ActionListener*. É invocado uma vez por segundo para actualizar os valores do tempo decorrido e tempo restante.

```
public void startTimer(int totalForms)
```

Inicia a contagem do tempo (decorrido).

Parameters:

`totalForms` - Número total de *forms* a converter.

`public void` **`pauseTimer()`**

Interrompe a contagem do tempo.

`public void` **`incNumFormsConvertidos()`**

Deve ser invocado sempre que a conversão de um novo *form* é concluída.

`public void` **`zeroRemainingTime()`**

Coloca o indicador do tempo restante a zero. Como o tempo restante é uma estimativa, pode acontecer que no final do processo de conversão o indicador do tempo restante contenha um valor diferente de zero. Este método deve então ser invocado no final da conversão, para garantir que o valor apresentado é zero.

Esta classe contém um objecto do tipo *Timer*, configurado para invocar o método *actionPerformed* desta classe (que implementa a interface *ActionListener*) uma vez por segundo. No método *actionPerformed* é feita a actualização do contador de tempo decorrido a cada invocação. O cálculo do tempo restante é feito da forma a seguir descrita. Se o número de *forms* convertidos é zero, então não apresenta qualquer valor; se o nº de *forms* convertidos é maior que zero, e apenas quando ocorre uma mudança no nº de *forms*, utiliza a seguinte fórmula para obter a estimativa do tempo restante:

tempoRestante = ((totalForms - formsConvertidos) * tempoDecorrido) / formsConvertidos;

Nos restantes casos, quando o nº de *forms* convertidos é maior que zero, e não houve mudança do nº de *forms* convertidos, diminui 1 segundo ao tempo restante a cada invocação.

5.2.2 ConnectionIndicator

extends JPanel

Descrição: Componente gráfico que apresenta um indicador de estado, que pode tomar um de dois valores. Graficamente, consiste num círculo vermelho ou verde, de acordo com o estado.

Constructor Detail

`public` **`ConnectionIndicator`**`(boolean estado)`

Construtor para esta classe.

Parameters:

`estado` - Indica o estado inicial: *true* significa "ligado"; *false* significa "desligado".

`public` **`ConnectionIndicator()`**

Construtor *default* para esta classe. Assume o estado *false* por defeito, que corresponde a "desligado".

Method Detail

`public boolean` **`getEstado`**`()`

Devolve o estado actual do indicador de estado.

`public void` **`setEstado`**`(boolean estado)`

Define o valor do estado do indicador.

`public void` **`switchState`**`()`

Troca o valor do estado do indicador.

`public Dimension` **`getPreferredSize`**`()`

Devolve um objecto do tipo *Dimension* indicando qual é o tamanho que o componente deve ter na interface gráfica.

5.2.3 OptionsDialog

extends JDialog

Descrição: Constitui o painel de opções da *Migration Tool.*

Constructor Detail

```
public OptionsDialog(MainWindow owner,
                     String title,
                     boolean modal)
```

Construtor para esta classe.

5.2.4 MainWindow

extends JFrame

Descrição: Constitui a janela principal da *Migration Tool.*

Constructor Detail

`public` **`MainWindow`**`(`*`Migration Tool`* `parent)`

Construtor para esta classe.

Parameters:

`parent` - A instância de *Migration Tool* que criou esta *MainWindow*.

Method Detail

`public String` **`getTemplatePath`**`()`

Devolve o *path* da *template*.

`public void` **`setTemplatePath`**`(String` *`path`*`)`

Define o *path* da *template*.

`public String` **`getClasspath90()`**

Devolve o *CLASSPATH* a ser usado na invocação do *FormConverter9i*.

`public String` **`getConnectionString()`**

Devolve a *Connection String* para ligar à BD, na forma: *user/password@database*.

`public String` **`getUser()`**

Devolve o nome de utilizador para ligação à BD.

`public String` **`getPassword()`**

Devolve a *password* para ligação à BD.

`public String` **`getDatabase()`**

Devolve o nome da BD à qual a ligação deve ser efectuada.

`public void` **`setUser`**`(String user)`

Define o utilizador a utilizar na ligação à BD.

`public void` **`setPassword`**`(String pass)`

Define a *password* a usar na ligação à BD.

`public void` **`setDatabase`**`(String db)`

Define o nome da BD à qual deve ser estabelecida a ligação.

`private void` **`fazConversao`**`(File[] files)`

Consiste no corpo da *thread* criada para levar a cabo todo o processo de conversão, deixando a *thread* da interface gráfica livre para a sua actualização.

Parameters:

`files` - array de *forms* (ficheiros *fmb*) a serem convertidos

---

`public String` **`buildMigracaoDirName()`**

Devolve o nome a dar à pasta onde todos os ficheiros criados durante o processo de conversão devem ser colocados. Este nome inclui a data e a hora em que o processo de conversão teve início.

`private String` **`findTemplate`**`(File file)`

Verifica se o programa consegue encontrar o ficheiro com a *template*. O argumento *file* é um dos *forms* a converter, usado para verificar se a *template* se encontra no mesmo directório. Se, nas opções, o utilizador tiver definido um *path* para a *template*,

o método tenta encontrá-la aí. Se nas opções estiver a indicação para procurar a *template* no directório do *form*, ela é procurada aí. Se a *template* for encontrada, o seu *path* completo é devolvido. Se não for encontrada, o método devolve *null*.

`private String` **`getFileParentDir`**`(String fullFormPath)`

Devolve o directório pai do ficheiro passado como argumento. O nome do ficheiro deve conter o *path* completo.

`public void` **`setProgressBarValue`**`(int value)`

Define o novo valor da *ProgressBar*. Esta actualização da interface gráfica é feita recorrendo ao método *invokeLater* da classe *SwingUtilities*, para que a actualização da barra de progresso seja feita na *thread* da interface gráfica, e não na *thread* onde o método é invocado. Caso contrário, a barra de progresso só seria actualizada quando a *thread* de conversão terminasse.

`public void` **`incProgressBarValue`**`()`

Incrementa em uma unidade o valor da ProgressBar.

MainWindow mantém uma relação muito estreita com a classe *MigrationTool*. Muitos dos métodos disponíveis em *MainWindow* não fazem mais do que invocar os métodos com o mesmo nome na classe *MigrationTool*. Isto acontece porque é a instância de *MainWindow* da aplicação que cria todos os objectos que implementam a lógica da aplicação, guardando estes objectos uma referência à *MainWindow* que os criou. São exemplos disto métodos como *getTemplatePath()*, *setTemplatePath()*, *getApplicationPath()*, *setApplicationPath()*, *getClasspath90()*, *getConnectionString()*, *etc.*

Para dar início à conversão, esta classe cria uma nova *thread*, de modo a que a *thread* da interface gráfica não fique bloqueada à espera que a conversão termine. Se tudo fosse feito na mesma *thread*, não seria possível ver a barra de progresso ser actualizada durante o processo, e toda a interface deixaria de responder completamente. Este sistema *multi-threading* é implementado através da criação de um objecto *Runnable*, associado à nova *thread*, cujo método *run()* implementa o ciclo onde os *forms* são processados.

Esta classe implementa um sistema de segurança que impede que um processo de conversão seja iniciado quando outro estiver já em curso. *MainWindow* possui um atributo *isWorking* que indica se um processo de conversão está já em curso ou não. Enquanto este tomar o valor *true*, não é possível abrir novos *forms* para conversão.

O algoritmo do processamento dos *forms*, iniciado quando o utilizador selecciona os ficheiros *fmb* que pretende processar, é ilustrado na figura 20. Este algoritmo está implementado no método *fazConversao()*.

Não estão referidos nesse diagrama os passos onde é feita a actualização da barra de progresso. Estes consistem em invocações ao método *incProgressBarValue()*, feitas depois de cada passo estar concluído. Este método é invocado várias vezes para o mesmo *form*, como por exemplo: depois de ter criado a estrutura de directórios; depois de cada passo do processamento do *form*; depois da compilação em 6i estar concluída; etc.

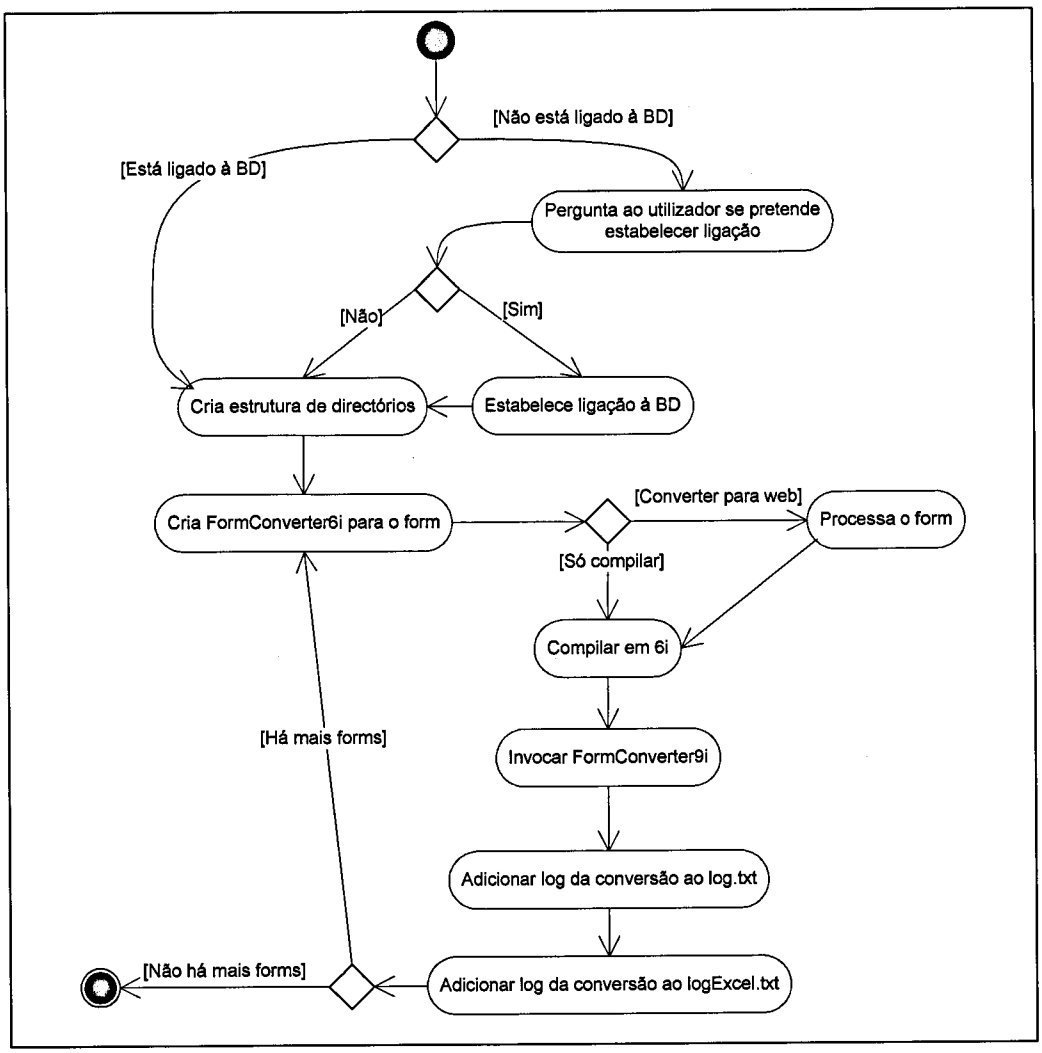


***Fig.20** – Descrição do algoritmo implementado na classe MainWindow, e invocado quando o utilizador inicia um processo de conversão.*

5.2.5 *Launcher*

Descrição: Aplicaçao que dá início à execuçao da *Migration Tool.*

Method Detail

```
public static void main(String[] args)
```

Ponto de entrada da aplicação.

Parameters:

`args` - Não tem argumentos.

Quando o utilizador dá início à *Migration Tool*, está a criar uma instância da classe *Launcher*. A invocação deve ser feita através da execução de um ficheiro *batch* do *Windows*, cujo conteúdo é o seguinte:

java –DPath=%*PATH*% –*classpath* "." Launcher

O que este comando faz é dar início à execução da classe *Launcher*, indicando à *Java Virtual Machine* que deve procurar as classes de que necessita no directório corrente. A opção "-D" é utilizada para passar uma propriedade de nome "*Path*" à máquina virtual, de modo a que esta lhe possa aceder durante a execução. Nesta propriedade é colocado o valor da variável de ambiente *PATH*. A utilização desta propriedade é descrita mais à frente.

Launcher verifica se os ficheiros *f60jdapi.jar* e *f90jdapi.jar* se encontram no directório da aplicação. Assume que o directório *bin* (o directório *bin* com todos ficheiros executáveis e bibliotecas da instalação do *Forms Builder 6i* com suporte para a JDAPI 6i) se encontra também no directório da aplicação.

A invocação de uma nova instância da *Java Virtual Machine* para executar a *Migration Tool* propriamente dita é feita passando o parâmetro *CLASSPATH* na chamada ao comando *java.exe*. Neste parâmetro é colocado o *path* completo do ficheiro *f60jdapi.jar*, para que a JDAPI 6i funcione correctamente. Quando a própria *Migration Tool* invocar o *FormConverter9i*, vai ter que fazer algo semelhante, passando desta feita o *path* completo do ficheiro *f90jdapi.jar*. É apresentada a seguir a lista de passos efectuados nesta classe antes de dar início à execução da *Migration Tool*, acompanhada de uma pequena explicação e detalhes de implementação, sempre que necessário:

- Obter *path* dos ficheiros *f60jdapi.jar* e *f90jdapi.jar*. A seguinte instrução permite obter um objecto *File* que representa o ficheiro *f60jdapi.jar* (é feito o mesmo para o ficheiro *f90jdapi.jar*):

  ```
  File jar60 = new File (Launcher.class.getResource ("f60jdapi.jar").getPath());
  ```

- Obter *path* da aplicação (*appPath*). Este é obtido a partir do *path* completo de um dos ficheiros anteriores.
- Construir a *string classpath60* com o parâmetro *CLASSPATH* a passar aquando da invocação da *Migration Tool*. Este *classpath60* deve conter o *path* da aplicação (para que a *Java Virtua Machine* consiga encontrar o *package* da própria *Migration Tool)*, e o ficheiro *f60jdapi.jar*, para que a *Migration Tool* consiga aceder à JDAPI.
- Construir a *string classpath90* à semelhança da anterior, com o *path* da aplicação e o *path* do ficheiro *f90jdapi.jar*.
- Construir a *string oldPath*, com o conteúdo da variável de sistema *PATH*. Este valor é passado à máquina virtual na invocação de *Launcher* (com "-DPath=%*PATH*%").
- Construir a *string binPath*, com o *path* do directório *bin* contendo os executáveis e bibliotecas necessários para correr a JDAPI 6i.
- Criar um novo processo para correr uma instância da linha de comandos:

  ```
  Process p = Runtime.getRuntime ().exec ("cmd").
  ```

- Obter referência para *input stream* e *output stream* do processo.
- Escrever para a *stream* do processo a *string* seguinte, seguida de mudança de linha:

  "Set *PATH* =" + binPath + "; %*PATH*%"

  Este passo consiste em executar o comando "set *PATH*..." na instância da linha de comandos criada, o que resulta na alteração da variável de ambiente *PATH* que os processos lançados a partir dessa linha de comandos irão "ver".

- Escrever para a *stream* do processo a *string*:

  "java –DPath=" + oldPath + " *–classpath* " + classpath60 + " migracao.*Migration Tool* " + classpath90 + " \"" + appPath + "\""

  Este passo consiste na execução do comando java na instância da linha de comandos criada. O *classpath* passada à *Java Virtual Machine* permite-lhe o acesso à JDAPI e ao *package* da própria *Migration Tool.* Note-se que são passados parâmetros à própria *Migration Tool*, nomeadamente o *classpath90* e o *appPath.* O *classpath90* vai ser usado para invocar o *FormsConverter9i*, e o *appPath* para a *Migration Tool* saber onde deve ler e criar os ficheiros de configuração da aplicação.

  A opção "-D" é aqui utilizada mais uma vez para passar o conteúdo original da variável de ambiente *PATH* sob a forma de uma propriedade, para que a aplicação lhe possa aceder em execução. Note-se que a variável de ambiente *PATH* que a aplicação "vê" é já a versão alterada (a que é definida com o comando "Set *PATH* =" + binPath + "; %*PATH*%"). Este *Path* passado como propriedade com a opção "-D" é visto pela *Java Virtual Machine* como um valor que as aplicações Java podem consultar, tal como se de um parâmetro da linha de comandos se tratasse, por exemplo.

  Assim, a *Migration Tool* é executada com o *PATH* alterado, a apontar para a pasta *bin* com suporte para a JDAPI 6i (este *PATH* alterado e o *CLASSPATH* a apontar para o ficheiro *f60jdapi.jar* são os requisitos para o correcto funcionamento da JDAPI 6i); mas tem acesso ao *PATH* original através de uma propriedade da *Java Virtual Machine*, para quando precisar dele.

5.2.6 MigrationTool

Descrição: Constitui a classe principal da *Migration Tool.*

Constructor Detail

`public MigrationTool()`

Construtor para esta classe. Constrói e apresenta a interface gráfica.

Method Detail

`public static void main(String[] args)`

Ponto de entrada na aplicação.

Parameters:

`args` - O 1° argumento é o *classpath* a usar aquando da invocação do *FormConverter9i*; O 2° argumento é o *path* da aplicação (o caminho onde está a ser executada).

`public void checkPrefsFile()`

Verifica se o ficheiro *Options.pref* existe no directório da aplicação. Se existir, carrega as preferências nele contidas. Se não existir, cria um novo com valores *default*, carregando-o também de seguida.

`public int loadOptionsPrefFile()`

Lê os dados do ficheiro *Options.pref*, e carrega-os. Devolve 0 se a operação for efectuada com sucesso, -1 caso contrário.

`public void` **`createOptionsPrefFileDefault`**`()`

Deve ser invocado para criar o ficheiro de preferências da aplicação, caso este ainda não exista. Se o ficheiro já existir, é substituído por um com os valores *default*.

`public void` **`createOptionsPrefFile`**`()`

Cria o ficheiro de preferências com os valores encontrados nos atributos desta classe.

`public String` **`getOptionsPrefFilePath`**`()`

Devolve o *path* completo para o ficheiro de preferências que guarda as definições actuais do painel de opções.

`public boolean` **`isWorking`**`()`

Devolve *true* se algum processo de conversão estiver em curso (só pode estar um em curso de cada vez).

`public void` **`setIsWorking`**`(boolean val)`

Define o valor da propriedade *isWorking*.

`public String` **`getOpenDirPath`**`()`

Devolve o *path* do directório onde o *FileChooser* deve iniciar.

`public void` **`setOpenDirPath`**`(String` *`path`*`)`

Define o *path* do directório onde o *FileChooser* deve iniciar.

`public String` **`getTemplatePath`**`()`

Devolve o *path* da *template*.

`public void` **`setTemplatePath`**`(String` *`path`*`)`

Define o *path* da *template*.

`public String` **`getApplicationPath`**`()`

Devolve o *path* do directório onde a *Migration Tool* se encontra instalada.

`public void` **`setApplicationPath`**`(String` *`path`*`)`

Define o *path* onde a *Migration Tool* se encontra instalada .

`public String` **`getClasspath90`**`()`

Devolve o *CLASSPATH* a usar na invocação do *FormConverter9i*.

`public void setClasspath90(String path)`

Define o *CLASSPATH* a usar na invocação do *FormConverter9i.*

`public String getUser()`

Devolve o nome do utilizador a usar para o acesso à BD.

`public String getPassword()`

Devolve a *password* a usar para o acesso à BD.

`public String getDatabase()`

Devolve o nome da Base de Dados à qual se deve ligar.

`void setUser(String newString)`

Define o novo utilizador.

`void setPassword(String newString)`

Define a nova *password.*

`void setDatabase(String newString)`

Define a nova BD.

`public boolean isConnected()`

Devolve *true* se houver uma ligação activa à BD, *false* caso contrário.

```
public int connect(String user,
                   String pass,
                   String db)
```

Estabelece uma ligação à BD, usando os parâmetros indicados. Se já houver uma ligação activa, é devolvido -1.

`public int connect(String connString)`

Liga-se à BD usando a *connection string* indicada.

`public void disconnect()`

Se houver uma ligação activa à BD, termina-a.

`public int toggleConnection()`

Alterna entre o estado ligado e desligado. Devolve *CONNECTED* ou *DISCONNECTED*, para indicar qual é o novo estado.

Esta classe constitui o ponto de entrada na *Migration Tool*. É uma instância desta classe que *Launcher* cria para iniciar a aplicação. É esta classe que apresenta a interface gráfica ao utilizador, através da instanciação de *MainWindow*.

É nesta classe que são guardadas as principais variáveis da aplicação, tal como o estado da ligação à BD, os parâmetros da ligação, a indicação de que uma conversão está em curso ou não, todos os *paths* usados (*classpath90*, *appPath*, *templatePath*), e todas as opções guardadas no ficheiro de preferências.

5.2.7 *FormConverter6i*

Descrição: Esta classe disponibiliza os métodos necessários para fazer as correcções e verificações num *form*.

Constructor Detail

```
public FormConverter6i(String formName,
                       MainWindow owner,
                       String templatePath)
```

Construtor para esta classe.

Parameters:

`formName` - *Path* completo do *form* a abrir.

`owner` - Instância de *MainWindow* que criou este objecto.

`templatePath` - *Path* completo do ficheiro *temp.fmb*.

Method Detail

```
public void parseForm()
```

Processa o *form* a que se refere este *FormConverter6i*. Este método deve ser invocado para dar início ao processo.

```
public void openIn9i(boolean shouldConnect)
```

Invoca um programa externo (*FormConverter9i*), que abre, corrige e compila o *form* com JDAPI 9i.

Parameters:

`shouldConnect` - indica se deve ser estabelecida a ligaçao à BD.

```
private void fixTemplate()
```

Remove o *ObjectGroup* da *template*, e adiciona-o de novo, de modo a corrigir possíveis falhas ao herdar da *template*. Corrige a ordenação dos *DataBlocks* e *Canvases*, e define a propriedade *HorizontalToolBarCanvas* do *form*.

```
private void fixLibraries()
```

Remove as bibliotecas antigas, substituindo-as pelas novas.

```
private void addLib(String libName)
```

Adiciona a biblioteca indicada ao *form*, garantindo que as suas dependencias (outras bibliotecas que esta utiliza) sao também adicionadas.

`private int` **`removeLibrary`**`(String libName)`

Remove a biblioteca com o nome indicado.

Returns:

0 se a biblioteca for removida, ou -1 se esta não existir no *form*.

`private boolean` **`isOldLibrary`**`(AttachedLibrary lib)`

Indica se a biblioteca indicada consta da lista de bibliotecas a serem removidas.

`public Block` **`getBlock`**`(String blockName)`

Devolve o bloco com o nome indicado.

`public Canvas` **`getCanvas`**`(String canvasName)`

Devolve o *Canvas* com o nome indicado.

`public ObjectGroup` **`getObjectGroup`**`(String groupName)`

Devolve o *ObjectGroup* com o nome indicado.

`public` *`ProgramUnit`* **`getProgramUnit`**`(String unitName)`

Devolve a *ProgramUnit* com o nome indicado.

`private void` **`parseProgramUnits`**`()`

Processa cada uma das *ProgramUnits* do *form*, efectuando as verificações e correcçoes necessárias.

`private void` **`parseTriggers`**`()`

Processa todos os *triggers* do *form*: *triggers* ao nível do *form*; ao nível do bloco; e ao nível do *item*.

`private void` **`parseTriggers`**`(JdapiIterator` *`triggers`*`)`

Processa cada um dos *triggers* devolvidos pelo iterador, efectuando as verificaçoes e correcçoes necessárias.

`private void` **`checkMouseTriggers`**`(Trigger trigger)`

Verifica a existência dos *triggers WHEN-MOUSE-ENTER* e *WHEN-MOUSE-LEAVE* no *form*, registando a sua ocorrência no *FormLog*.

`private void` **`checkRun_Product`**`(String triggerText)`

Procura no texto passado como parâmetro a *string* "run_product", e regista no *FormLog* a sua ocorrência.

`private void` **`checkCreate_Timer`**`(String triggerText)`

Procura no texto passado como parâmetro a *string* "create_timer", e regista a sua ocorrência no *FormLog*.

`private void` **`checkActiveX`**`(String triggerText)`

Procura no texto passado como parâmetro a *string* "dispatch_event" (que denuncia a utilização de um controlo *ActiveX*), e regista a sua ocorrência no *FormLog*.

`private String` **`fixGet_File_Name`**`(String triggerText)`

Procura no texto passado como parâmetro a *string* "get_file_name", e regista no *FormLog* a sua ocorrência. Corrige todas as ocorrências para "cpc_get_file_name", devolvendo o texto corrigido.

`private String` **`fixTool_Env`**`(String triggerText)`

Procura no texto passado como parâmetro a *string* "tool_env", e regista no *FormLog* a sua ocorrência. Corrige todas as ocorrências para "cpc_tool_env", devolvendo o texto corrigido.

`private String` **`fixText_IO`**`(String triggerText)`

Procura no texto passado como parâmetro a string "text_io", e regista no *FormLog* a sua ocorrência. Corrige todas as ocorrências para "cpc_text_io", devolvendo o texto corrigido.

`private String` **`fixWin_API`**`(String triggerText)`

Procura no texto passado como parâmetro a *string* "win_api", e regista no *FormLog* a sua ocorrência. Corrige todas as ocorrências de "win_api_environment" para "cpc_win_api_environment", devolvendo o texto corrigido. As restantes ocorrências são apenas registadas.

`private String` **`fixHosts`**`(String triggerText)`

Procura no texto passado como parâmetro a *string* "*host*". Corrige as ocorrências de "*host*" para "reports_pck.abre_report" quando se tratarem de chamadas a *reports*, ou para "cpc_hosts.cpc_host" nos restantes casos. Utiliza a classe *Host*.

`private boolean` **`isNameValidChar`**`(char ch)`

Indica se é possível usar o *char* indicado para constituir nomes de funções e métodos em PL/SQL.

`public String` **`buildLogPath`**`()`

Devolve o *path* completo para o ficheiro de *log* a ser criado.

`public String` **`buildLogPathExcel`**`()`

Devolve o *path* completo para o ficheiro de *log Excel* a ser criado.

`public void` **`createLog`**`()`

Cria o ficheiro *log.txt* na pasta "migracao_".

`public void` **`createLog`**`(String fileName)`

Cria o ficheiro *log.txt* com o nome indicado.

Parameters:

`fileName` - *Path* completo do ficheiro de *log* (*log.txt*) a ser criado.

`public void` **`createLogExcel`**`()`

Cria o ficheiro *logExcel.txt* na pasta "migracao_".

`public void` **`createLogExcel`**`(String fileName)`

Cria o ficheiro de *log* do Excel com o nome indicado.

Parameters:

`fileName` - *Path* completo do ficheiro de *log* Excel (*logExcel.txt*) a ser criado.

`public void` **`saveForm`**`()`

Guarda as alterações efectuadas ao *form*.

`public void` **`saveForm`**`(String newName)`

Guarda as alterações efectuadas ao *form*, criando um novo ficheiro com o nome indicado.

`public void` **`createDirectoryTree`**`(String migracaoDir)`

Cria, na pasta onde o *form* se encontra, um directório "migracao_". Dentro deste, cria os directorios "sources6i", "execs6i" e "execs9i".

Parameters:

`migracaoDir` – *path* do directorio raiz a criar ("migracao_").

`public void` **`setMigracaoDir`**`(String dir)`

Define o nome do directório a usar para colocar todos os ficheiros resultantes do processo de conversão.

`public int` **`compileForm`**`(boolean shouldConnect)`

Compila o *form* em 6i através da invocaçao do compilador de linha de comandos. O parâmetro *shouldConnect* indica se o compilador externo deve efectuar o *login* na BD antes de compilar.

Returns:

0 se não ocorreram erros de compilaçao; -1 se ocorreram erros.

```
private void moveFmxFile()
```

Move o ficheiro *fmx* resultante da compilação para a pasta "execs6i".

```
private void moveErrFile()
```

Move o ficheiro *err* gerado pelo compilador para a pasta "_erros". Isto só deve ser feito quando ocorrem erros de compilação, caso contrário o ficheiro *err* deve ser apagado.

```
private void deleteErrFile()
```

Apaga o ficheiro *err* gerado pelo compilador de linha de comandos. Só deve ser usado depois de se verificar que a compilação ocorreu sem erros, caso contrário este ficheiro deve ser guardado.

```
public void moveFmbFileWithError()
```

Guarda o ficheiro *fmb* já com alterações na pasta "_erros", caso este exista. Deve ser invocado quando ocorrem erros de compilação em 6i.

```
public String replaceAllCaseInsensitive(String str,
                                        String regex,
                                        String replacement)
```

Faz o mesmo que o método *replaceAll()* da classe *String* do Java, mas fazendo a pesquisa em modo "case insensitive".

```
public String getFormPath()
```

Devolve o *path* do *form* associado a este *FormConverter6i*. Devole uma *string* igual a *this.getFormDir()* + "/" + *this.getFormName()*.

```
public String getFormName()
```

Devolve o nome do *form*, com a extensão *fmb* (não inclui o *path* completo).

```
public String getFormDir()
```

Devolve o *path* do directório onde se encontra o *form*.

```
public String insertString(String token,
                           int index,
                           String text)
```

Insere o texto *token* no meio da *string text*, na posição indicada em *índex*.

Returns:

Devolve o texto alterado.

`public String` **`fixCompileErrors`**`()`

Caso tenham ocorrido erros de compilação em 6i, verifica se estes ocorreram devido ao facto de faltar alguma bilioteca. Em caso afirmativo, adiciona as bibliotecas em falta.

Para cada *form* a ser processado, é criada uma instância de *FormConverter6i*. Esta disponibiliza métodos como *parseForm()*, *saveForm()*, *compileForm()*, *openIn9i()*, *createLog()*, *createLogExcel()*, que, entre outros, constituem a interface da classe com o exterior. A instância de *MainWindow* que cria as várias instâncias de *FormConverter6i* invoca estes métodos para controlar o processamento dos *forms*.

O método *parseForm()* efectua todas as verificações e alterações ao *form*. Os métodos *saveForm()* e *compileForm()* guardam as alterações e compilam o *form* na plataforma 6i, respectivamente. O método *openIn9i()* invoca a execução de *FormConverter9i* para efectuar o processamento necessário à compilação em 9i.

Cada instância de *FormConverter6i* cria uma instância de *FormLog*, na qual vai registando todo o processamento efectuado.

O algoritmo principal implementado nesta classe é aquele onde é efectuado o processamento dos *forms*. Este é constituído pelos seguintes passos:

1. Criar *FormLog*
2. Corrigir bibliotecas
3. Recarregar a *template* (se essa opção tiver sido seleccionada)
4. Processar código das *ProgramUnits*
5. Processar código dos *Triggers*

Os passos 4 e 5, que involvem o processamento do código PL/SQL, são os mais complexos. Estes consistem em percorrer todos os objectos relevantes (*ProgramUnits* ou *Triggers*) e efectuar, para cada um, as verificações e alterações ao código.

No caso das *ProgramUnits*, como estas se encontram todas ao nível do *form*, obtém-se directamente um iterador sobre todas elas. A figura 21 apresenta um diagrama de actividade que descreve o funcionamento do algoritmo que efectua as alterações às *ProgramUnits* (implementado no método *parseProgramUnits()*). Como esse diagrama torna evidente, apenas as *ProgramUnits* cujo código PL/SQL não se encontre no estado "herdado" são processadas (o código PL/SQL é uma das propriedades da *ProgramUnit* e pode, portanto, ser herdado de uma *PropertyClass*, por exemplo). Isto acontece porque, se o código está a ser herdado, deve ser alterado na fonte. Se fosse efectuada a alteração directamente na *ProgramUnit*, a propriedade correspondente ao código passaria ao estado *Overriden Inherited Value*, e perderia assim a ligação com a propriedade de onde estava a obter o seu valor.

No passo seguinte do algoritmo, verifica-se o tipo da *ProgramUnit*. Se for do tipo *PackageSpec* esta é também ignorada, já que numa *ProgramUnit* deste tipo não existe código PL/SQL que possa ser alvo das verificações e alterações a efectuar (estas contêm apenas assinaturas de métodos e definições de variáveis). São processadas apenas as do tipo *PackageBody, Procedure* e *Function.*

Só depois de passar todas estas validações é que uma *ProgramUnit* chega de facto à fase de processamento. Esta começa com as verificações necessárias, passando depois às alterações. Uma série de métodos cujo nome começa com "*check*" encarrega-se de fazer as verificações e respectivo registo no *log,* enquanto que as funções invocadas de seguida, com nomes começados por "*fix*", verificam a existência das ocorrências de que estão encarregues e substituem-nas pelas novas versões.

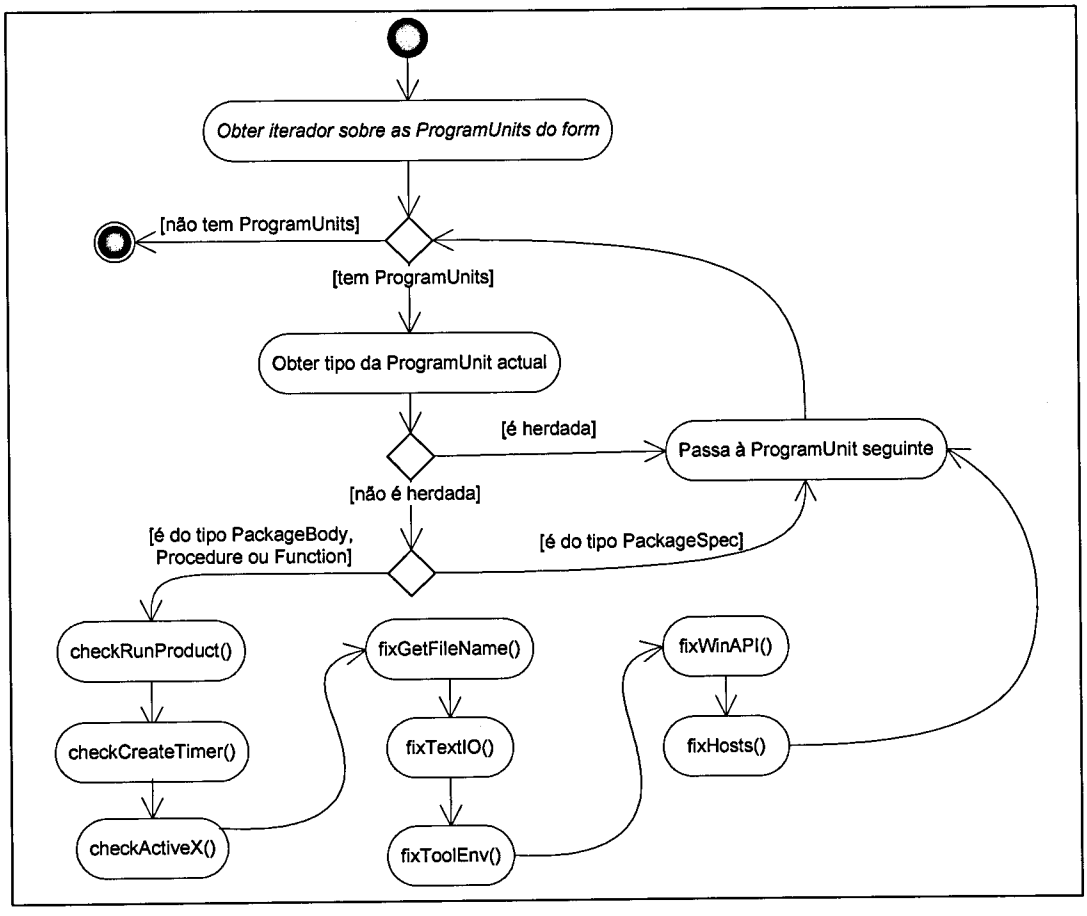


Fig.21 *– Descrição do algoritmo que efectua o processamento das ProgramUnits, implementado no método parseProgramUnits() da classe FormConverter6i.*

O processamento dos *triggers* é um pouco mais complexo, pois estes não se encontram todos directamente ao nível do *form* e, portanto, acessíveis através de um mesmo iterador. O processamento dos *triggers* é feito então em dois passos:

1. Percorre-se todos os objectos do *form* que possam ter *triggers,* obtendo-se iteradores sobre os *triggers* de cada objecto;
2. Percorre-se todos os *triggers* de cada iterador, efectuando-se um processamento semelhante àquele descrito para as *ProgramUnits.*

A fig.22 ilustra o algoritmo implementado no método *parseTriggers(void)*, onde é realizado o passo com o número 1. O método *parseTriggers(JdapiIterator iterador)* é invocado várias vezes na execução deste algoritmo, para processar os vários *triggers* do iterador que lhe é passado como parâmetro. Neste segundo método é implementado o passo 2.

O método *parseTriggers(JdapiIterator iterador)* implementa um algoritmo muito semelhante ao de *parseProgramUnits()*. De facto, a única diferença é que no processamento das *ProgramUnits* se efectua uma filtragem por tipo de *ProgramUnit*, enquanto que esta verificação não existe no caso dos *Triggers*. A noção de "tipo" não existe nos *Triggers*. A outra verificação efectuada nas *ProgramUnits* (referente à herança do código PL/SQL), contudo, é feita também nos *Triggers*.

Na fig.22, os pontos onde o método *parseTriggers(JdapiIterator)* é invocado correspondem às actividades destacadas a cor azul.

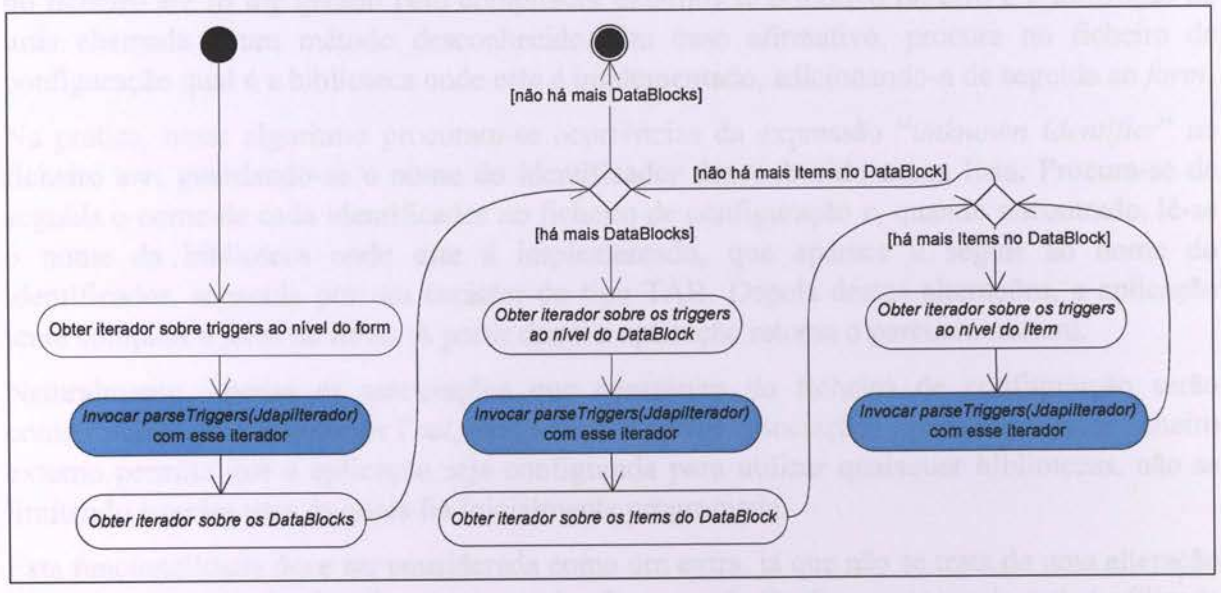


***Fig.22** – Descrição parcial do algoritmo que efectua o processamento dos triggers. Este é o algoritmo implementado no método parseTriggers(void) da classe FormConverter6i. Este diagrama descreve os passos levados a cabo até ao ponto em que se obtém os vários iteradores sobre os Triggers. O processamento sobre esses iteradores é muito semelhante ao descrito na figura 21.*

Os procedimentos *check* e fix** são invocados da mesma forma para o processamento das *ProgramUnits* e dos *Triggers*. Recebem o texto que constitui o código PL/SQL do *Trigger* ou *ProgramUnit* como parâmetro, e trabalham sobre ele. Os métodos *check* limitam-se a procurar ocorrências das palavras-chave correspondentes, registando-as no objecto *FormLog* associado ao *FormConverter6i*. Os métodos *fix** procuram ocorrências das respectivas palavras-chave, e substituem-nas pelas novas palavras (efectuando o registo junto do *FormLog*). O texto alterado é devolvido como valor de retorno destas funções. O método *fixHosts()*, contudo, invoca um algoritmo bastante mais complexo que o descrito, pois as alterações que este tem que fazer não se limitam apenas à substituição diecta de uma palavra-chave por outra. A secção 5.2.9 deste capítulo, que descreve o componente *Host*, aborda este assunto com mais detalhe.

O método *fixCompileErrors()* desta classe implementa uma funcionalidade que não foi contemplada na análise de requisitos da aplicação. De facto, aquando do início da especificação da *Migration Tool*, não se previa que esta viria a ser necessária. Durante o desenvolvimento da aplicação foram efectuados vários testes, tendo sido convertidos e compilados vários *forms* desenvolvidos na CPCHS. Uma percentagem bastante elevada destes *forms* apresentava uma característica bizarra: não era possível compilá-los em 6i, sendo que o motivo dos erros de compilação era a falta de bibliotecas cujas funções e procedimentos eram utilizadas pelos mesmos. O motivo pelo qual estes *forms* perderam as referências a determinadas bibliotecas de que necessitavam permanece desconhecido, até porque o assunto não foi muito aprofundado. Contudo, implementou-se na *Migration Tool* uma funcionalidade que permite corrigir este problema. Criou-se, num ficheiro de configuração[18], uma listagem das bibliotecas existentes e respectivos procedimentos e funções. O algoritmo implementado no método *fixCompileErrors()* consiste em, caso ocorra algum erro de compilação, verificar no ficheiro *err* (o *log* gerado pelo compilador externo) se o motivo do erro é a utilização de uma chamada a um método desconhecido. Em caso afirmativo, procura no ficheiro de configuração qual é a biblioteca onde este é implementado, adicionando-a de seguida ao *form*.

Na prática, neste algoritmo procuram-se ocorrências da expressão "*unknown identifier*" no ficheiro *err*, guardando-se o nome do identificador desconhecido numa lista. Procura-se de seguida o nome de cada identificador no ficheiro de configuração e, quando encontrado, lê-se o nome da biblioteca onde este é implementado, que aparece a seguir ao nome do identificador, separada por um carácter do tipo TAB. Depois destas alterações, a aplicação tenta compilar o *form* de novo. A partir daqui a aplicação retoma o percurso normal.

Naturalmente, apenas as associações que constarem do ficheiro de configuração serão contempladas pela *Migration Tool*, mas o facto de estas associaçãos serem feitas num ficheiro externo permite que a aplicação seja configurada para utilizar quaisquer bibliotecas, não se limitando àquelas para as quais foi inicialmente programada.

Esta funcionalidade deve ser considerada como um extra, já que não se trata de uma alteração necessária à migração dos *forms* para a plataforma *web*. Os *forms* nos quais esta é utilizada apresentavam já erros na sua versão original, na plataforma 6i. Trata-se, portanto, de uma correcção, e não de um passo necessário à conversão. Contudo, a sua implementação aumentou significativamente a percentagem de *forms* convertidos com sucesso pela *Migration Tool*, conforme se pode verificar pelos resultados apresentados no capítulo 6 (nestes cálculos ignora-se o facto de os *forms* poderem já ter erros na sua versão original).

5.2.8 FormConverter9i

Descrição: Classe responsável pela compilação dos *forms* para a plataforma 9i.

Constructor Detail

```
public FormConverter9i(String formName,
                       String migracaoDir,
                       String connString)
```

Construtor para esta classe.

Parameters:

[18] O ficheiro de configuração utilizado é o ficheiro *libraries.pref*, que era já usado para armazenar o nome das bibliotecas que deveriam ser removidas, e quais as suas substitutas.

`formName` - *path* completo do ficheiro *fmb* do *form* a compilar.

`migracaoDir` - Directório criado para o projecto de migração, onde os produtos da compilaçao devem ser colocados.

`connString` - ConnectionString a usar para a ligaçao à BD. Se for null, o *form* é compilado sem ser estabelecida a ligação.

Method Detail

`private void` **`dragDataBlocks()`**

"Arrasta" os Datablocks "CPCHSBEANS" e "WEBUTIL", colocados no início ao abrir o *form* com a *template* 9i, para o fim da lista de blocos.

`private int` **`compileForm()`**

Compila o *form* para a plataforma 9i.

Returns:

0 em caso de sucesso; -1 se ocorreu algum erro.

`private void` **`moveFmxFile()`**

Move o ficheiro *fmx* resultante da compilação para a pasta "execs9i".

`private void` **`moveErrFile()`**

Move o ficheiro *err* gerado aquando da compilaçao para a pasta "_erros". Este método só deve ser invocado quando ocorrem erros na compilaçao, caso contrário o ficheiro *err* deve ser eliminado.

`private void` **`deleteErrFile()`**

Apaga o ficheiro *err* gerado aquando da compilaçao. Este método só deve ser invocado depois de se verificar que não ocorreu qualquer erro na compilaçao, caso contrário o ficheiro deve ser guardado, pois conterá informaçao útil acerca dos erros encontrados.

```
public String replaceAllCaseInsensitive(String str,
                                         String regex,
                                         String replacement)
```

Faz o mesmo que o método *replaceAll* da classe *String* do Java, mas fazendo a pesquisa em modo *case insensitive*.

`public String` **`getFormName()`**

Devolve o nome do *form*, incluindo a extensão *fmb* (não inclui o *path* completo).

`private Block` **`getBlock`**`(String blockName)`

Devolve o *DataBlock* com o nome indicado, ou *null* se não existir um bloco com esse nome.

`public static void main(String[] args)`

Ponto de entrada na aplicação. A sintaxe para invocação é a seguinte:

FormConverter9i "formPath" "migracaoDir" <"connectionString">.

Parameters:

`args` – 1) "formPath": *path* completo do *fmb* a compilar; 2) "migracaoDir": directório criado para o projecto de conversão, onde onde se encontram os directórios "execs9i", "execs6i", etc.; 3) "connectionString" - *Connection String* a usar na ligação à BD. Este último parâmetro é opcional.

Esta classe é responsável por efectuar a compilação do *form* para a plataforma 9i. Teoricamente, ela não faz alterações ao *form*. Na prática, contudo, o *form* tem que ser alterado para que o ficheiro executável criado funcione correctamente na plataforma *web*. As alterações efectuadas, no entanto, para além de serem de pequena dimensão, são descartadas depois da compilação. Isto acontece não só porque são alterações que não fariam sentido na versão 6i do *form*, mas também porque se o ficheiro fosse guardado com a versão da JDAPI aqui utilizada (JDAPI 9i), deixaria de ser compatível com o *Forms Builder6i* e a JDAPI 6i.

A alteração que tem de ser efectuada está relacionada com a *template* utilizada na plataforma *web*. Esta *template* (que é um ficheiro *temp.fmb* diferente daquele que é usado na plataforma cliente-servidor) é constituída por alguns objectos que não existem na versão para 6i. Acontece que, quando o *form* é aberto com a JDAPI 9i, essa *template* é utilizada, em substituição da que tinha sido usada para o processamento em 6i. Os novos objectos são adicionados ao *form* no momento em que este é aberto com esta API (acontece o mesmo quando se abre o *form* no *Forms Builder 9i*), e são sempre colocados no início das listas a que pertencem (os *DataBlocks* adicionais são colocados no início da lista de *DataBlocks*, os *Canvases* no início da lista de *Canvases,* etc.)

Dos componentes adicionados pela nova *template*, os que trazem problemas são os dois *DataBlocks* acrescentados, nomeadamente "CPCHSBEANS" e "WEBUTIL". Estes novos *DataBlocks* existem nesta *template* porque vêm disponibilizar substitutos para algumas das funcionalidades suportadas em 6i que deixaram de ser compatíveis com a plataforma *web*. O *DataBlock* CPCHSBEANS contém alguns *JavaBeans* que substituem controlos *OCX* e *ActiveX* usados em 6i. O bloco *WEBUTIL*, desenvolvido pela própria Oracle, permite que sejam invocados programas locais a partir de um *form* que esteja a correr num servidor *web* (que é exactamente o que acontece aquando da execução das aplicações da CPCHS na plataforma 9i).

O problema não reside na existência dos novos blocos, mas sim na posição que assumem por defeito (à frente de todos os outros blocos do *form*). Quando um *form* é aberto pelo *runtime*, o foco da interface gráfica é, por defeito, atribuído ao primeiro *item* do primeiro *DataBlock*. É possível que o *form* defina em código qual é o *item* ao qual o foco deve ser atribuído. Pode também indicar qual é o bloco cujo primeiro *item* irá obter o foco. Mas nem sempre isto acontece. Segundo os testes efectuados, a grande maioria dos *forms* da CPCHS não define qual é o *item* (ou bloco cujos *items)* que deve ter o foco inicialmente. Nestes casos, o posicionamento dos dois novos blocos antes de todos os outros provoca o aparecimento de um *canvas* que não deveria ficar visível, cujos *items* se encontram num dos novos blocos. O

surgimento destes *items* é indesejado, sendo resolvido com o posicionamento dos dois blocos em último lugar (quando são necessários, estes são explicitamente invocados).

Assim, o que esta classe faz é unicamente mover os dois novos blocos para último lugar (através do método *move(int)* disponibilizado pela classe *DataBlock* da JDAPI), compilando de seguida o *form* (recorrendo ao método *compile()* disponibilizado pela classe *FormModule* da JDAPI). *FormConverter9i* é ainda responsável por verificar se ocorreu algum erro de compilação. Em caso afirmativo, deve guardar o ficheiro *err* criado pelo compilador na pasta "_erros". Caso não ocorram erros, coloca o ficheiro executável (*fmx*) no directório "execs9i", e apaga o ficheiro *err*.

Esta classe necessita de comunicar ao *FormConverter6i* que o invocou se ocorreu algum erro durante a compilação em 9i. Esta informação é utilizada depois por essa classe para actualizar o *log file*, e ainda para decidir o que fazer com o ficheiro *fmb*. Isto é feito através da linha de comandos. Quando *FormConverter6i* cria um Processo "cmd" para lançar a máquina virtual Java onde vai correr o *FormConverter9i*, obtém referências para as *streams* de *input* e *output* do Processo (que é a instância da linha de comandos). Então, o *FormConverter9i* escreve para a sua *stream* de saída padrão a mensagem "Sem erros no 9i" quando não ocorrem erros, e "Com erros no 9i" quando ocorrem. O *FormConverter6i,* depois de ordenar a execução do *FormConverter9i*, só tem que ficar à "escuta" na *stream* de saída do processo que criou para interceptar esta mensagem.

5.2.9 Host

Descrição: Auxilia o processamento dos *forms*. Substitui as invocações à built-in *Host* do PL/SQL pelas novas versões compatíveis com a plataforma web.

Constructor Detail

```
public Host()
```

Construtor para esta classe.

Method Detail

```
public String fixHosts(String triggerText)
```

Procura no texto passado como parâmetro a string "*host*". Corrige as ocorrências de "*host*" para "reports_pck.abre_report" quando se tratarem de chamadas a *reports*, ou para "cpc_hosts.cpc_host" nos restantes casos. Devolve o texto já corrigido.

Uma das tarefas da *Migration Tool* é substituir todas as invocações ao procedimento *built-in* do PL/SQL *host.* Este procedimento é utilizado para invocar programas externos, através da execução de um comando do sistema. Nos *forms* da CPCHS é utilizada a seguinte instrução, por exemplo, para abrir a calculadora do *Windows*: *host('calc');* a tarefa da *Migration Tool*, neste caso, é substituir essa instrução por: *cpc_hosts.cpc_host('calc');*

Outra utilização típica da *built-in host* é a abertura de *reports*. Como a plataforma *Forms 6i* disponibiliza um programa de linha de comandos para abrir *reports (rwrun60* ou *r25run32)*, é possível utilizar uma chamada a *host* para os abrir. Contudo, para o correcto funcionamento em *web*, os *reports* não podem ser invocados com o procedimento *cpc_hosts.cpc_host()*. Existe outro procedimento, *reports_pck.abre_report()*, que deve ser usado nestes casos.

Esta classe é então responsável por identificar quando é que as invocações a *host* se tratam de chamadas a *reports*, para saber qual a função a utilizar. Mais ainda, enquanto que o procedimento *host* recebe um único parâmetro (à semelhança de *cpc_hosts.cpc_host()*), *reports_pck.abre_report()* recebe 3 ou 4 parâmetros (o último é opcional). Para além de ter que identificar se se trata de uma chamada a um *report* ou não, esta classe tem que efectuar o processamento da *string* utilizada como parâmetro no procedimento *host*, na tentativa de encontrar os vários parâmetros de que necessita.

A assinatura do procedimento que deve ser invocado no caso dos *reports* é a seguinte:

reports_pck.abre_report(report string, params string, connStr string, command string);

Todos estes parâmetros são também passados na invocação do *report* com a *built-in host()*. Contudo, aí são passados sob a forma de uma única *string* que consiste numa chamada a um comando do sistema (com o nome do comando seguido dos vários parâmetros separados por espaço).

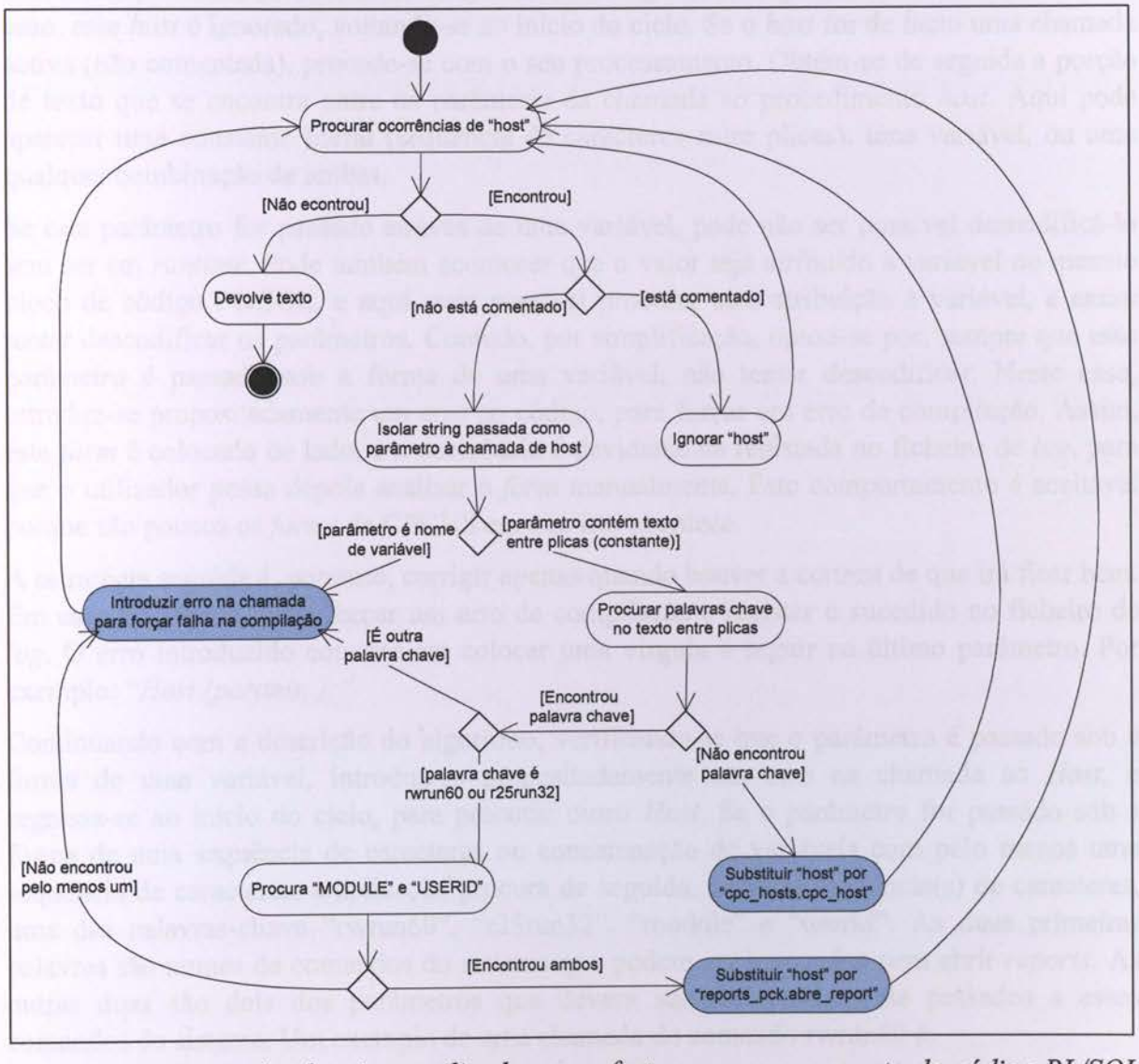


***Fig.23** – Descrição do algoritmo utilizado para efectuar o processamento do código PL/SQL relativamente às chamadas à built-in host(). As actividades a azul são aquelas em que a aplicação toma uma decisão relativamente ao Host que está a processar.*

O problema deste processamento é que a *string* passada ao procedimento *host* é normalmente construída em porções distintas do código, resultando da concatenação de variáveis com sequências de caracteres, dificultando a realização do *parsing* dos parâmetros. O protótipo desenvolvido não tem este problema totalmente resolvido. Mas permite já uma percentagem de conversões com sucesso bastante elevada, e dá a garantia de que, se não conseguir efectuar correctamente a alteração, o *form* irá falhar na compilação, de modo a que o utilizador saiba exactamente que esse *form* exige a sua atenção.

A figura 23 contém um diagrama que esquematiza o algoritmo utilizado para efectuar este processamento dos *hosts*. Este algoritmo, implementado na classe *host*, é executado através da invocação da função *fixHosts(String triggerText)*.

Para implementar este algoritmo, recorreu-se às funcionalidades disponibilizadas pelo *package java.util.regex*, que permite a utilização de expressões regulares em Java. No ciclo principal deste algoritmo é efectuada uma pesquisa no texto pelo termo "*host*". O algoritmo termina quando não houver mais nenhum *host* para processar. Sempre que um *host* é encontrado, verifica-se se este se encontra em código que esteja comentado. Se for esse o caso, esse *host* é ignorado, voltando-se ao início do ciclo. Se o *host* for de facto uma chamada activa (não comentada), procede-se com o seu processamento. Obtém-se de seguida a porção de texto que se encontra entre os parêntesis da chamada ao procedimento *host*. Aqui pode aparecer uma constante literal (sequência de caracteres entre plicas), uma variável, ou uma qualquer combinação de ambas.

Se este parâmetro for passado através de uma variável, pode não ser possível descodificá-lo sem ser em *runtime*. Pode também acontecer que o valor seja atribuído à variável no mesmo bloco de código PL/SQL, e aqui seria possível procurar essa atribuição à variável, e assim tentar descodificar os parâmetros. Contudo, por simplificação, optou-se por, sempre que esse parâmetro é passado sob a forma de uma variável, não tentar descodificar. Neste caso, introduz-se propositadamente um erro no código, para forçar um erro de compilação. Assim, este *form* é colocado de lado, e a ocorrência é devidamente registada no ficheiro de *log*, para que o utilizador possa depois analisar o *form* manualmente. Este comportamento é aceitável porque são poucos os *forms* da CPCHS em que isto acontece.

A estratégia seguida é, portanto, corrigir apenas quando houver a certeza de que irá ficar bem. Em caso de dúvida, deve forçar um erro de compilação e registar o sucedido no ficheiro de *log*. O erro introduzido consiste em colocar uma vírgula a seguir ao último parâmetro. Por exemplo: "*Host (params,);*"

Continuando com a descrição do algoritmo, verificando-se que o parâmetro é passado sob a forma de uma variável, introduz-se propositadamente um erro na chamada ao *Host,* e regressa-se ao início do ciclo, para procurar outro *Host*. Se o parâmetro for passado sob a forma de uma sequência de caracteres ou concatenação de variáveis com pelo menos uma sequência de caracteres, a aplicação procura de seguida, nessa(s) sequência(s) de caracteres, uma das palavras-chave "rwrun60", "r25run32", "module" e "userid". As duas primeiras palavras são nomes de comandos do sistema que podem ser invocados para abrir *reports*. As outras duas são dois dos parâmetros que devem ser obrigatoriamente passados a esses comandos do sistema. Um exemplo de uma chamada do comando rwrun60 é:

rwrun60 MODULE=diario.rep USERID=user/pass@db PARAMFORM=NO

Qualquer chamada a rwrun60 ou r25run32 deve conter sempre os parâmetros MODULE e USERID, sendo os restantes opcionais. As palavras-chave são procuradas pela ordem referida

("rwrun60", "r25run32", "module", "userid") e, se nenhuma for encontrada, a aplicação assume que se trata de uma chamada a um outro comando do sistema, e não a um *report* (os mais comuns são chamadas ao *Internet Explorer* e à calculadora do *Windows*). Aqui existe a possibilidade de a aplicação falhar. Esta abordagem funcionou em todos os *forms* nos quais a aplicação foi testada e, tendo em conta os métodos utilizados na CPCHS, deverá funcionar para todos eles. Contudo, não está totalmente garantido que a aplicação não falhe neste ponto, substituindo um *host* por *cpc_hosts.cpc_host* indevidamente. Esta substituição indevida não é detectada em compilação. Contudo, a execução do *form* na plataforma *web* permite detectar o problema, aquando da invocação do *report* cuja chamada foi substituída por *cpc_host.cpc_hosts*. Na plataforma *web*, o *report* deveria ser aberto dentro de uma janela do *Internet Explorer*, sendo executado no lado do servidor. Com a chamada errada, a aplicação tenta abrir o *report* no lado do cliente, assumindo que existe aí o *runtime* do *Forms 6i*. Se esse *runtime* existir, e o *report* estiver acessível a partir daí (que é o que acontece com as máquinas de desenvolvimentos da CPCHS) o *report* é aberto localmente (numa aplicação própria que consiste no *runtime* dos *reports*). Caso contrário, o *report* não é aberto, e uma mensagem de erro é apresentada. Em qualquer um dos casos é possível o utilizador aperceber-se do problema.

Como foi já referido, esta abordagem não é 100% fiável, mas é aceitável, dado que a quantidade de substituições erradas efectuadas nos testes foi de 0%, e uma substituição errada pode ser detectada sem grandes dificuldades, caso aconteça.

O problema descrito ocorre quando nenhuma das palavras-chave é encontrada (pois a aplicação assume sempre que se trata de uma chamada a outro comando e não a um *report*). Se, por outro lado, qualquer uma das palavras-chave for encontrada, a aplicação tem a certeza de que se trata de uma chamada a um *report*. Sabe, portanto, que o *host* encontrado deve ser substituído por *reports_pck.abre_report*. Mas existe ainda a possibilidade de não conseguir extrair os parâmetros necessário à invocação deste último procedimento. Como foi já referido, as palavras-chave são procuradas pela ordem atrás apresentada. Se a palavra-chave encontrada for "rwrun60" ou "r25run32", continua o processamento na tentativa de encontrar os parâmetros "MODULE" e "USERID", necessários à invocação do novo procedimento. Se, por outro lado, não encontrar "rwrun60" ou "r25run32", mas apenas "MODULE" ou "USERID", sabe que se trata de um *host* que abre um *report*, mas não vai poder continuar. Isto porque é importante saber se era usado o comando "rwrun60" ou o "r25run32". O nome do comando é passado à nova função no 4º parâmetro, aquele que é opcional (se for omitido, a função assume que se trata do comando "rwrun60"). Sem esta informação, a aplicação força mais uma vez a ocorrência de um erro na compilação, para obrigar o utilizador a corrigir o *host* manualmente.

A situação que falta descrever é aquela em que, tendo a certeza de que se trata de uma chamada a *report*, a aplicação encontrou ainda na *string* o nome do comando ("rwrun60" ou "r25run32"). Procura de seguida, na mesma *string*, os parâmetros "MODULE" e "USERID". Se encontrar os dois, tem então todos os parâmetros de que precisa para a invocação de *reports_pck.abre_report*. A chamada original ao *host* é colocada em comentário, sendo acrescentada a chamada à nova função imediatamente a seguir. Se não for possível encontrar pelo menos um dos parâmetros obrigatórios, é mais uma vez introduzido um erro na chamada, forçando erro na compilação, e voltando ao início do ciclo.

5.2.10 FormLog

Descrição: Armazena toda a informação relativa ao processamento de um *form*.

Constructor Detail

`public FormLog(FormConverter6i form)`

Construtor para esta classe.

Method Detail

`public int getNumHostReportsErrors()`

Devolve o nº de erros encontrados a corrigir chamadas a reports com *host*().

`public void setNumHostReportsErrors(int num)`

Define o nº de erros encontrados a corrigir chamadas a reports com *host*().

`public boolean getFoundErrors6i()`

Devolve *true* se foram encontrados erros a compilar em 6i, *false* caso contrário.

`public void setFoundErrors6i(boolean found)`

Indica se foram encontrados erros a compilar em 6i.

`public boolean getFoundErrors9i()`

Devolve *true* se foram encontrados erros em 9i, *false* caso contrário.

`public void setFoundErrors9i(boolean found)`

Indica se foram encontrados erros a compilar em 9i.

`public void addInsertedLib(String libName)`

Adiciona a biblioteca indicada à lista de bibliotecas que foram adicionadas ao *form*. Se a biblioteca já tiver sido adicionada, não é adicionada de novo.

`public void addRemovedLib(String libName)`

Adiciona a biblioteca indicada à lista de bibliotecas que foram removidas do *form*.

`public ArrayList getInsertedLibs()`

Devolve um *array* com os nomes das bibliotecas que foram adicionadas ao *form*.

`public ArrayList getRemovedLibs()`

Devolve um *array* com os nomes das bibliotecas que foram removidas do *form*.

`public void setTemplateReloaed(boolean reloaded)`

Define o valor do atributo que indica se a *template* foi carregada ou não.

```
public boolean isTemplateReloaded()
```

Devolve o valor da propriedade que indica se a *template* foi recarregada ou não.

```
public int getNumCreate_Timer()
```

Devolve o nº de ocorrências de *create_timer* encontradas no *form*.

```
public int getNumText_IO()
```

Devolve o nº de ocorrências de *text_io* encontradas no *form*.

```
public int getNumTool_Env()
```

Devolve o nº de ocorrências de *tool_env* encontradas no *form*.

```
public int getNumGet_File_Name()
```

Devolve o nº de ocorrências de *get_file_name* encontradas no *form*.

```
public int getNumHostsOther()
```

Devolve o nº de ocorrências de *host* encontradas no *form*, para outras chamadas que não a invocação de *reports*.

```
public int getNumHostsReports()
```

Devolve o nº de ocorrências de *host* encontradas no *form*, para chamadas a *reports*.

```
public int getNumRun_Products()
```

Devolve o nº de ocorrências de *run_product* encontradas no *form*.

```
public int getNumMouseLeave()
```

Devolve o nº de *triggers when-mouse-leave* encontrados no *form*.

```
public int getNumMouseEnter()
```

Devolve o nº de *triggers when-mouse-enter* encontrados no *form*.

```
public void addWinApiCall(String call)
```

Regista uma nova ocorrencia de uma chamada a *WinAPI**.

```
public int getWinApiCallCount(String call)
```

Devolve o nº de ocorrências de chamadas à *WinAPI** indicada.

```
public Object setWinApiCallCount(String call,
                                 int count)
```

Define o nº de ocorrências da chamada a *WinAPI** indicada no parâmetro *call*.

```
public String[] getWinApiCalls()
```

Devolve um *array* com os nomes das chamadas a *WinAPI** encontradas.

```
public int getNumWin_API_Environment()
```

Devolve o nº de chamadas a *Win_Api_Environment* encontradas.

```
public void setNumRun_Products(int numRun_Products)
```

```
public void incNumRun_Products()
```

```
public void setNumHostsReports(int numHostsReports)
```

```
public void incNumHostsReports()
```

```
public void addNumHostsReports(int numHostsReports)
```

```
public void setNumHostsOther(int numHostsOther)
```

```
public void incNumHostsOther()
```

```
public void addNumHostOther(int numHostsOther)
```

```
public void setNumGet_File_Name(int numGet_File_Name)
```

```
public void incNumGet_File_Name()
```

```
public void setNumTool_Env(int numTool_Env)
```

```
public void incNumTool_Env()
```

```
public void setNumText_IO(int numText_IO)
```

```
public void incNumText_IO()
```

```
public void setNumCreate_Timer(int numCreate_Timer)
```

```
public void incNumCreate_Timer()
```

```
public void setNumMouseEnter(int numMouseEnter)
```

```
public void incNumMouseEnter()
```

`public void` **`setNumMouseLeave`**`(int numMouseLeave)`

`public void` **`incNumMouseLeave`**`()`

`public void` **`makeLog`**`(String fileName)`

Cria o ficheiro de *log* com a informação deste *FormLog*, com o nome indicado. Se já existir um ficheiro com esse nome, a informação é acrescentada no final (*append*).

`public void` **`makeLogExcel`**`(String fileName)`

Cria o ficheiro de *log* em versão de compatibilidade com o *log Excel*, usando o nome de ficheiro indicado.

---

6 Avaliação de Resultados

Neste capítulo serão apresentados os resultados obtidos nos testes finais de utilização do protótipo da *Migration Tool*. Deixando já antever algumas conclusões, são aqui apresentadas estatísticas do resultado da conversão de algumas das aplicações da CPCHS para a plataforma *web* com a *Migration Tool*. Foram convertidas na sua totalidade ou quase totalidade as aplicações "EPR – *Processo Clínico Electrónico*", "Gestão Hospitalar", e "Processo de Enfermagem".

Os resultados apresentados na Tabela 1 referem-se à conversão de 223 *forms* através da *Migration Tool* (com carregamento da *template*), constituindo uma boa estimativa do desempenho que se pode esperar na utilização da aplicação num ambiente real. São comparados nesta tabela os valores obtidos antes e depois da implementação do método *fixCompileErrors()*, responsável por corrigir erros relacionados com falta de bibliotecas de funções. Isto permite também obter uma estimativa da quantidade de *forms* que não seria possível compilar em 6i sem alterações, independentemente de se tratar de uma conversão para a plataforma 9i ou não.

Os testes foram efectuados numa máquina com as seguintes características: processador Pentium 4, a 2.4GHz; 504 MB de memória RAM; disco rígido IDE de 7200 rpm. Esta informação é relevante apenas para os resultados da coluna "Duração", pois os restantes valores não são influenciados pelas características da máquina.

	Compilação em 6i			***Compilação em 9i***			
	Com erros	*Total*	*% sucesso*	*Com erros*	*Total*	*% sucesso*	*Duração (min)*
Sem fixCompileErrors	*32*	*223*	***85,65%***	*40*	*223*	***82,06%***	*48*
Com fixCompileErrors	*14*	*223*	***93,72%***	*22*	*223*	***90,13%***	*51*

Tabela 1 *– Resultado da conversão de 223 forms da Gestão Hospitalar com a Migration Tool. É discriminada a percentagem de forms compilados com sucesso para cada uma das plataformas (cliente-servidor e web), sendo ainda comparados os resultados da conversão dos mesmos forms antes e depois da implementação do método de correcção de erros de compilação por falta de bibliotecas.*

Os *forms* utilizados no processo de conversão antes da implementação do método *fixCompileErrors()* são exactamente os mesmos que foram usados na conversão depois da implementação desse método. Tratam-se de *forms* da "Gestão Hospitalar", embora não constituam a totalidade da aplicação. Das várias aplicações da CPCHS, esta é provavelmente aquela cujos *forms* poderiam levantar mais problemas, já que se tratam de *forms* relativamente mais complexos que os restantes, pelo que os resultados obtidos podem ser encarados como uma visão pessimista, esperando-se melhores taxas de conversão com sucesso noutras situações.

Os resultados obtidos permitem concluir que as melhorias introduzidas ao nível da compilação com o método *fixCompileErrors* permitiram aumentar significativamente a

percentagem de *forms* compilados e convertidos com sucesso. No caso dos testes aqui apresentados, conseguiu-se converter 18 *forms* a mais com esse melhoramento, tanto em 6i como em 9i.

A taxa de compilação com sucesso em 9i, conforme se pode verificar na tabela, é sempre inferior à taxa correspondente em 6i. Isto acontece porque, apesar de ter sido já levado a cabo um enorme esforço para criar substitutos para as bibliotecas que já não são suportadas em 9i, existem algumas que não foram ainda substituídas. Consequentemente, o ficheiro que indica à *Migration Tool* quais são as bibliotecas que deve substituir não contempla essas para as quais não existe uma nova versão. Quando os *forms* são abertos com a versão 9i da JDAPI, estas bibliotecas não são encontradas, e os *forms* são compilados sem elas. Naturalmente, ocorrem erros durante a compilação, já que os *forms* invocam funções e procedimentos que o compilador não consegue encontrar.

As novas bibliotecas podem ser versões das anteriores alteradas para suportar as duas plataformas, ou pode acontecer que seja mesmo criada uma nova biblioteca especificamente para a plataforma *web*[19]. Em qualquer um dos casos, o compilador para 6i "vê" uma versão da biblioteca, e o compilador para 9i "vê" a outra, pois cada versão do *Oracle Forms* cria uma chave no *registry* do *Windows* indicando o local onde deve procurar as bibliotecas.

Relativamente à duração do processo de conversão dos *forms*, a avaliação que se faz é bastante positiva. Como já se esperava, o processo de conversão através da *Migration Tool* é muito mais rápido do que a conversão manual. Em média, a aplicação demorou cerca de 0.2 minutos a converter um *form*, enquanto que manualmente se demora cerca de 20 minutos.

Foram efectuadas conversões integrais (ou quase integrais) de algumas aplicações da CPCHS com a *Migration Tool*, depois de comprovada a sua eficiência (apesar de se tratar ainda de um protótipo com vários melhoramentos passíveis de serem implementados). As aplicações convertidas foram o "*Processo Clínico Electrónico*", a "Gestão Hospitalar", e o "Processo de Enfermagem". A conversão destas aplicações foi levada a cabo com a versão mais recente da *Migration Tool*, que incluía já a correcção dos erros de compilação por falta de bibliotecas.

O "*Processo Clínico Electrónico*" trata-se de uma aplicação diferente das restantes. É constituído por um número de *forms* muito reduzido. Contudo, os *forms* principais desta aplicação (GHPC7900 e GHPC8000) são os maiores e mais complexos de todos, consistindo no teste mais difícil para a *Migration Tool*. O GHPC8000 não foi sequer submetido à conversão por parte da *Migration Tool*. Este inclui funcionalidades não suportadas na *web* para as quais não foram ainda desenvolvidos substitutos, nomeadamente vários controlos OCX utilizados para efectuar interligação com outras aplicações. Sabe-se à partida, portanto, que este *form* não pode ser compilado para a plataforma *web* sem que os componentes OCX referidos sejam removidos, assim como todas as referências aos mesmos em código PL/SQL. Naturalmente, a *Migration Tool* não poderia ser programada para efectuar esta tarefa, pois trata-se de retirar funcionalidades ao *form*. Este tem que ser convertido manualmente, sendo removidas as funcionalidades não suportadas na *web* (note-se que a versão *web* do "*Processo Clínico Electrónico*" não está ainda em utilização em nenhum dos clientes da CPCHS). Os

[19] A criação das novas versões das bibliotecas é uma tarefa que começou a ser levada a cabo ainda antes do início deste estágio, não sendo objectivo do mesmo a sua criação. Daí ser aceitável que a *Migration Tool* não consiga converter com sucesso uma percentagem de *forms* tão significativa por este motivo.

restantes *forms* do "*Processo Clínico Electrónico*", contudo, foram convertidos com sucesso, incluindo o GHPC7900, que é também bastante complexo. A conversão desta aplicação para a nova plataforma teve uma taxa de sucesso de 95%, sendo que dos 20 *forms* que a constituem[20], 19 foram convertidos pela *Migration Tool* sem problemas. O único *form* não convertido trata-se do GHPC8000, pelos motivos já atrás referidos.

A "Gestão Hospitalar" é a maior aplicação da CPCHS em número de *forms*. Estão actualmente em utilização efectiva 550 *forms*, de um total de 750 já desenvolvidos. Os 200 não utilizados já estão obsoletos, tendo as suas funcionalidades sido integradas noutros *forms*, ou simpesmente deixaram de ser necessárias. Até à data de redacção deste relatório, foram convertidos pela *Migration Tool* (com ou sem sucesso) 450 desses 550 *forms* em utilização na "Gestão Hospitalar". As taxas de conversão com sucesso são aproximadamente aquelas apresentadas na tabela 1. Cerca de 90% dos *forms* foram totalmente convertidos com sucesso (compilaram correctamente em 6i e em 9i), e cerca de 94% apenas foram compilados com sucesso na plataforma 6i (estes não podem, portanto, ser considerados como convertidos com sucesso).

No "Processo de Enfermagem", 72 dos 75 *forms* foram convertidos com sucesso (compilaram correctamente em 6i e em 9i), o que corresponde a uma taxa de sucesso de 96%. Neste caso, os *forms* que não foram compilados correctamente continham erros já na versão original, pelo que se conclui que os *forms* do "Processo de Enfermagem" não apresentaram qualquer problema na conversão propriamente dita.

[20] Na verdade, o *Processo Clínico* é constituído por 5 *forms*. Os restantes tratam-se de *forms* usados como módulos reutilizáveis, e contêm objectos que são adicionados aos *forms* que deles necessitam. Neste caso, são usados no GHPC8000, sob a forma de sub-ecrãs dispostos no painel central que este *form* disponibiliza para o efeito.

7 Conclusões e perspectivas de trabalho futuro

Findo o período de estágio, conclui-se que todo o trabalho foi realizado com sucesso. Ficam, contudo, alguns desenvolvimentos pendentes, deixando em aberto a continuidade do trabalho no futuro.

Relativamente à *Migration Tool*, conseguiu-se que esta realizasse todas as tarefas que se previa necessárias inicialmente, e ainda outras necessárias para resolver alguns problemas que foram encontrados durante o seu desenvolvimento. Problemas como o da falta de bibliotecas dos *forms* ainda antes de serem processados pela *Migration Tool*, que agora é resolvido pela aplicação.

A aplicação, contudo, não cobre totalmente todos os aspectos da migração das aplicações para a plataforma *web*. Num trabalho futuro, fará sentido tentar resolver o problema do carregamento da *template* (descrito no Anexo B). A conclusão da interface gráfica da aplicação seria também importante. O algoritmo de processamento dos *hosts* é outro ponto onde a aplicação pode ser melhorada. Aqui, uma melhoria seria, por exemplo, tentar encontrar no código dos *forms* os dados que o programa necessita para saber de que tipo de *host* se trata, e também para identificar os vários parâmetros usados na invocação do comando externo. Na versão actual, a aplicação "desiste" se não encontrar esses dados directamente na instrução onde o *host* é invocado.

Considerando apenas o protótipo da *Migration Tool* implementado, pode-se dizer que traz já vantagens à empresa, acelerando consideravelmente o processo de migração para a nova plataforma, ao mesmo que corrige alguns problemas existentes com os *forms*, independentemente do processo de migração.

Relativamente aos desenvolvimentos do *Processo Clínico Electrónico*, existe ainda muito trabalho a fazer até que a nova versão fique pronta. Contudo, não era objectivo deste estágio ter a nova versão concluída. Fica, no entanto, a nota de que existe bastante trabalho para fazer nesta área. A especificação das novas interfaces gráficas ficou concluída, tendo estas sido já usadas em muitos ecrãs da aplicação. Contudo, há alguns ainda onde esta precisa de ser implementada.

Referências e Bibliografia

Tecnologia *Java* – http://java.sun.com/

Tecnologia *Oracle Forms* – http://www.oracle.com/technology/products/forms/index.html

Oracle Forms 9i – http://www.oracle.com/technology/products/forms/techlisting9i.html

JDAPI – http://www.oracle.com/forms/10g/help/__inl.inline.true/__inl.topic.
jdapi_overview_html/__inl.html

ANEXO A: *Forms Builder* e a estrutura dos *Forms*

Nesta secção será feita uma breve introdução à ferramenta *Forms Builder*, não com o objectivo de servir de tutorial para a utilização da aplicação, mas sim numa perspectiva orientada à estrutura dos *forms*. Pretende-se que o leitor fique familiarizado com os conceitos que esta ferramenta introduz no desenvolvimento de aplicações, e a melhor forma de o fazer é descrevendo o modo como esta organiza um *form*.

O conteúdo deste texto é válido para qualquer uma das versões 6i ou 9i do *Forms Builder*, já que a grande diferença entre as duas reside na estrutura das aplicações finais e forma como são distribuídas, e não no modo como são desenvolvidas. O ambiente de desenvolvimento é muito semelhante em ambas as versões, assim como o seu modo de funcionamento. Difere o formato das aplicações criadas, já que a plataforma sobre a qual são executadas, essa sim, é completamente diferente.

Forms Builder: A Ferramenta

O *Forms Builder* é uma ferramenta do tipo RAD (*Rapid Application Development*), que permite o rápido desenvolvimento de aplicações sobre a plataforma *Oracle Forms*.

Com esta ferramenta é possível desenvolver aplicações-cliente com ligação a Bases de Dados Oracle de uma forma muito simples e rápida, e totalmente transparente para o programador.

O desenvolvimento das aplicações é conseguido através das facilidades disponibilizadas pelo IDE (*Integrated Development Environment*) do *Forms Builder*, nomeadamente o seu editor gráfico de interfaces, e o editor de código. No editor gráfico é possível desenhar toda a interface gráfica das aplicações, sendo possível definir grande parte do seu comportamento mesmo sem escrever uma única linha de código. As funcionalidades específicas da aplicação a desenvolver, a lógica do negócio, podem ser implementadas de seguida na linguagem de programação disponibilizada pela ferramenta, o PL/SQL.

O PL/SQL é uma linguagem de programação *procedimental*, sem suporte para programação "baseada em" ou "orientada a" objectos, sendo por isso um pouco limitada. Contudo, tendo em conta o propósito a que se destina, o de complementar as funcionalidades das aplicações criadas no editor gráfico, revela-se suficientemente versátil. As facilidades que disponibiliza para interagir com a Base de Dados são a característica que mais se destaca nesta linguagem. É possível incluir instruções SQL directamente no código dos programas, sendo extremamente fácil e directo ler, alterar ou adicionar registos à Base de Dados.

O IDE do *Forms Builder* é constituído por 4 componentes principais:

- *Object Navigator* – apresenta todos os constituintes do *form* hierarquicamente, permitindo seleccioná-los, de modo a que possam ser editados de seguida. O *Object Navigator* disponibiliza dois modos de agrupar os componentes do *form*:
    - *Ownership View* – agrupa os objectos de acordo com a sua hierarquia lógica.
    - *Visual View* – agrupa os objectos de acordo com a sua representação gráfica (por exemplo, todos os objectos que aparecem em determinada janela são colocados hierarquicamente abaixo dessa janela).

- *Property Palette* – apresenta uma lista de propriedades editáveis do objecto actualmente seleccionado.
- *Layout Editor* – editor gráfico; permite criar a interface gráfica das aplicações, disponibilizando mecanismos que facilitam o processo de disposição dos componentes visuais.
- *PL/SQL Editor* – editor de código PL/SQL.

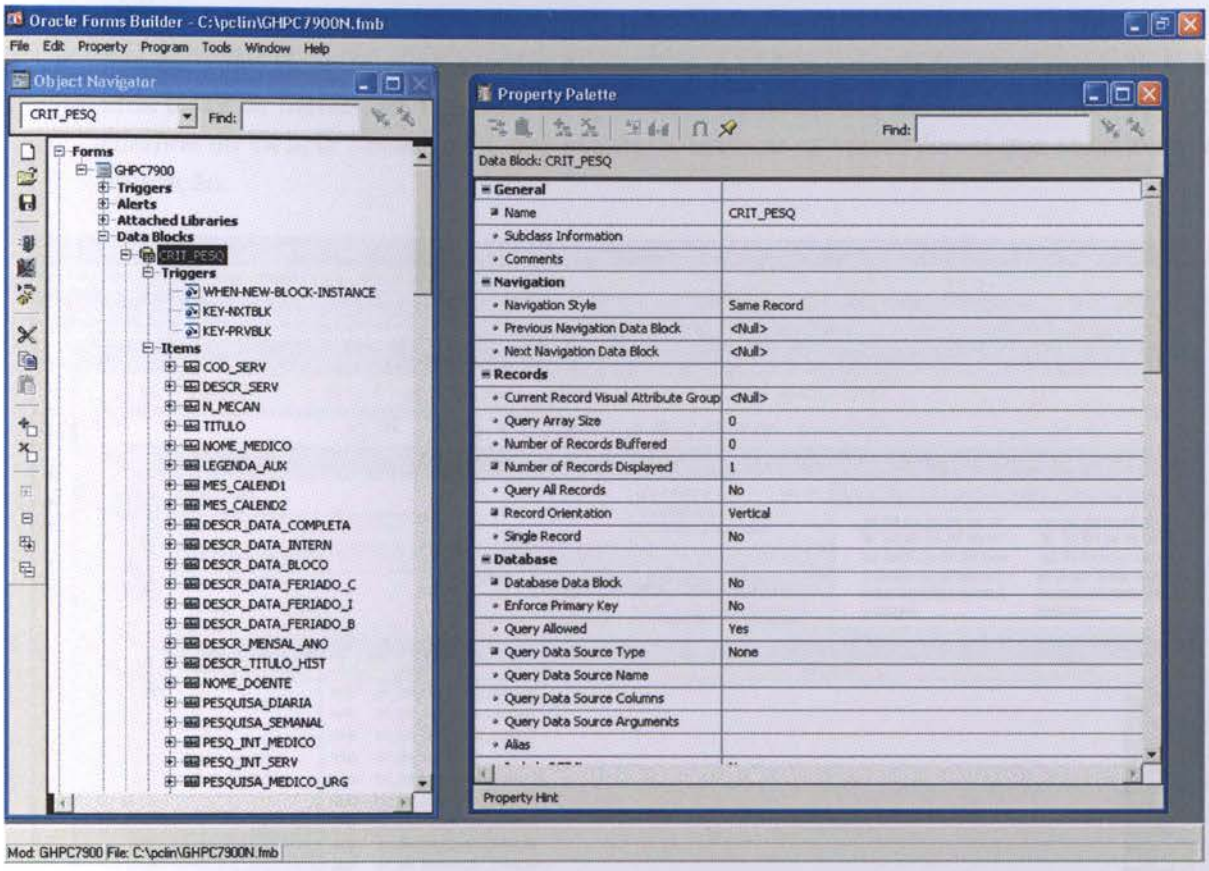


Fig.24 *– IDE do Forms Builder 6, com um form aberto. Estão visíveis o Object Navigator e a Property Palette.*

O *Forms Builder* permite a compilação dos *forms,* e imediata execução das aplicações, sempre a partir do mesmo ambiente de desenvovimento.

Esta ferramenta utiliza os seguintes formatos de ficheiros:

- **.fmb* – um ficheiro *fmb* contém o código fonte de um *form.* Inclui não só o código PL/SQL associado ao *form*, mas também o respectivo *layout*, e propriedades de cada objecto. É possível criar e editar este tipo de ficheiros com o *Forms Builder.*
- **.fmx* – um ficheiro *fmx* consiste num *form* compilado. Estes ficheiros são interpretados pelo *runtime* do *Oracle Forms* aquando da execução da aplicação. O *Forms Builder* é capaz de criar estes ficheiros a partir dos *fmb*, mas não é possível editá-los.
- **.pll* – (PL/SQL Library) um ficheiro *pll* consiste numa biblioteca de procedimentos e funções PL/SQL, que podem estar organizados em *packages*. Para além dos próprios

ficheiros *fmb*, é possível guardar código PL/SQL em bibliotecas externas, de modo a serem facilmente reutilizadas. O *Forms Builder* permite a edição e criação deste tipo de ficheiros através do seu editor de PL/SQL.

- **.olb* – (*Object Library*) um ficheiro *olb* consiste num conjunto de objectos constituintes dos *forms*. À semelhança das bibliotecas de PL/SQL, estas bibliotecas de objectos existem no sentido de promover a reutilização de componentes. Desta feita, permitem a reutilização de quaisquer componentes dos *forms*, e não somente de código.
- **.mmb* – um ficheiro *mmb* contém a estrutura e código associado a um menu (a toda uma barra de menus, na verdade) da aplicação. Estes menus são utilizados ao nível do *runtime* do *Oracle Forms* para, por exemplo, invocar os vários *forms* que constituem a aplicação.

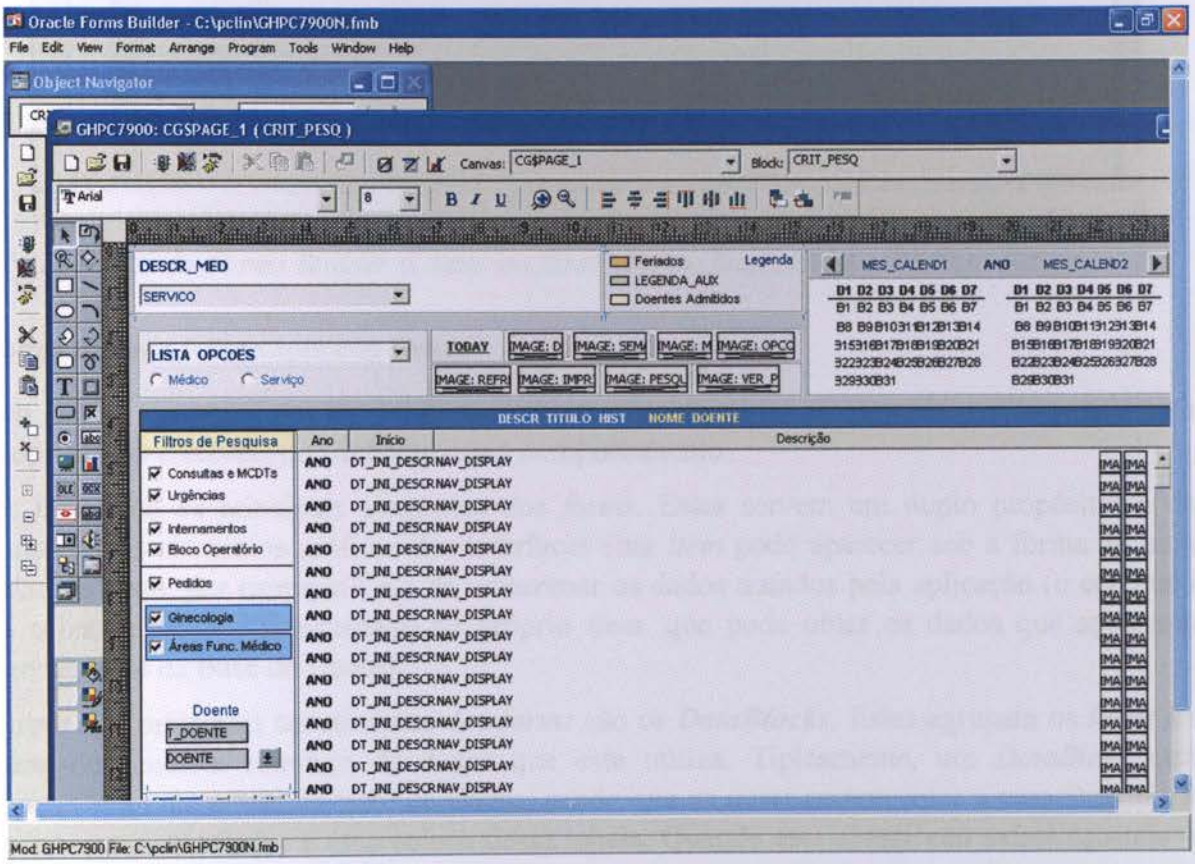


Fig.25 *– IDE do Forms Builder 6, com um form aberto. Está visível o Layout Editor.*

Para além das várias facilidades de desenvolvimento disponíveis, o *Forms Builder* possui uma séria de assistentes (*wizards*) que criam todo o código PL/SQL e outros componentes necessários à realização das tarefas mais comuns, no sentido de tornar ainda mais rápida a criação de aplicações completas, com funcionalidades avançadas. A título de exemplo, o *Data Block Wizard* permite a criação dos vários componentes necessários à obtenção de uma aplicação que apresenta registos de uma tabela com ligação a outra tabela através de uma chave estrangeira, do género *master-detail*. Todo o código de sincronização entre as duas tabelas é criado automaticamente pelo *wizard*.

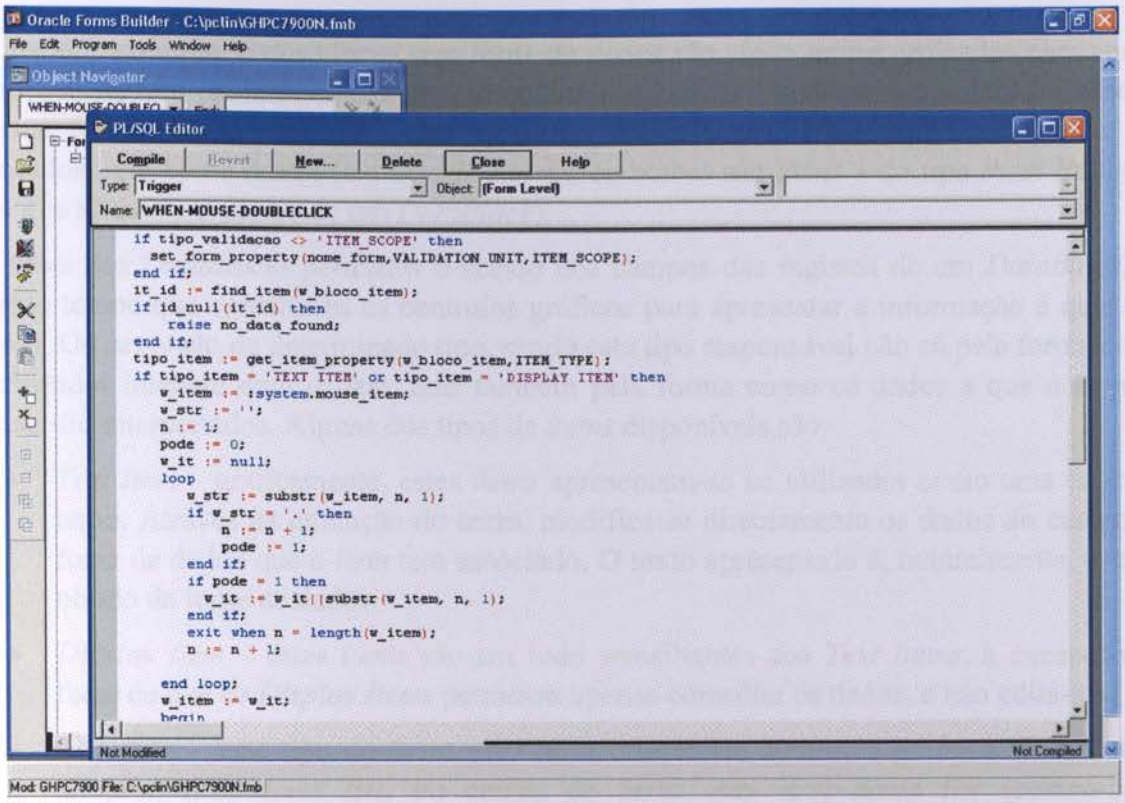


***Fig.26** – IDE do Forms Builder 6, com um form aberto. Está visível o PL/SQL Editor.*

Forms Builder: A Estrutura dos *Forms*

Um *form* é composto por um conjunto de elementos de vários tipos, cada um dos quais com propriedades editáveis que definem o seu comportamento.

Os ***items*** são os principais elementos dos *forms*. Estes servem um duplo propósito: o de constituir os elementos gráficos das interfaces (um *item* pode aparecer sob a forma de uma caixa de texto, por exemplo), e o de armazenar os dados tratados pela aplicação (o conteúdo da caixa de texto é armazenado no próprio *item,* que pode obter os dados que apresenta directamente da Base de Dados).

Outros dos principais constituintes dos *forms* são os ***DataBlocks***. Estes agrupam os *items* do *form*, definindo a estrutura de dados que este utiliza. Tipicamente, um *DataBlock* está associado a uma tabela da Base de Dados, sendo que os *items* pertencentes a esse *DataBlock* podem estar associados a uma coluna dessa tabela. Quando esta associação existe (quando o *DataBlock* está associado a uma tabela, e um *item* desse *DataBlock* a uma coluna da mesma), então é possível utilizar esse *item* para apresentar dados dessa coluna da tabela, para alterar esses dados, e para inserir novos registos. O *DataBlock* é então usado para, localmente, guardar registos da fonte de dados. O acesso aos campos desses registos é conseguido através do *items* do *DataBlock*. As fontes de dados de um *DataBlock* são, tipicamente, tabelas da Base de Dados. Mas a fonte de dados pode também ser uma função que devolva os registos, ou até uma instrução SQL do tipo SELECT, que devolve o conjunto de registos (esta abordagem permite juntar dados de várias tabelas diferentes, ao contrário da utilização directa da tabela como fonte). Existe ainda a possibilidade de se criar um *DataBlock* que não tenha uma fonte de dados associada. Estes blocos são controlados exclusivamente no código PL/SQL da aplicação. Podem-se inserir e remover registos destes *DataBlocks*, assim como alterar os existentes, sem que estas alterações tenham quaisquer repercussões numa Base de

Dados, por exemplo. Estes blocos sem fonte de dados são ainda muito utilizados para conter *items* que servem exclusivamente para controlar a aplicação, e não para manipular dados de alguma forma. Os botões que eventualmente possam existir numa aplicação podem ser colocados num bloco destes (em *Oracle Forms,* os botões são *items* – do tipo *Push Button* –, pelo que só existem dentro de um *DataBlock*).

Os *items* dos *DataBlocks* perm*item* o acesso aos campos dos registos de um *DataBlock*, ao mesmo tempo que constituem os controlos gráficos para apresentar a informação à qual dão acesso. Os *items* são de determinado tipo, sendo este tipo responsável não só pela forma como o utilizador interage com o *item*, mas também pela forma como os dados a que o *item* dá acesso são interpretados. Alguns dos tipos de *items* disponíveis são:

- *Text Item* – graficamente, estes *items* apresentam-se ao utilizador como uma caixa de texto. Através da alteração do texto, modifica-se directamente os dados do campo da fonte de dados que o *item* tem associado. O texto apresentado é, naturalmente, o texto obtido da fonte de dados.
- *Display Item* – estes *items* são em tudo semelhantes aos *Text Items*, à excepção do facto de que os *Display Items* perm*item* apenas consultar os dados, e não editá-los.
- *List Item* – este tipo de *items* permite a criação de controlos gráficos do tipo lista simples, *drop-down list*, ou caixas de texto com *drop-down list* (*combo-box*). Perm*item* fazer o mapeamento entre o valor da fonte de dados e o valor apresentado ao utilizador.
- *Push Button* – estes *items* apresentam-se ao utilizador sob a forma de botões. Não é possível associar *Push Buttons* a uma fonte de dados, pelo que estes *items* aparecem tipicamente em blocos sem fonte de dados.
- *CheckBox* – representam as típicas caixas de selecção, sendo possível associar-lhes um de dois valores. São ideais para controlar campos de dados do tipo booleano.
- *Radio Group* – estes *items* são um pouco mais complexos que os anteriores. Um *Radio Group* é na verdade um grupo de *Radio Buttons*. Note-se que os *Radio Buttons* só podem aparecer dentro de *items Radio Group*, e os *Radio Buttons* não são *items* na acepção do conceito no *Forms Builder*. Um *Radio Button* não pode nunca estar associado a um campo da fonte de dados. O *Radio Group* é que pode ter uma associação desse género, sendo que cada um dos seus *Radio Buttons* corresponde a um valor que o campo da fonte de dados pode tomar. Naturalmente, apenas um dos *Radio Buttons* do *Radio Group* pode estar seleccionado, sendo o valor associado a esse *Radio Button* aquele que o campo irá tomar.

Existem ainda vários outros tipos de *items*, mas os mais importantes no tratamento de dados são aqueles acima descritos. A título de curiosidade, outros tipos possíveis são *Image Item*, *ActiveX Control*, *OLE Container*, *VBX Control, Bean Area*, etc.

Associados aos *items*, aos *DataBlocks*, ou directamente aos *forms,* podem existir ***triggers***. Tratam-se de procedimentos PL/SQL especiais que são invocados automaticamente pelo *runtime* aquando da ocorrência de determinados eventos (por exemplo, o *trigger when-button-pressed* de um *item* do tipo *Push Button* é *disparado* quando o utilizador da aplicação pressiona esse botão). Existem muitos tipos de *triggers* diferentes, sendo que nem todos eles podem ser usados em todas as circunstâncias. *Triggers* como *when-button-pressed* só podem ser definidos ao nível dos *items*, e só são de facto utilizados quando esses *items* são do tipo

Push Button. O *trigger when-new-form-instance*, por exemplo, só pode ser definido ao nível do *form*, assim como *when-block-leave* só faz sentido ao nível dos *DataBlocks*. Muitos *triggers*, contudo, estão disponíveis nos três níveis referidos (nível do *form*, do bloco e do *tem*). Aqui o programador pode optar por utilizar o *trigger* onde for mais apropriado. É-lhe ainda dada a liberdade de definir o mesmo *trigger* em mais do que um nível diferente. Por exemplo, é possível definir o *trigger post-query* ao nível do bloco e, ao mesmo tempo, ao nível do *item*.

A execução de código PL/SQL é sempre iniciada através de um *trigger*. No entanto, nem todo o código PL/SQL da aplicação tem que se encontrar nos *triggers*. É possível definir ***ProgramUnits*** ao nível de cada *form*, que consistem em funções (que possuem valor de retorno) e procedimentos (que não possuem valor de retorno) escritos em PL/SQL. As *ProgramUnits* podem ainda apresentar uma estrutura mais complexa, consistindo em *packages* de funções e procedimentos, ao nível dos quais podem ser definidas variáveis. O código das *ProgramUnits* pode ser invocado a partir dos *triggers* do *form*, e também a partir de outras *ProgramUnits*.

É ainda possível colocar as *ProgramUnits* dentro de bibliotecas, e não dentro dos *forms*. Deste modo, é possível reutilizar o código sempre que necessário. Estas bibliotecas de código são, fisicamente, ficheiros com extensão *pll*. Para que possam ser acedidas a partir de um *form*, devem ser adicionadas ao mesmo, aparecendo na sua lista de objectos constituintes sob a forma de ***AttachedLibraries***.

Na mesma filosofia de reutilização, é possível criar ***ObjectLibraries***. Fisicamente, tratam-se de ficheiros com extensão *olb*, sendo que não têm uma representação directa dentro de um *form*. Tratam-se de conjuntos de ***ObjectGroups*** e estes, sim, já surgem como constituintes dos *forms*. Um *ObjectGroup* é constituído por um conjunto de objectos de qualquer tipo. Quando um *ObjectGroup* é adicionado a um *form*, os objectos que contém passam a estar disponíveis nesse *form* da mesma forma que os restantes objectos que o constituem. O procedimento que o programador deve levar a cabo para reutilizar os componentes existentes numa *ObjectLibrary* é o seguinte: (1) abrir o *form* onde pretende utilizar os objectos; (2) abrir a *ObjectLibrary* que contém o *ObjectGroup* cujos objectos pretende utilizar (neste ponto, estão visíveis no *Object Navigator* o *form* e a *ObjectLibrary*); (3) *arrastar* o *ObjectGroup* cujos objectos pretende utilizar da *ObjectLibrary* para o *form;* (4) responder à pergunta que é colocada pelo *Forms Builder*, indicando se pretende simplesmente "copiar" os objectos para o *form*, ou se pretende "herdá-los". A opção de "copiar" faz com que os objectos do *ObjectGroup* passem a fazer parte do *form*, tal como se tivessem sido criados aí directamente. A opção "herdar" copia os objectos para o *form*, mas estes mantém uma ligação com os objectos originais do *ObjectGroup*. Deste modo, se estes forem modificados, as alterações reflectem-se nos objectos do *form*. Da mesma forma que um *ObjectGroup* pode ser colocado numa *ObjectLibrary*, para facilitar a sua reutilização, este pode também ser guardado directamente num *form*. O propósito destes *ObjectGroups* continua a ser o mesmo, o de promover a reutilização de componentes. A diferença é que, ao invés de se ter o *ObjectGroup* dentro de uma *ObjectLibrary*, este surge dentro de um *form*. É possível "copiar" ou "herdar" um *ObjectGroup* de um *form* para outro.

Os *DataBlocks* agrupam os *items* logicamente, de acordo com a informação que perm*item* tratar. Mas visualmente, na interface gráfica, a disposição dos *items* pode ter uma lógica completamente diferente. Os ***Canvases*** e ***Windows*** são os constituintes dos *forms* que perm*item* controlar directamente o *layout*. Um *form* tem sempre pelo menos um objecto do

tipo *window*, sendo este a janela onde o conteúdo do *form* é apresentado ao utilizador. Os *forms* devem ter também pelo menos um *canvas*. Os *canvases* agrupam os *items*, e aparecem dentro de uma janela (*window*). Qualquer *item* visível na interface gráfica tem que estar dentro de um *canvas* que, por sua vez, aparece dentro de uma janela.

Como vários *canvases* podem estar associados à mesma janela, sendo possível haver sobreposições, o posicionamento dos objectos na interface gráfica torna-se bastante flexível. Principalmente se tivermos em conta que existem vários tipos de *canvases*, desde os simples *content canvas,* passando pelos *stacked canvas*, até aos *tab canvas*.

Para além dos *items*, existem outros tipos de objectos que podem ser colocados dentro dos canvas e, portanto, ter uma representação visual. Estes objectos, denominados ***boilerplates***, perm*item* adicionar texto fixo, linhas, rectângulos ou outros polígonos, etc. Estes, ao contrário dos *items*, não estão associados a qualquer fonte de dados (não existem dentro de um *DataBlock*), e o seu conteúdo não pode ser modificado em *runtime*. Se, por exemplo, se criar uma legenda para um botão com um *boilerplate* do tipo texto, esse texto não pode ser alterado no código da aplicação durante a execução do programa.

Um *DataBlock* dá acesso a registos da sua fonte de dados, permitindo visualizá-los e editá-los através da representação gráfica que os seus *items* podem possuir. Um *item,* do tipo *TextItem,* por exemplo, não tem que corresponder graficamente a uma única *Text Box.* O número de componentes visuais que constituem o *item* depende da propriedade *Number Of Records Displayed* do mesmo. Se esta tomar o valor 1, que é o valor por defeito, um *item* corresponde a um componente gráfico (no exemplo do *Text Item* o componente gráfico é uma *Text Box*), que apresenta o valor da coluna/campo desse *item* no registo actual. Se, por exemplo, esse valor for 5, e seguindo ainda o exemplo do *Text Item*, vão existir 5 *Text Boxes* na interface gráfica, correspondendo todas elas ao mesmo *item*. Assim é possível apresentar ao mesmo tempo o valor de 5 registos da fonte de dados. Na figura 25 é possível observar este conceito. Existe um *item* chamado "Ano", cujo tipo é *Text Item.* Graficamente, este surge como uma repetição de *Text Boxes*, em número igual ao valor da propriedade *Number Of Records Displayed* do *item* (todas as palavras "Ano" que aparecem na mesma coluna são na verdade a representação visual do mesmo *item "Ano"*).

Nestes *items* com vários registos (*multi-record*), é possível destacar visualmente o registo actual (aquele que está actualmente seleccionado) através da propriedade *Current Record Visual Attribute*, que permite definir as características visuais do registo seleccionado de forma a serem diferentes dos restantes. Tanto esta propriedade (*Current Record Visual Attribute*) como *Number Of Records Displayed* podem ser definidas individualmente para cada um dos *items*, e/ou podem ser definidas para o *DataBlock.* Se um *item* tiver alguma destas propriedades definida, utiliza esse valor, caso contrário utiliza o do *DataBlock* a que pertence.

Todos os constituintes dos *forms* referidos, à excepção dos *boilerplates*, possuem um conjunto de propriedades que podem ser alteradas, em *design* ou *runtime.* As propriedades dos *boilerplates* só podem ser definidas em *design*-time. O número de propriedades que pode ser alterado para cada um dos vários componentes é enorme, o que dá ao programador uma grande flexibilidade na construção das aplicações. Estas propriedades podem ser alteradas aquando do desenvolvimento das aplicações (*design-time*) através da *Property Palette* do *Forms Builder.* As propriedades que esta permite consultar e editar mudam de acordo com o tipo de objecto que estiver seleccionado.

O código PL/SQL que constitui os *Triggers* e as *ProgramUnits* é acedido atrvés de uma propriedade destes objectos chamada *Trigger Text*.

Um outro constituinte dos *forms* é a ***PropertyClass***. Os objectos deste tipo servem unicamente para agrupar um conjunto de propriedades e respectivos valores. Podem depois ser referidos por qualquer objecto, passando esse objecto a "herdar" os valores das suas propriedades da *PropertyClass*. Isto permite definir uma única vez, na *PropertyClass*, os valores das propriedades que se pretende, referenciando a *PropertyClass* depois em qualquer objecto que se pretenda que utilize esses valores. Se estes forem alterados, as propriedades correspondentes em todos os objectos que "herdam" da *PropertyClass* serão actualizados.

As propriedades dos objectos encontram-se num de 4 estados possíveis, que estão relacionados com a questão da herança. Estes estados são:

- *Default Value* – o valor da propriedade não foi definido, assumindo o valor *default*.
- *Inherited Value* – o objecto ao qual a propriedade pertence herdou os valores das suas propriedades de uma *PropertyClass* ou de outro objecto do mesmo tipo, e a propriedade em questão assume esse valor (o valor não foi alterado pelo utilizador, mas não corresponde ao valor por defeito).
- *Overriden Default Value* – o valor da propriedade, inicialmente o valor *default*, foi alterado (pelo utilizador em *design-time*, ou no código da aplicação em *runtime*).
- *Overriden Inherited Value* – o objecto ao qual a propriedade pertence herdou os valores das suas propriedades de uma *PropertyClass* ou de outro objecto do mesmo tipo. Contudo, esse valor foi alterado (pelo utilizador em *design-time*, ou no código da aplicação em *runtime*).

Quando o valor de uma propriedade que tenha sido herdada é alterado, esta passa do estado *Inherited Value* para *Overriden Inherited Value*, e a relação com a propriedade "pai" é quebrada, pelo que subsequentes alterações à propriedade "pai" já não se reflectem na propriedade "filha".

Existe ainda outro conceito importante quando se cria interfaces gráficas no *Forms Builder*. É possível que, num dado momento, apenas uma parte dos *items* com representação gráfica estejam visíveis. O *canvas* a que estes pertencem pode estar oculto por trás de outro, por exemplo. Contudo, sempre que o foco passar para um *item* que não esteja visível[21], o canvas em que este se encontra é forçado a aparecer, de modo a que o *item* com foco fique totalmenet visível. Existe, portanto, também o conceito de *item* actual, para além do de registo actual.

Quando o *form* é executado, a interface gráfica apresentada inicialmente pode ser afectada pelo comportamento inerente a este conceito. O *form* possui uma propriedade que indica qual é o *DataBlock* inicial. Quando o *form* é executado, se se tiver definido o valor dessa propriedade, o foco é atribuído ao primeiro *item* visível do *DataBlock* inicial. Se essa propriedade não estiver definida, o foco inicial é atribuído ao primeiro *item* do primeiro *DataBlock* do *form*. Isto significa, portanto, que a ordem pela qual os *DataBlocks* aparecem no *form* é importante, caso a propriedade referida não esteja definida.

[21] O foco indica qual é o *item* no qual o cursor se encontra. O utilizador pode mudar o foco para o *item* seguinte no ciclo de foco pressionando a tecla TAB, por exemplo.

ANEXO B: O processo de migração para *Oracle Forms 9i* na CPCHS

Neste anexo, descreve-se detalhadamente o processo de migração para a plataforma *Oracle Forms 9i*, sendo referidos os vários problemas que este processo ainda possui por resolver

Compatibilidade com Oracle Forms 6i

A migração efectuada na CPCHS não é definitiva, sendo que se poderia até pôr em causa a utilização do termo "conversão", que é muitas vezes usado para referir as alterações efectuadas aos *forms*. Este processo de migração consiste em efectuar alterações aos *forms*, de modo a que seja possível compilá-los com o *Forms Builder 9i*. Os *forms*, depois de alterados, continuam a ser ficheiros *fmb* da plataforma *Oracle Forms 6i*. Estes só devem ser abertos com o *Forms Builder 9i* para serem compilados, não se podendo guardar quaisquer alterações com esta versão do *Forms Builder* (sob pena de não ser possível voltar a abrir os *forms* com o *Forms Builder 6i* para se proceder aos desenvolvimentos para essa plataforma).

Componentes Reutilizáveis

Vários *forms* que constituem as aplicações da CPCHS utilizam tecnologias que não são compatíveis com a plataforma *web*, ou então que precisam de algumas alterações na forma como são utilizados. A generalidade destes componentes encontra-se isolada em bibliotecas de funções ou de objectos (ficheiros *pll* ou *olb*). A CPCHS iniciou um processo de conversão destes componentes, de modo a torná-los compatíveis com a nova plataforma *web*. O objectivo era criar componentes compatíveis com ambas as plataformas sempre que possível, para substituir os anteriores. Quando tal não era viável, criavam-se novos componentes específicos para a plataforma *web*, mantendo-se as duas versões.

O resultado deste processo, que a CPCHS levou a cabo antes do início deste estágio, foi a criação de novas funções e procedimentos, localizados em novas bibliotecas PL/SQL (ficheiros *pll*), que tratavam de garantir o correcto funcionamento em ambas as plataformas. Relativamente às bibliotecas de objectos (ficheiros *olb*), não foi necessário criar novas versões, bastando compilar as já existentes com o compilador da plataforma 9i.

Relativamente a estes componentes reutilizáveis, o processo de migração passa então por remover as bibliotecas antigas para as quais foram criadas novas versões e substituí-las pelas novas, e alterar a invocação dos procedimentos e funções obsoletos pela invocação dos novos que se encontram nas novas bibliotecas. Na maior parte dos casos, esta substituição consiste em adicionar simplesmente o texto "cpc_" antes das chamadas aos procedimentos e funções. Estes novos métodos cujo nome começa com "cpc_", na maior parte dos casos, verificam qual é a plataforma em que a aplicação está a ser executada, e invocam depois os procedimentos e funções específicos de cada plataforma. Este passo do processo é descrito mais à frente.

Template

Outro componente utilizado por todos os *forms* da CPCHS é a *template*. Trata-se de um *ObjectGroup* que reside num *form* denominado *temp.fmb* (embora pudesse encontrar-se numa *ObjectLibrary*). Este componente é constituído por vários objectos que são herdados por todos os *forms*, implementando funcionalidades comuns a todos eles. Disponibiliza também vários objectos cuja utilização, apesar de não ser obrigatória, é recomendada, de modo a que o comportamento dos vários *forms* seja o mais consistente possível. É na *template* que se

encontram as janelas de diálogo que devem ser apresentadas quando ocorre algum erro, ou se pretende mostrar um aviso, ou quando se pretende fazer uma pergunta ao utilizador, etc. Foi necessário criar uma nova versão da *template* específica para a plataforma 9i, pois alguns componentes da versão 6i tiveram de ser substituídos por outros recorrendo a novas tecnologias. A título de exemplo, a nova *template* possui um novo *DataBlock*, denominado *WEBUTIL*, que disponibiliza vários *JavaBeans*. Estes permitem que a aplicação invoque a execução de componentes na máquina do cliente. Sem esta funcionalidade, por exemplo, não seria possível abrir a calculadora do *Windows* na máquina cliente aquando da execução das aplicações na plataforma *web*.

Existe, portanto, uma versão da *template* para a plataforma 6i e outra para a plataforma 9i. Em teoria, do ponto de vista da migração, não é necessário efectuar qualquer alteração aos *forms* por causa da *template*, pois a nova versão é utilizada automaticamente quando os *forms* são abertos no *Forms Builder 9i*. Na prática, contudo, a utilização da *template* traz alguns problemas. Estes problemas são descritos de seguida.

Por motivos que não foram ainda apurados, alguns *forms* evidenciam um comportamento anómalo em relação à *template*. Estes possuem todas as propriedades de todos os objectos da *template* no estado *Overriden Inherited Value* (ao invés do estado *Inherited Value*). Trata-se de um problema independente da migração, trazendo inconvenientes no normal desenvolvimento das aplicações (quando for feita alguma alteração na *template,* esta não se repercute nos *forms* que se encontram nesta situação). Por este motivo é que o problema não foi mais aprofundado neste estágio.

No entanto, os *forms* que se encontram nesta situação funcionam correctamente na plataforma 6i, se bem que por vezes possam estar a executar código desactualizado. Contudo, não é possível compilá-los para a plataforma 9i. Quando são abertos com o *Forms Builder 9i*, em condições normais, os *forms* actualizam os valores de todas as propriedades herdadas da *template*, incluindo o código dos *Triggers*. Os *forms* que se encontram na situação anómala descrita, não herdam essas propriedades, pelo que não são actualizados com as "correcções" feitas para correr na nova plataforma. Nestas circunstâncias, não é sequer possível compilar estes *forms* com o *Forms Builder 9i*.

Apesar de se desconhecer a causa deste problema, encontrou-se uma solução bastante simples. Basta remover a *template* dos *forms*, e adicioná-la novamente. Ao remover a *template*, todos os objectos nela definidos (mesmo aqueles cujas propriedades haviam sido alteradas) são também removidos. Adicionar a *template* novamente faz com que os objectos sejam adicionados de novo ao *form*, mas desta vez com as propriedades no estado correcto (*Inherited Value*). Como, para todos os outros *forms* "normais", cujos objectos herdados da *template* possuem as propriedades no estado correcto, este procedimento não introduz qualquer inconveniente, decidiu-se efectuar esta operação (eliminar a *template* e carregá-la novamente) para todos os *forms* que tivessem propriedades no estado *Overriden Inherited Value* aquando da migração para a nova plataforma. Isto não só iria permitir que os *forms* carregassem correctamente a *template* da plataforma 9i, como também iria corrigir um problema que os *forms* apresentavam já inicialmente na plataforma 6i.

Infelizmente, verificou-se mais tarde que esta solução não está livre de riscos, mais uma vez devido a uma característica anómala dos *forms*, semelhante à descrita anteriormente. Na situação para a qual a utilização da *template* foi idealizada, os *forms* deveriam ter todas as propriedades dos objectos herdados da *template* no estado *Inherited Value*. Foi já referido que

existem alguns *forms* que, por motivos ainda desconhecidos, perderam inadvertidamente a relação com a *template* em todas as propriedades. Existem outros, contudo, em que apenas algumas propriedades foram alterados e se encontram, portanto, no estado *Overriden Inherited Value*. Estas tratam-se de alterações efectuadas pelos programadores propositadamente, para alterar o comportamento *default*, que é herdado da *template*. Naturalmente, esta prática não deveria ser levada a cabo. E, neste caso, a solução descrita atrás (eliminar a *template* e carregá-la novamente) pode ter consequências muito graves: todas as alterações ao código dos *Triggers* herdados da *template* que tenham sido efectuadas são perdidas. E verificou-se que em alguns casos são implementadas funcionalidades importantes nestes *Triggers*.

É necessário ter-se alguma sensibilidade nestes casos, para se averiguar se é seguro recarregar a *template* ao efectuar a migração. Apesar do número de *forms* que se encontram nestas situações anómalas ser reduzido, trata-se de um problema que necessita de atenção.

Correcções ao código PL/SQL

Tal como já foi referido, a migração dos *forms* implica a substituição de determinadas funções e procedimentos PL/SQL por novas versões dos mesmos, já preparadas para a nova plataforma, que se encontram nas novas bibliotecas adicionadas aos *forms*. As funções e procedimentos que não funcionam correctamente na plataforma *web* e para os quais foi já criada uma versão compatível são descritos de seguida.

1. Get_file_name

Esta função não é compatível com a nova plataforma *web* pois, embora esta não origine qualquer erro, tem um efeito que não é o esperado. A sua execução é feita no lado do servidor, pelo que o valor que devolve é referente à máquina servidora, e não à máquina onde o utilizador se encontra. A nova versão da mesma, denominada *cpc_get_file_name,* tem exactamente a mesma assinatura (mesmo número e tipo de argumentos), pelo que a correcção consiste simplesmente em adicionar "*cpc_*" antes de todas as ocorrências de *get_file_name*. O que esta nova versão da função faz é verificar se a execução está a ser feita na plataforma 6i ou 9i. Se for 6i, invoca a função original, *get_file_name*, normalmente. Se a plataforma de execução da aplicação for o *Oracle Forms 9i*, então utiliza as funcionalidades disponibilizadas pelos *JavaBeans* do *DataBlock WEBUTIL* para invocar a execução localmente, na máquina onde o utilizador se encontra.

2. Tool_env

Como os valores devolvidos pela função devem fazer referência à máquina onde o utilizador se encontra, esta não pode ser utilizada normalmente na plataforma *web*, dado que iria devolver dados referentes ao servidor onde a aplicação se encontra alojada. A nova versão desta função, denominada *cpc_tool_env*, recebe os mesmos argumentos que a original, e verifica qual é a plataforma onde a aplicação está a ser executada. Tratando-se da plataforma *Oracle Forms 6i*, invoca a função *tool_env* original. Na plataforma 9i invoca a execução da função localmente através do *WEBUTIL*.

3. Text_io

Trata-se de um *package* que disponibiliza várias funções e métodos para realizar entrada e saída de dados para ficheiros. Foi criada uma nova versão deste *package* cujas funções e

procedimentos são exactamente os mesmos do anterior. A única diferença entre ambos é o nome, que na nova versão é *cpc_text_io*. A correcção das invocações deste *package* consiste em acrescentar "*cpc_*" antes do nome do *package*. A invocação da função *text_io.open_file()* (função chamada *open_file* que se encontra no *package text_io*) seria substituída por *cpc_text_io.open_file()*. As novas funções e procedimentos deste *package* efectuam a mesma verificação descrita nos casos anteriores. Invocam aos métodos homónimos no *package* original se a execução estiver a ser feita na plataforma 6i, ou, no caso da execução via *web*, provocam a execução local através do *WEBUTIL*.

4. Win_api_environment

Trata-se de um *package* que disponibiliza várias funções para aceder a variáveis de ambiente do *Windows* na máquina onde a aplicação é executada. A correcção das invocações a funções e procedimentos deste *package* é igual àquela descrita para *text_io*, sendo que o nome do novo *package* é *cpc_win_api_environment*.

5. Host

Host é um procedimento que se encontra numa biblioteca *built-in* da plataforma *Oracle Forms*, utilizado para efectuar a invocação de comandos de sistema, externos à aplicação. O problema com este procedimento é o facto de efectuar a execução na máquina onde a aplicação está a correr que, na plataforma *web*, é o servidor onde a aplicação está alojada, e não a máquina onde o utilizador se encontra. A utilização desta função para invocar, por exemplo, a calculadora do *Windows*, teria como efeito, em 9i, a execução da calculadora no servidor. A correcção deste procedimento consiste então em invocar a nova versão do mesmo, chamada *cpc_host*, que se encontra localizada num novo *package* chamado *cpc_hosts*. Na prática, devem-se substituir as ocorrências de *host* por *cpc_hosts.cpc_host*. Ambas as versões recebem um único parâmetro, que é a *string* que constitui o comando a ser executado.

Contudo, este procedimento é também utilizado na CPCHS para apresentar *reports*. Isto é feito através da invocação dos comandos *rwrun60* ou *r25run32*, que são duas versões diferentes do *runtime* de *reports* da Oracle. A utilização da nova versão do *host* para invocar os *reports* teria como resultado a execução do *report* localmente, na máquina onde o utilizador se encontra. O inconveniente disto é que seria necessário ter os *reports* instalados nessa máquina, o que vai contra o princípio da nova plataforma, cujo propósito é exactamente não ser necessário instalar nada na máquina cliente. Mais ainda, a plataforma *Oracle Forms 9i* suporta a invocação de *reports* no servidor via *web*, sendo estes apresentados ao utilizador num *browser web*, à semelhança da aplicação propriamente dita. Foi criado então um procedimento distinto para a invocação de *reports*, denominado *abre_report*, localizado num novo *package* chamado *reports_pck*. Portanto, as utilizações da *built-in host* para a invocação de *reports* devem ser substituídas por *reports_pck.abre_report*. Neste caso, a assinatura dos métodos não é a mesma, sendo que *abre_report* recebe quatro parâmetros, e não um só. É necessário separar a *string* passada como parâmetro ao *host* original de modo a obter-se os vários parâmetros necessários à invocação da nova versão, *abre_report*.

Esta nova versão, compatível com ambas as plataformas, invoca um *host* normalmente, concatenando os vários parâmetros que recebe separadamente, quando verifica que a plataforma de execução é o *Oracle Forms 6i*. Na plataforma *web*, e ao contrário dos casos até

aqui descritos, invoca a execução do *report* no lado do servidor, apresentando-o ao utilizador no *browser web*[22].

Resumindo, os *hosts* utilizados para abrir *reports* devem ser substituídos por *reports_pck.abre_report*, sendo necessário construir os parâmetros para este procedimento a partir do único parâmetro do procedimento *host*. Os *hosts* usados para a invocação de outros comandos de sistema, devem ser substituídos por *cpc_hosts.cpc_host*, sendo que neste caso não é necessário alterar os parâmetros.

Verificações no código PL/SQL

Os procedimentos e funções referidos nas secções anteriores não são os únicos que não funcionam correctamente na plataforma *Oracle Forms 9i*. Existem outras incompatibilidades para as quais não foi ainda criada uma versão substituta. Estas funcionalidades incompatíveis coma nova plataforma são descritas nesta secção.

Em *Oracle Forms 6i*, é possível criar, ao nível dos *Items*, *Triggers* do tipo ***when-mouse-enter*** e ***when-mouse-leave***. Estes *Triggers*, invocados quando o ponteiro do rato entra e sai da área de um *Item*, respectivamente, já não são suportados na plataforma *Oracle Forms 9i*. De acordo com a própria Oracle, a utilização destes *Triggers* produziria um tráfego excessivo entre o cliente e o servidor de cada vez que o utilizador movimentasse o ponteiro do rato sobre os *Items* que os definem. Apesar de não ser emitido qualquer aviso ou erro de compilação quando estes *Triggers* são utilizados nos *forms*, estes simplesmente não são invocados na plataforma *web*. Qualquer funcionalidade implementada aí implementada é, perdida aquando da migração para a *web*. Na CPCHS ainda não foi criada uma forma automática de solucionar este problema, pelo que os *Forms* que utilizam estes *Triggers* devem ser vistos caso a caso. Durante o processo de migração deve-se, portanto, registar a ocorrência destes *Triggers* nos *forms*, para mais tarde serem analisados.

Outra funcionalidade que origina problemas de execução na plataforma 9i é a utilização de ***Timers***. Tratam-se de componentes que, depois de iniciados e enquanto não forem terminados, invocam um procedimento PL/SQL repetidamente, com intervalos de tempo que podem ser configuráveis. Também por questões de eficiência, os *Timers* não devem ser utilizados na plataforma *web* (um *Timer* programado com um período de 1 segundo iria originar um tráfego constante entre cada um dos clientes que estivesse a executar a aplicação e o servidor aplicacional). Como também não existe uma forma automática de substituir a utilização de *Timers*, esta deve ser registada para posterior análise. A utilização destes componentes é detectada pela invocação do procedimento *create_timer()* no código PL/SQL.

A utilização de controlos ***ActiveX***, funcionalidade suportada apenas em *Oracle Forms 6i*, não é possível na plataforma *web*. A única forma de substituir estes componentes é desenvolver componentes substitutos numa outra tecnologia suportada na *web*, nomeadamente *JavaBeans*. A utilização de controlos *ActiveX*, denunciada pela existência da função *dispatch_event()* no código PL/SQL da aplicação, deve também ser registada, para análise posterior.

Para além do *package win_api_environment*, existem outros *packages* que disponibilizam o acesso a certas funcionalidades do *Windows*, nomeadamente o acesso ao *registry*, ao sistema de ficheiros, etc. Ainda não foram criadas novas versões destes *packages*, cujo nome é sempre

---

[22] Na versão actual das aplicações da CPCHS, isto ainda não acontece, porque ainda não se procedeu à configuração do servidor aplicacional para disponibilizar *reports*.

do tipo ***win_api****, dado que são poucos os *forms* que os utilizam. A sua utilização deve ser, para já, apenas registada.

Existe ainda outro procedimento cuja utilização não é suportada na *web*, e para o qual não foi ainda criado um substituto. Trata-se do procedimento ***run_product***, cuja utilização deve ser registada. Tendo em conta que são muito poucos os *forms* que o utilizam, é mais provável que estes *forms* sejam modificados para não utilizarem este procedimento, ao invés de se criar uma nova versão do mesmo compatível com a plataforma *web*.

Compilação em Oracle Forms 6i

Depois de se efectuarem todas as correcções e verificações atrás descritas, o passo seguinte no processo de migração é guardar as alterações efectuadas ao ficheiro *fmb* do *form* (as alterações devem ter sido efectuadas com o *Forms Builder 6i*). Depois, disso, o *form* deve ser compilado, criando-se um ficheiro *fmx* compatível com a plataforma 6i.

Compilação em Oracle Forms 9i

Segue-se a compilação do ficheiro *fmb* com um compilador da plataforma *Oracle Forms 9i*. Apesar de o ficheiro *fmb* ter sido criado no *Forms Builder 6i*, é possível abri-lo com o *Forms Builder 9i*, e compilá-lo aí. Deve ter-se o cuidado de nunca salvar o *form* no *Forms Builder 9i*, sob pena de não voltar a ser possível abri-lo no *Forms Builder 6i*, e continuar o desenvolvimento para a plataforma 6i.

Assim que se abre o ficheiro *fmb* com o *Forms Builder 9i*, a nova versão da *template* é carregada, e o *form* passa a conter alguns novos constituintes que não existiam antes, nomeadamente dois novos *DataBlocks*: *WEBUTIL e CPCHSBEANS*. Estes são colocados automaticamente no topo da lista de *DataBlocks* do *form*. Antes de se efectuar a compilação, estes dois blocos devem ser movidos para o fim da lista, pois o seu posicionamento à frente de outros blocos pode afectar o comportamento da aplicação (quando um *form* é executado, torna visível o primeiro *Item* do primeiro *DataBlock* da lista, caso não esteja especificado qual é o *DataBlock* cujos *Items* obtêm o foco inicial).

Criação de Log Files

Todas as operações realizadas durante o processo de migração devem ser registadas num ficheiro de *log*. Devem ficar registados os nomes dos *forms* que já foram processados; a indicação de que foram compilados com sucesso ou não em ambas as plataformas; para cada *form*, devem registar-se as alterações efectuadas; e ainda, registar as ocorrências dos componentes não suportados.

Durante a fase do estágio de introdução à migração para a plataforma *web*, foram convertidos vários *forms*, manualmente, tendo sido criado um *log* numa folha de cálculo do Excel. Este *log,* contudo, não é muito completo, sendo que se decidiu criar um novo, num formato mais completo, bastante mais exaustivo, que só faria sentido ser criado por uma aplicação de conversão automática. Contudo, por questões de compatibilidade com o trabalho manual previamente realizado, decidiu-se que a aplicação de conversão automática deveria também criar uma outra versão do ficheiro de *log* (para além do *log* principal, o mais completo) compatível com aquele previamente elaborado.

ANEXO C: O Novo *Look-And-Feel* do *Processo Clínico Electrónico*

Neste anexo, serão apresentadas as novas directivas a seguir no desenho das interfaces gráficas. Não se pretendia um aspecto visual completamente novo, mas sim alterações ao já existente. Isto porque a criação de interfaces gráficas completamente novas implicaria a necessidade de, novamente, dar formação aos utilizadores do *Processo Clínico Electrónico*.

A descrição do novo estilo das interfaces é apresentada através de exemplos, com bastantes ilustrações. Optou-se por esta forma de apresentação dado que uma descrição unicamente textual teria que ser muito extensa e exaustiva para cobrir todos os aspectos do novo estilo. Começa-se por utilizar um *form* chamado *GHPC7510* do *Processo Clínico Electrónico* (figs.27, 28 e 29). Este foi criado no âmbito deste estágio, para colmatar uma lacuna que existia na aplicação. Este *form* tem o objectivo de permitir gerir a entrega de relatórios de exames. A elaboração de relatórios é uma funcionalidade muito bem suportada pela nova versão do *Processo Clínico* e, para completar todo o circuito relacionado com os relatórios, faltava apenas a gestão de entregas.

O segundo *form* utilizado para descrever as novas directivas para o desenho de interfaces do *Processo Clínico* é o *form* de *Atendimento em Ambulatório*. Este *form* não pertence ao *Processo Clínico*, mas sim à *Gestão Hospitalar*. No entanto, foi utilizado na criação do novo *layout* por se tratar de um *form* relativamente complexo a nível gráfico, possuindo vários tipos de componentes para os quais era necessário definir as novas directivas. Note-se, no entanto, que esta versão do *form* aqui apresentada (fig.30) não está de facto a ser utilizada na CPCHS, tendo sido criada apenas para este propósito[23].

As figuras apresentadas nesta secção incluem vários notas acerca do desenho das novas interfaces. As notas que contêm texto como *r25g50b75* indicam a cor do componente para o qual apontam, em formato RGB (*Red, Green, Blue*).

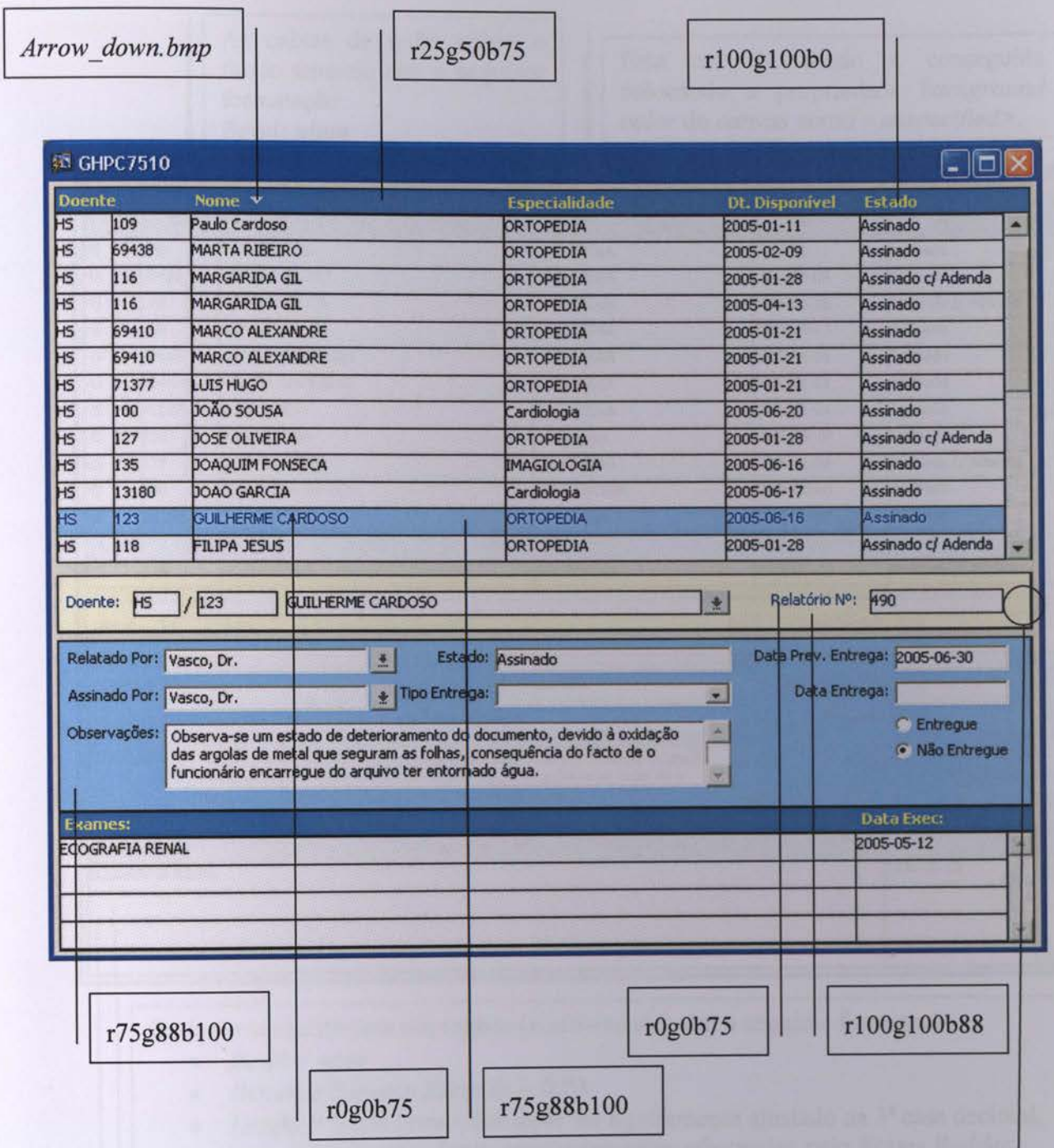
Na fig.27, a nota com o texto *Arrow_down.bmp* indica o nome da imagem que deve ser utilizada nos *Items* do tipo *Image Item* que constituem os botões do cabeçalho das tabelas onde o utilizador pode *clicar* para ordenar os registos por qualquer uma das colunas. A imagem apresentada nestes *Items* deve mudar sempre que o utilizador *clica* neles, alternando entre *Arrow_down.bmp* e *Arrow_up.bmp*. Note-se que, em qualquer instante, apenas uma das colunas possui a seta de ordenação desenhada.

As tabelas (construídas a partir de *DataBlocks* que apresentam vários registos) devem agora ter sempre o registo actual realçado, com o aspecto indicado na fig.27. Isto nem sempre acontecia anteriormente e, quando acontecia, as cores e o estilo do realce utilizados não eram uniformes.

Sugere-se o agrupamento dos *Items* em caixas sempre que isto faça sentido, principalmente nos *forms* constituídos por muitos componentes visuais, de forma a facilitar a leitura da interface por parte do utilizador.

---

[23] Foi já criada uma nova versão definitiva deste *form*, mas seguindo o novo *layout* da *Gestão Hospitalar*, e não o do *Processo Clínico*. O novo *layout* da *Gestão Hospitalar* foi desenvolvido com base no do *Processo Clínico*. No entanto, utiliza um esquema de cores diferente, limitando-se quase exclusivamente à utilização de diferentes tons de azul.



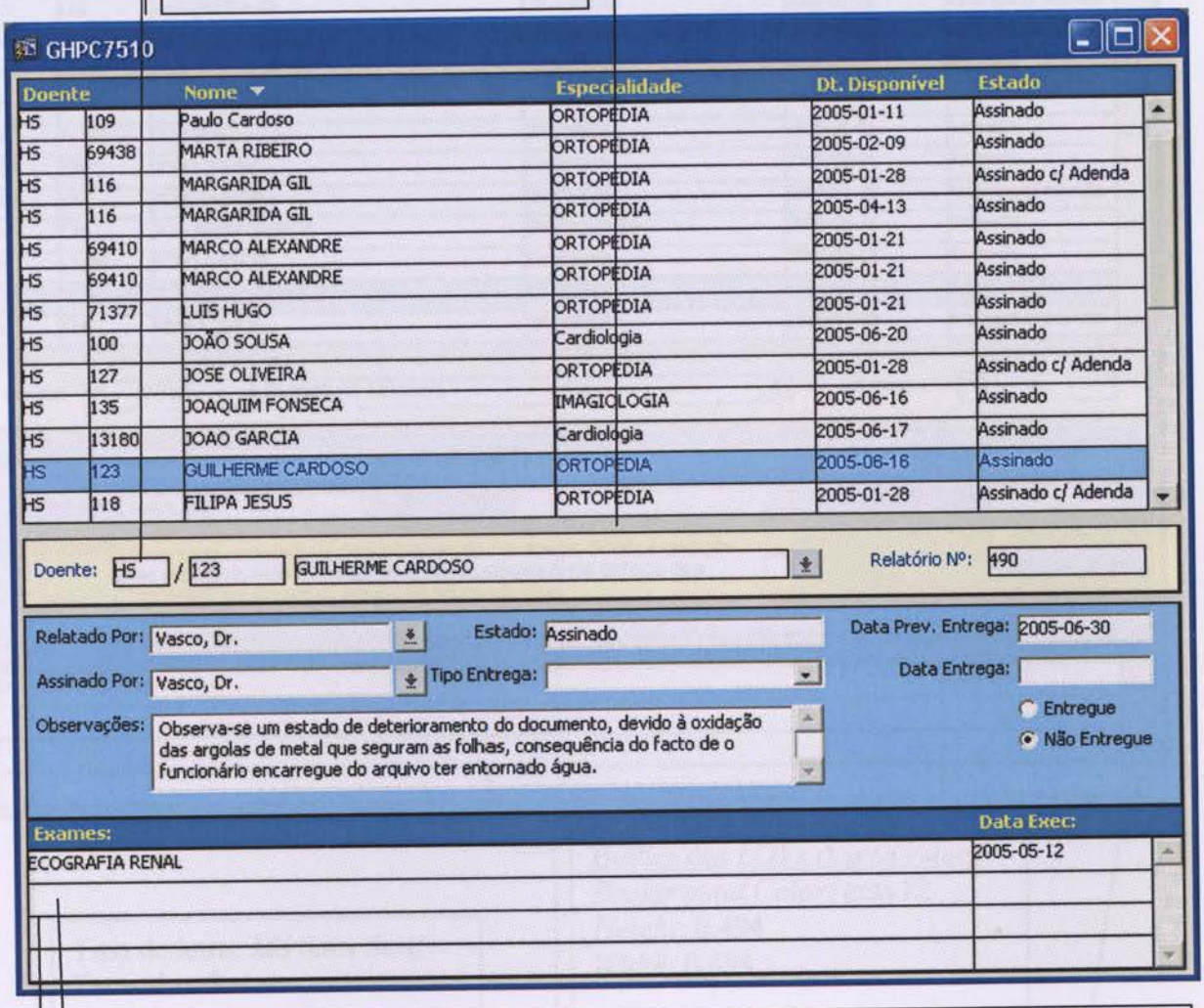
É conseguido um leve efeito tridimensional colocando um rectângulo preto por baixo do rectângulo que serve de fundo ao bloco, ligeiramente desviado para a direita e para baixo.

NOTA: o rectângulo preto deve ficar visível a toda a volta do rectângulo do bloco, e não apenas nas bordas direita e inferior. No entanto, nestas últimas (direita e inferior), deve ver-se uma maior porção do fundo preto.

***Fig.27** – Algumas características do novo estilo das interfaces gráficas do Processo Clínico.*

As caixas de texto sobre o fundo amarelo têm a seguinte formatação:
Bevel: plain

Esta cor de fundo é conseguida colocando a propriedade *background color* do *canvas* como <*unspecified*>.



Os *Items* com mais que um registo (*multi-record*) têm a seguinte formatação:

- *Bevel = none*
- *Distance Between Records* = 0.03
- *Height* = 0.486 (este valor deve ser ligeiramente ajustado na 3ª casa decimal, para evitar que possíveis arredondamentos efectuados pelo *Forms Builder* façam com que a distância entre alguns registos seja superior a outros)

Os *Items* com vários registos têm um rectângulo preto por baixo, que fica visível através dos espaços deixados entre os registos; a borda à volta da tabela com os vários registos é também, na verdade, o rectângulo do fundo, que é ligeiramente maior que os items que contêm os registos;
Aqui, nas tabelas, o fundo preto **não deve** originar o efeito tridimensional dos outros blocos;

***Fig.28** – Algumas características do novo estilo das interfaces gráficas do Processo Clínico.*

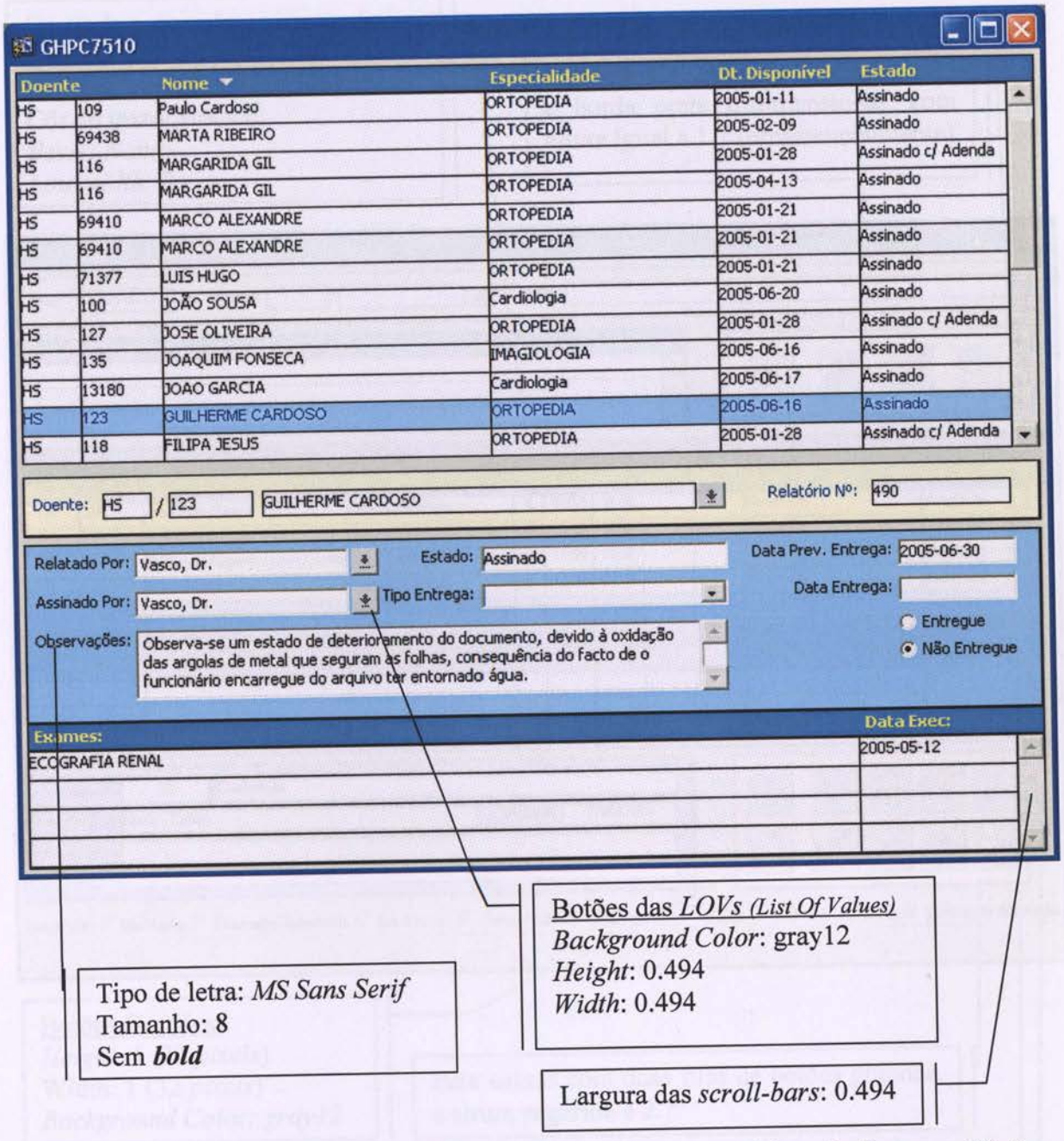
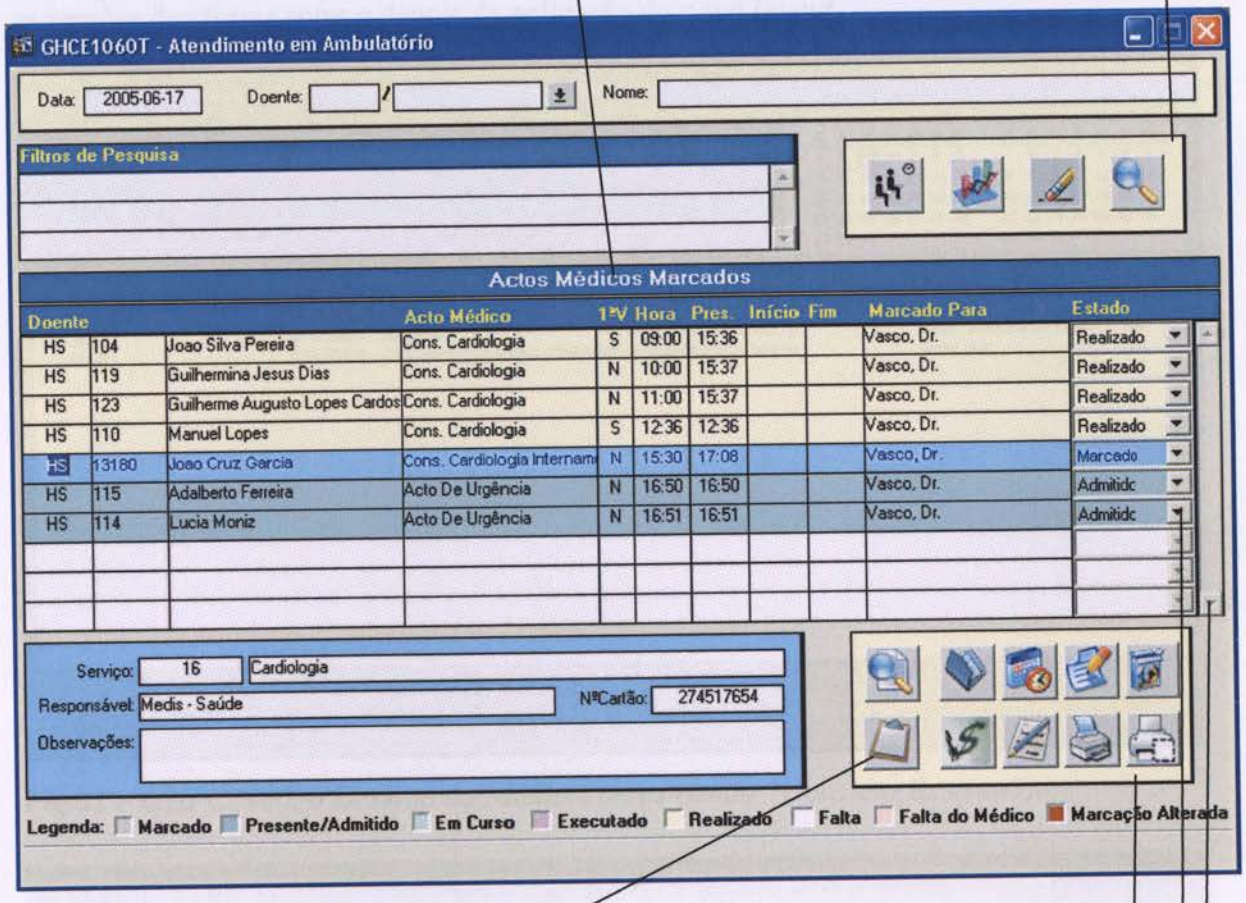


Fig.29 *– Algumas características do novo estilo das interfaces gráficas do Processo Clínico.*

Outro aspecto que precisa de ser uniformizado é a formatação dos botões (*Items* do tipo *Push Button*). O tamanho dos botões deve ser 16x16 (botões pequenos), ou 32x32 (botões grandes), e a cor de fundo deve ser sempre *gray12* (uma cor definida por defeito no *Forms Builder*). Sempre que for possível, os botões devem aparecer dentro de caixas (fig.30). Se o número de botões não justificar a criação de uma caixa própria, ou sempre que se julgue conveniente, os botões devem ser colocados numa caixa juntamente com outros *Items* de outro tipo, desde que a associação a esses *Items* faça sentido. No *form GHPC7510*, por exemplo, são colocados ao lado de algumas caixas de texto botões que abrem uma janela que apresenta uma lista de valores que podem ser seleccionados. Faz sentido associar o botão à caixa de texto para a qual devolve os valores.

Os títulos, quando existem, devem aparecer em caixas;
Background color: r25g50b75;
Côr do texto: Branco;
Bevel: None;
Line width: 0;

Os botões grandes (32 x 32) devem aparecer em caixas de fundo amarelo e borda preta tridimensional, com altura igual a 1.7 (preferencialmente)



Botões Grandes;
Height: 1 (32 *pixels*)
Width: 1 (32 *pixels*)
Background Color: gray12

Para caixas com duas filas de botões grandes, a altura sugerida é 2.7

ATENÇÃO
Nas tabelas, as colunas do tipo *PopList* (*List Item*) têm formatação diferente dos outros *Items*:

- Não se define propriedade *Bevel* (esta não existe para estes *Items*);
- *Distance Between Records* = 0
- *Height*: deve ser ajustada para que a coluna fique com a mesma altura que as outras (aproximadamente 0.5)

A altura da *scroll-bar* das tabelas deve ser igual à soma das alturas dos registos mais o espaçamento entre eles, não incluindo o cabeçalho da tabela.

Fig.30 *– Algumas características do novo estilo das interfaces gráficas do Processo Clínico.*

Ainda em relação aos botões, os ícones utilizados estão a ser gradualmente substituídos por outros mais modernos. As imagens da secção seguintes permitem comparar os ícones antigos com os novos.

Exemplos

Nesta secção são apresentados alguns ecrãs do *Processo Clínico,* sendo colocadas lado a lado as versões dos *forms* antes e depois da aplicação do novo *layout*.

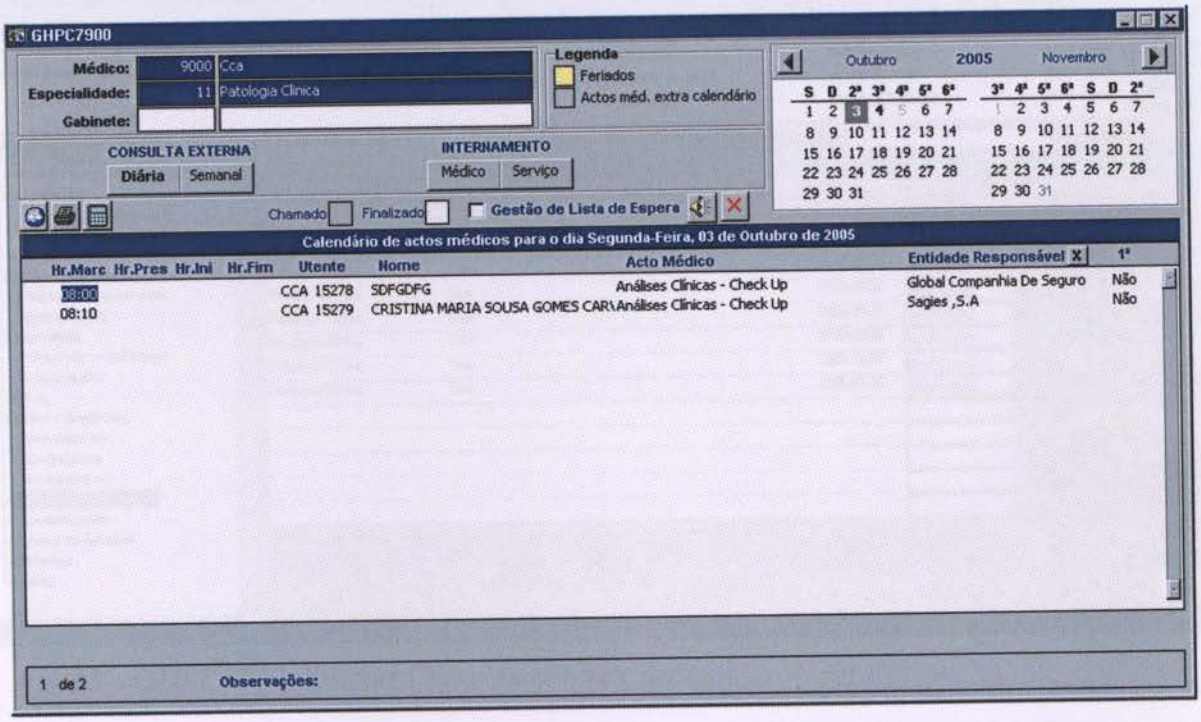


Fig.31 *– GHPC7900, o Desktop do Médico, antes (modo "Consulta Externa").*

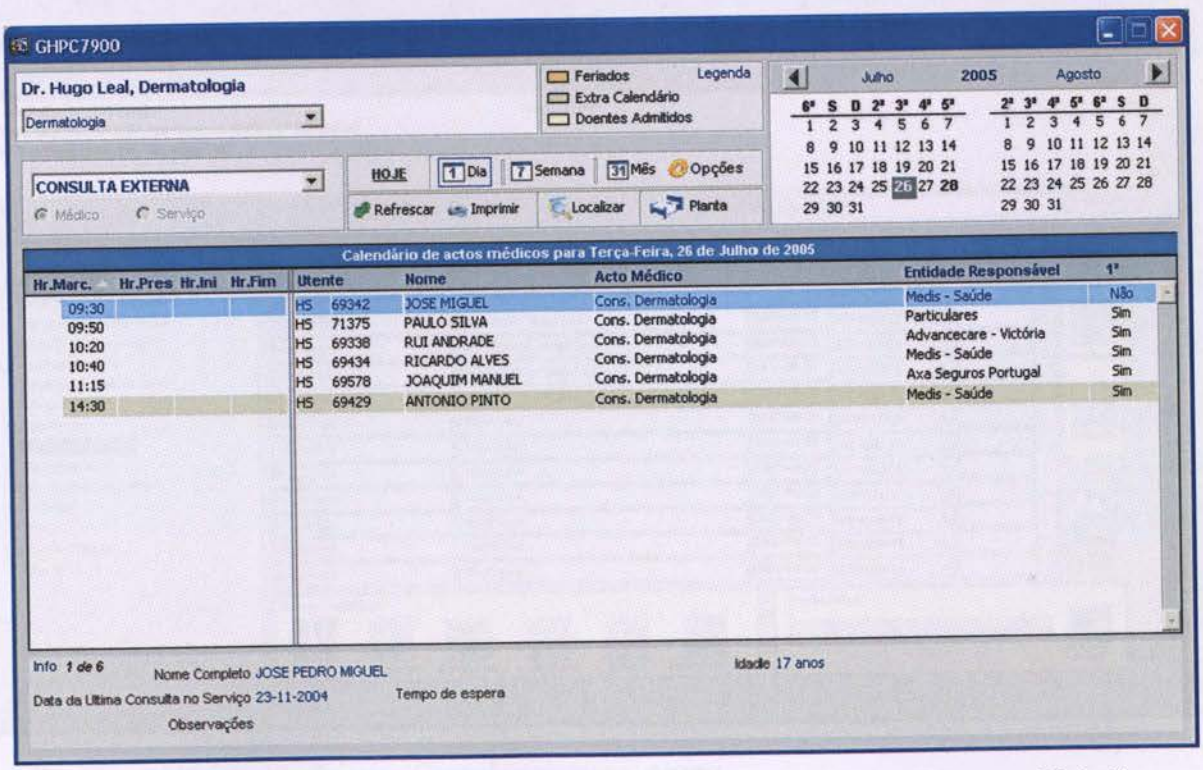


Fig.32 *– GHPC7900, o Desktop do Médico, agora (modo "Consulta Externa, diário").*

Os principais *forms* do *Processo Clínico* são o *GHPC7900*, que consiste no *desktop* do médico, e o *GHPC8000*, acessível a partir do primeiro, onde é consultada e registada toda a informação clínica. Ao contrário da maioria dos outros *forms*, estes dois não são estáticos. A interface que apresentam vai mudando, de acordo com as opções que o utilizador seleccionar.

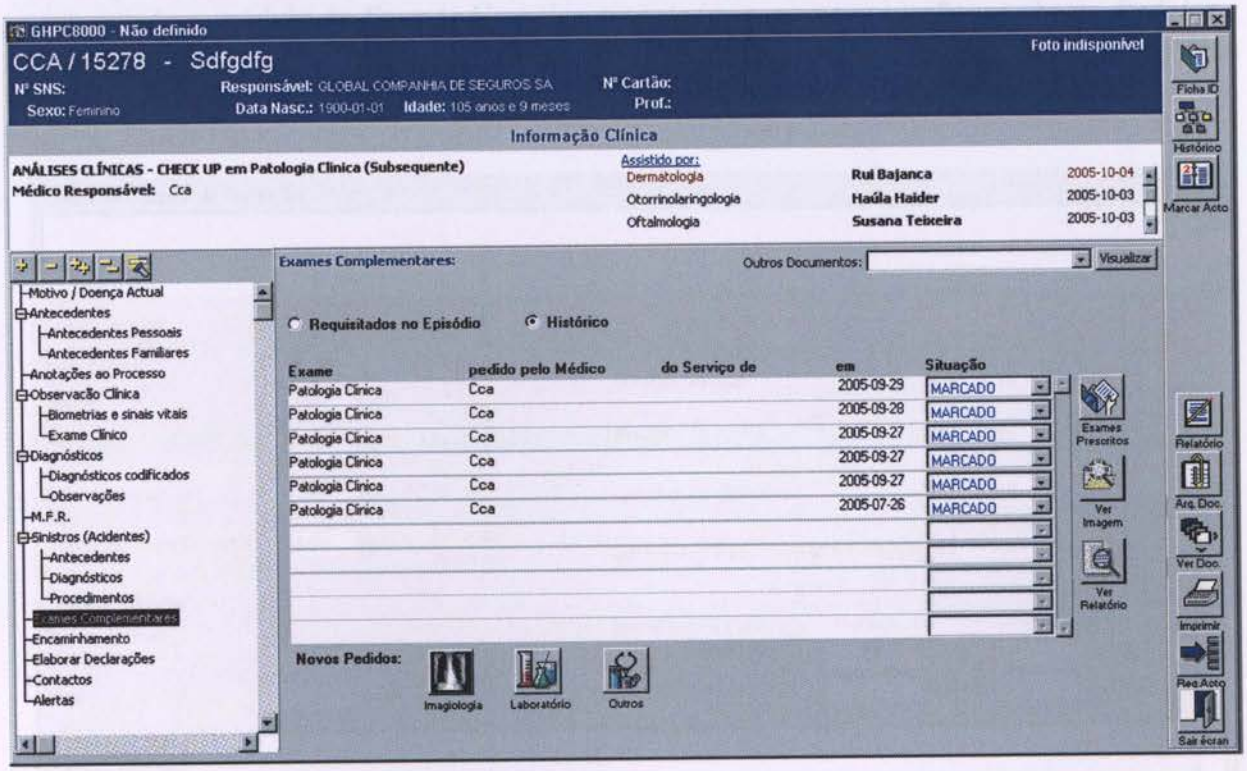


Fig.33 *– GHPC8000, Exames Complementares, antes.*

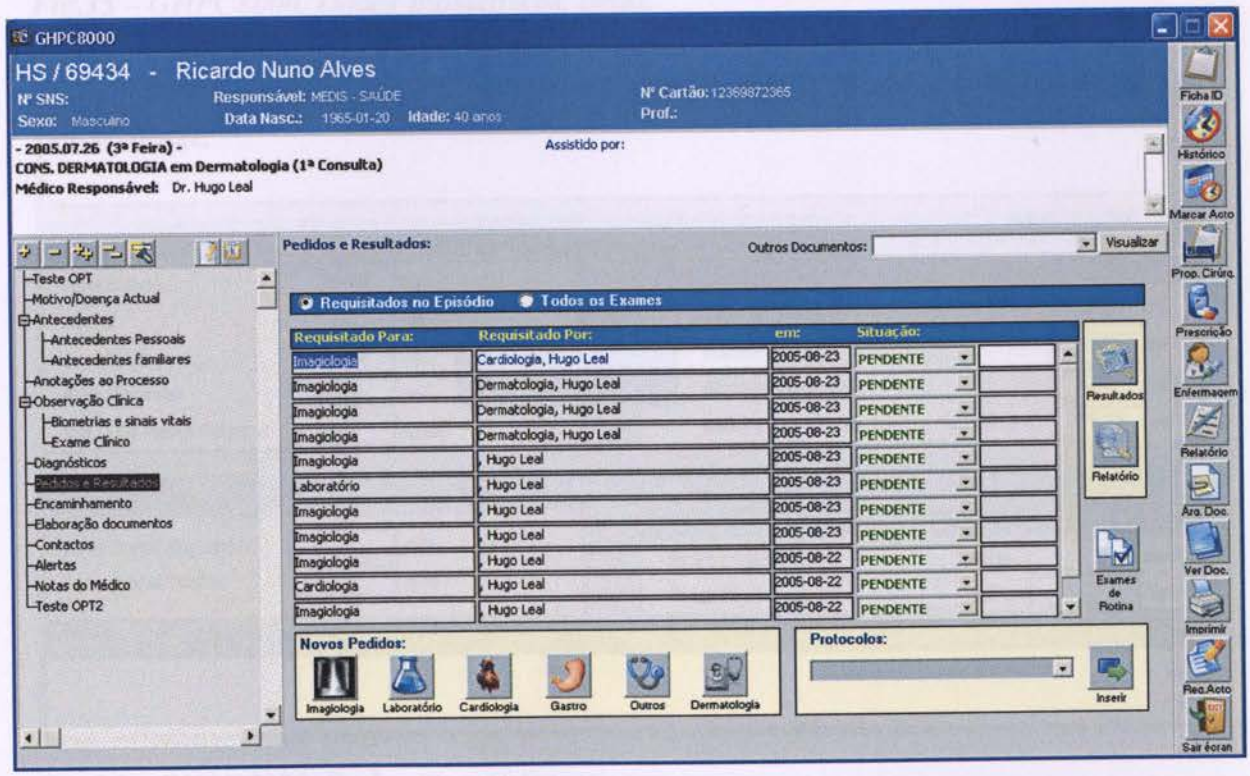


Fig.34 *– GHPC8000, Exames Complementares, agora.*

As figuras 31 e 32 permitem comparar as versões do *GHPC7900* antes e depois da realização do estágio. Este foi um dos *forms* que sofreu mais alterações, não só no aspecto gráfico, mas também funcional. Nas imagens, é visível o calendário no modo diário da *Consulta Externa*.

Nas figs.33 e 34, é possível comparar as duas versões do *GHPC8000*. Aqui, está a ser apresentado o módulo de *Exames Complementares* (que, na nova versão, se chama *Pedidos e Resultados*). As figs.35 e 36 apresentam o módulo dos *Dados Biométricos*, que aparece também integrado no *GHPC8000* (nas imagens, contudo, só é visível a área do módulo).

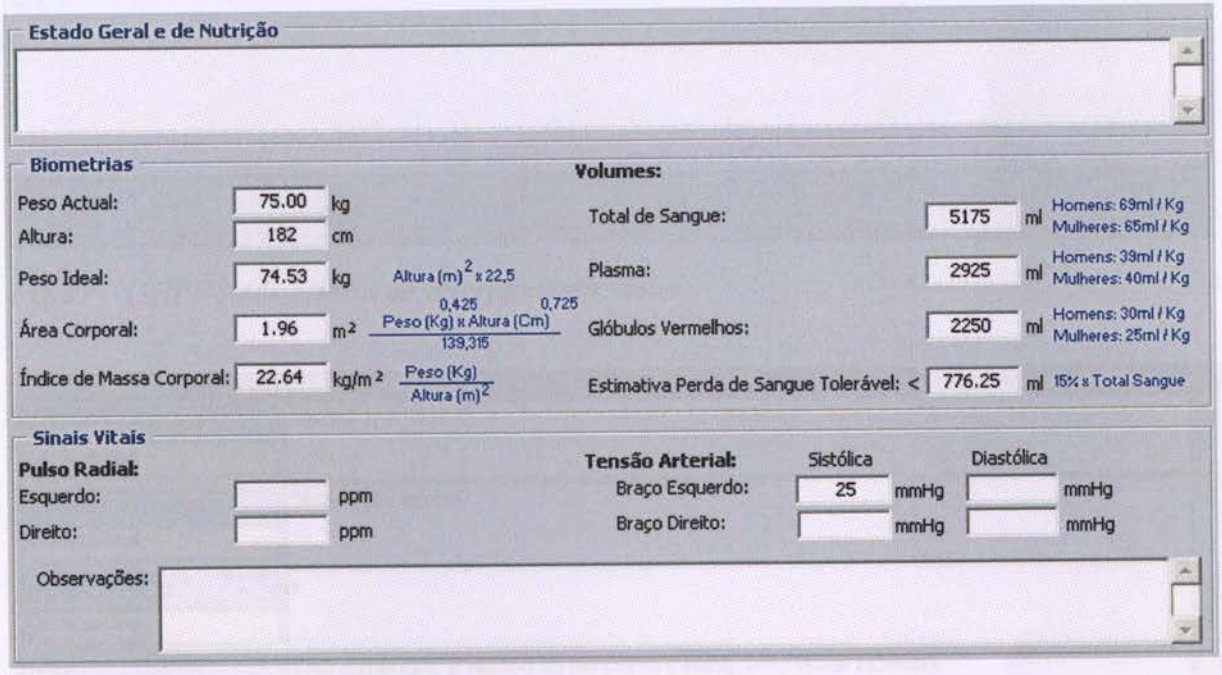


Fig.35 *– GHPC8000, Dados Biométricos, antes.*

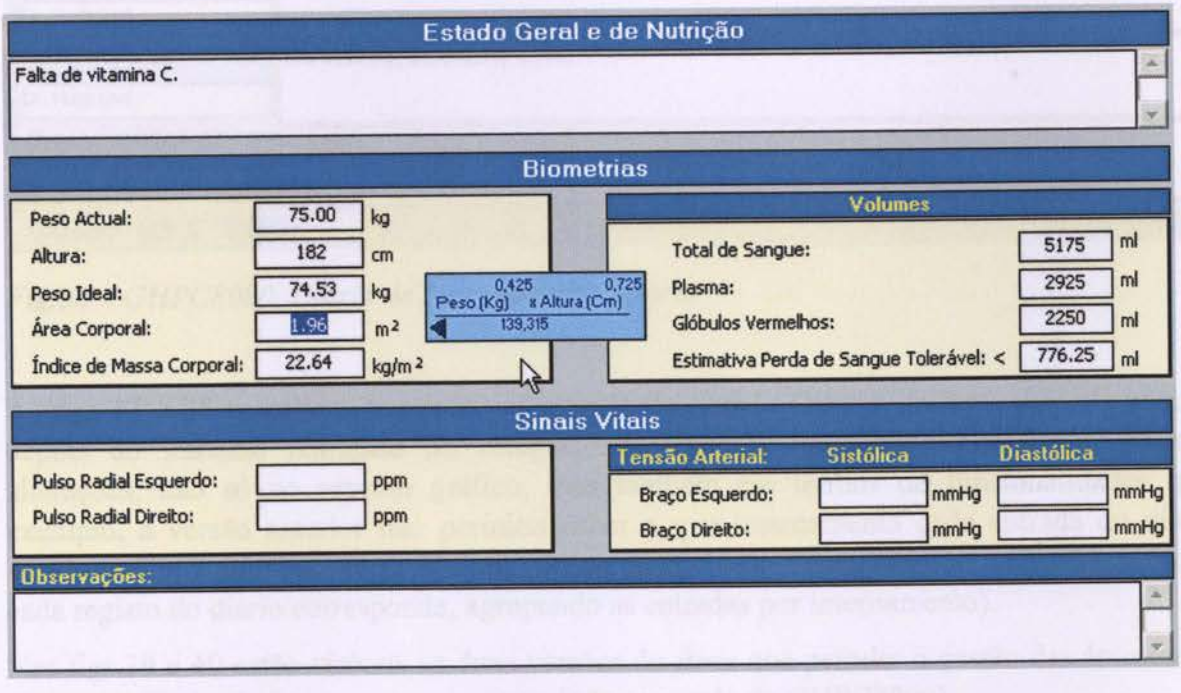


Fig.36 *– GHPC8000, Dados Biométricos, agora.*

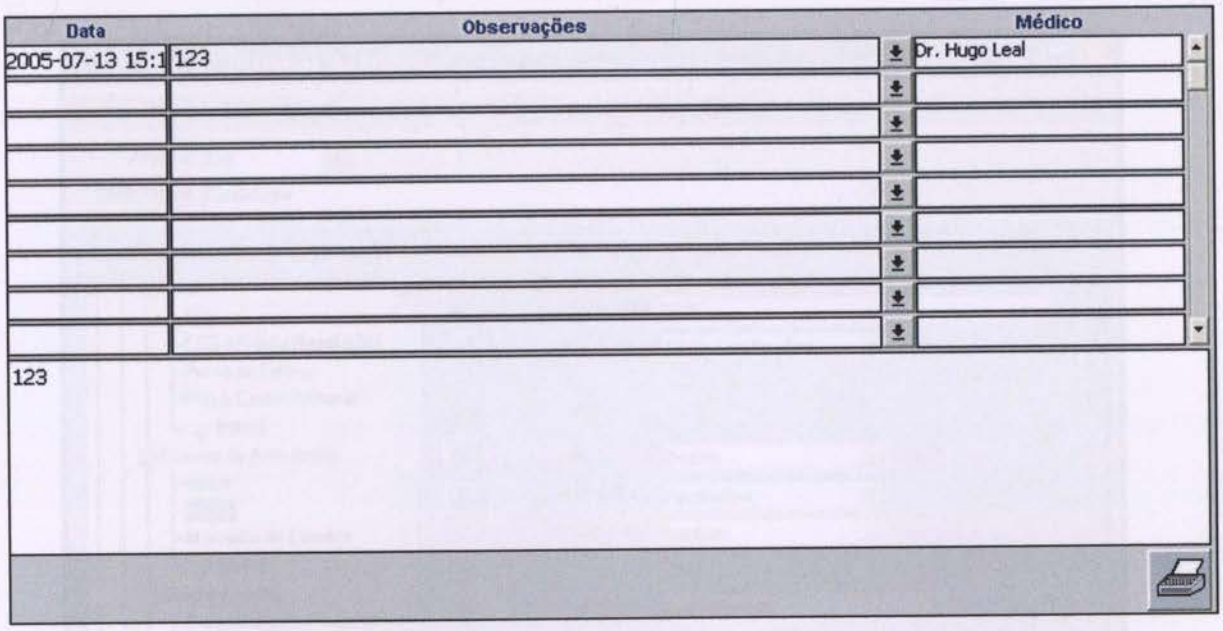


Fig.37 *– GHPC8000, Diário de Internamento, antes.*

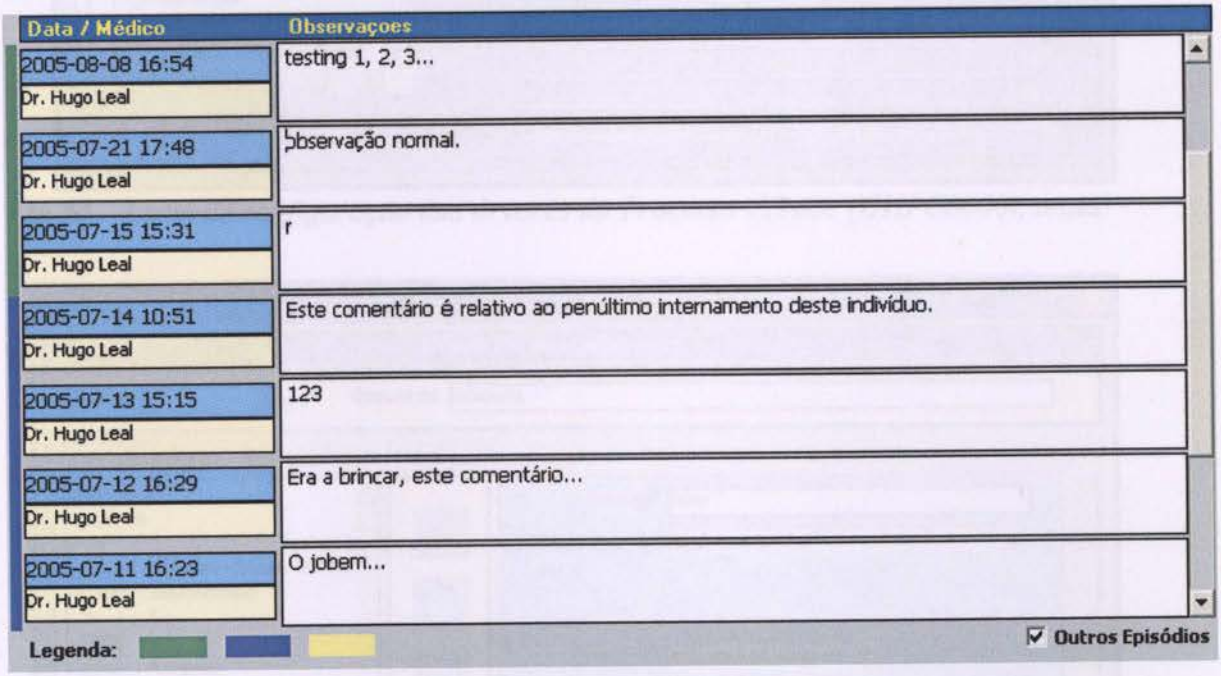


Fig.38 *– GHPC8000, Diário de Internamento, agora.*

As figs.37 e 38 permitem comparar as versões do *Diário Clínico de Internamento* antes e depois do trabalho realizado no estágio. Este módulo do *GHPC8000* sofreu grandes alterações, não só no aspecto gráfico, mas também em termos de funcionalidades (por exemplo, a versão anterior não permitia saber a que internamento cada entrada do diário correspondia; a nova versão utiliza um código de cores para identificar o internamento a que cada registo do diário corresponde, agrupando as entradas por internamento).

Nas figs.39 e 40 estão visíveis as duas versões do *form* que permite a gestão das árvores do *Processo Clínico* (a árvore que aparece no lado esquerdo do GHPC8000).

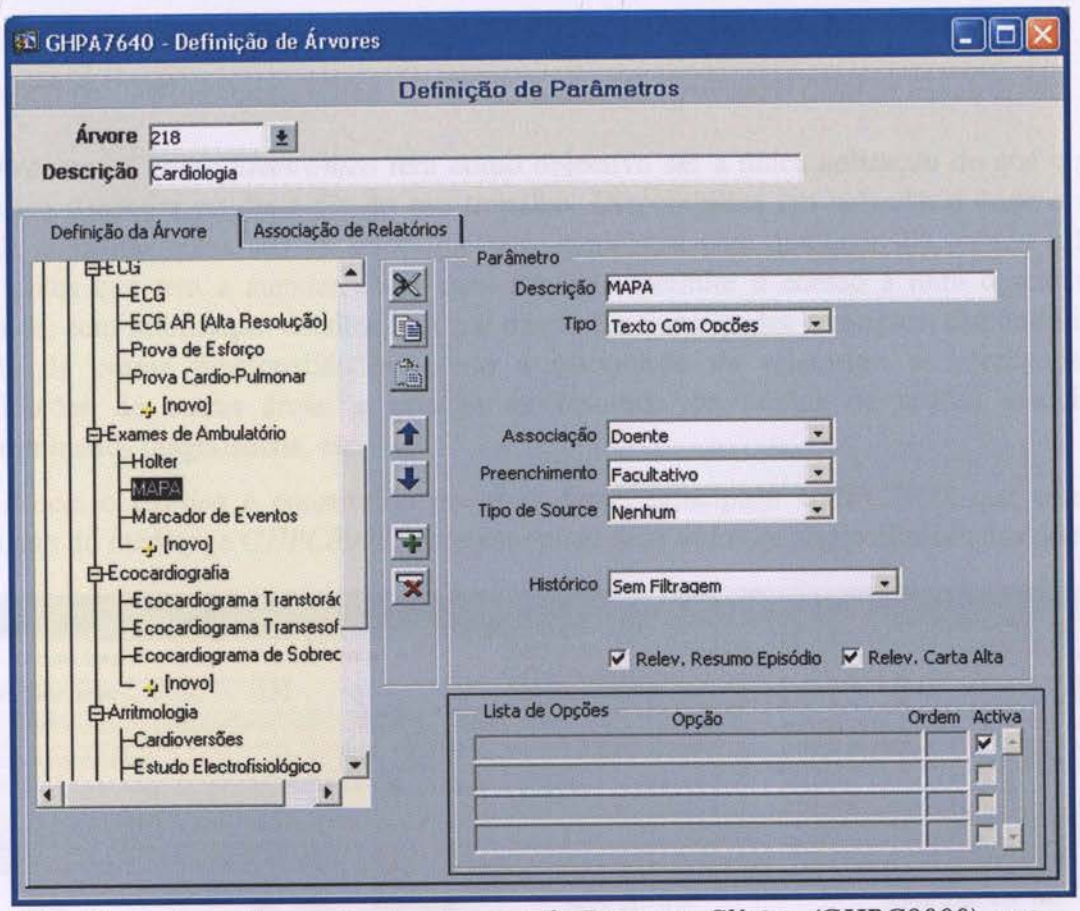


Fig.39 *– Form de configuração das árvores do Processo Clínico (GHPC8000), antes.*

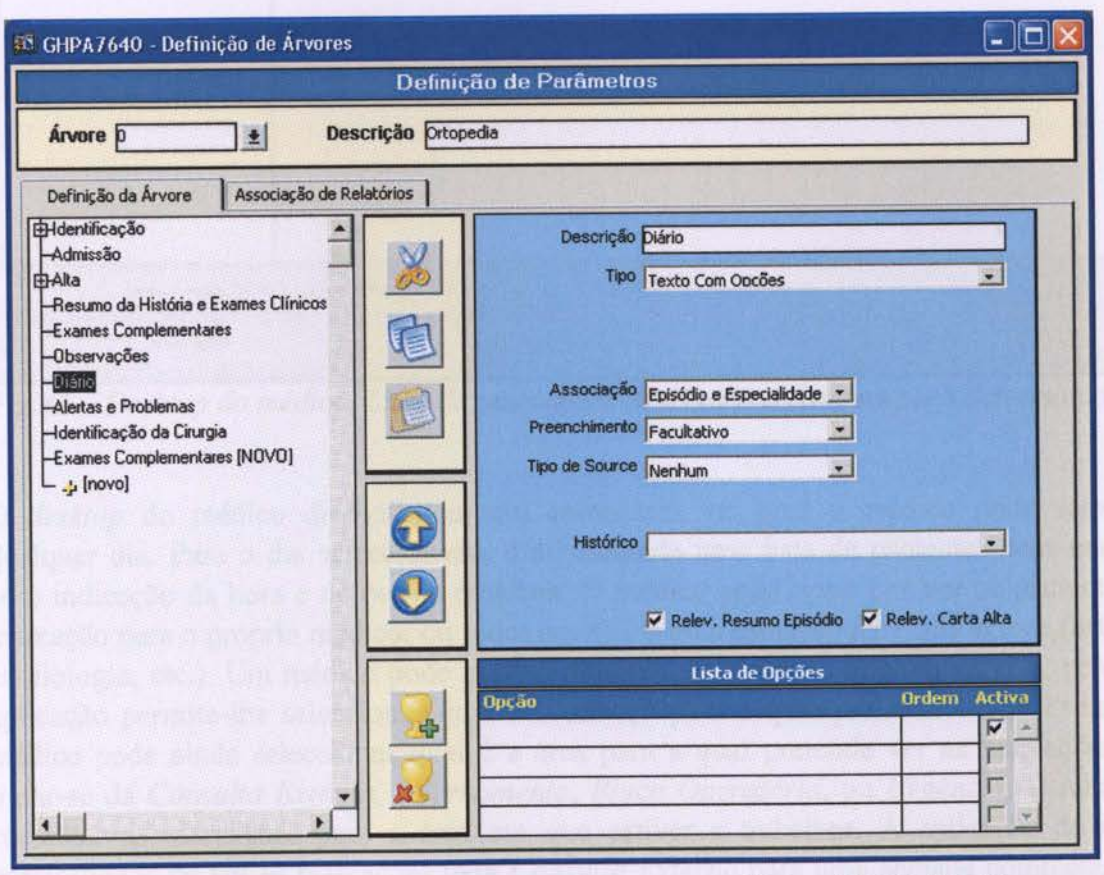


Fig.40 *– Form de configuração das árvores do Processo Clínico (GHPC8000), agora.*

ANEXO D: Funcionalidades da nova versão do *Processo Clínico Electrónico*

O *Processo Clínico Electrónico* tem como objectivo ser a única aplicação de que o pessoal médico necessita no dia-a-dia do seu trabalho. Disponibiliza um calendário onde o médico pode consultar quais são os actos que tem agendados para qualquer dia, vendo a lista de pacientes que tem a atender. Para cada paciente, permite o acesso a todo o seu historial clínico, com informação detalhada. A par destas funcionalidades principais, disponibiliza uma série de outras funcionalidades, como a elaboração de relatórios, a interligação com aplicações de outras áreas, a criação de resumos, protocolos de acções associados a determinados diagnósticos, etc.

O *Processo Clínico* é constituído por dois *forms* principais: *GHPC7900*, que constitui o *desktop* do médico, e *GHPC8000*, onde este pode gerir todos os dados clínicos dos doentes.

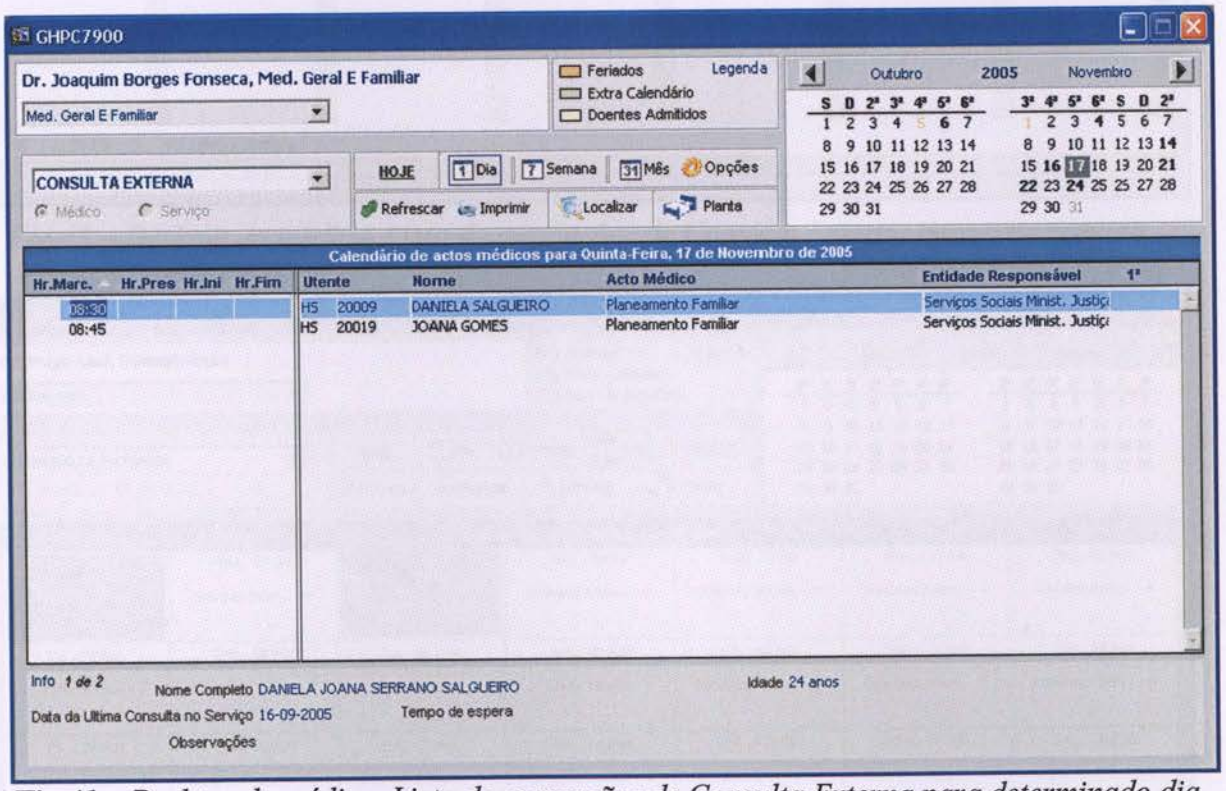


***Fig.41** – Desktop do médico. Lista de marcações de Consulta Externa para determinado dia.*

O *desktop* do médico disponibiliza um calendário, no qual o médico pode seleccionar qualquer dia. Para o dia seleccionado, é apresentada uma lista de pacientes com marcação, com indicação da hora e de outros detalhes. O médico pode optar por ver os pacientes com marcação para o próprio médico, ou todos aqueles para a especialidade que exerce (ortopedia, cardiologia, etc.). Um médico pode exercer funções em mais do que um serviço, pelo que a aplicação permite-lhe seleccionar qual é o serviço para o qual pretende ver informação. O médico pode ainda seleccionar qual é a área para a qual pretende ver as marcações. Pode tratar-se da *Consulta Externa*, *Internamento*, *Bloco Operatório*, ou *Urgência*. O médico irá consultar as marcações para a área em que estiver a trabalhar. A aplicação dá ainda a possibilidade de ver as marcações para *Consulta Externa* para uma semana completa, ou até para um mês inteiro, para além de permitir a consulta para um dia de cada vez.

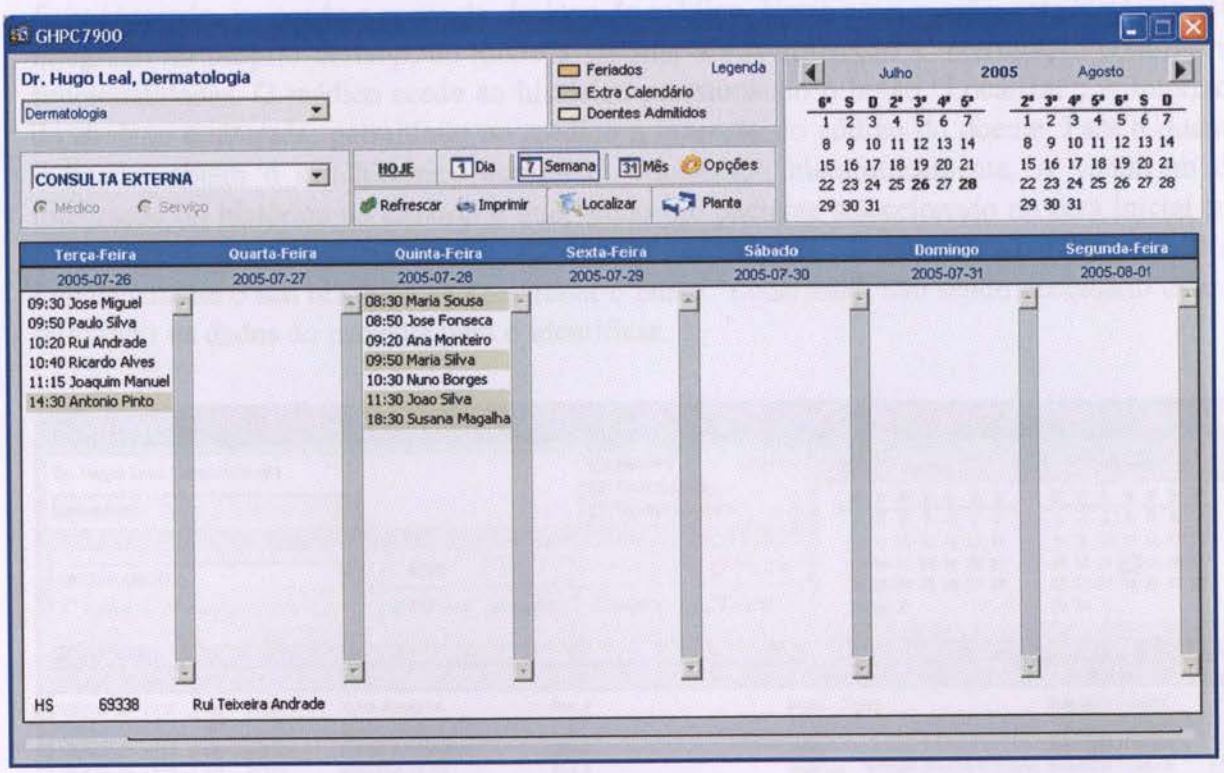


***Fig.42** – Desktop do médico. Lista de marcações de Consulta Externa para uma semana.*

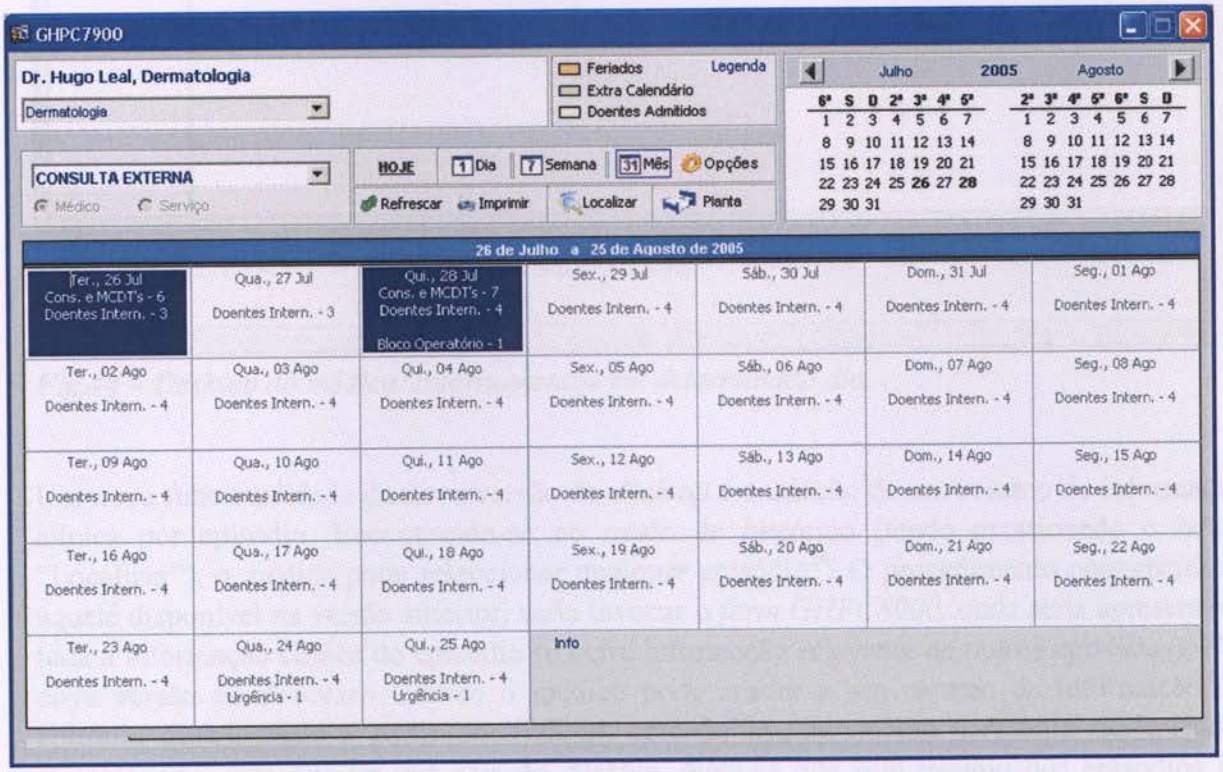


***Fig.43** – Desktop do médico. Marcações de Consulta Externa para um mês.*

Estas são as funcionalidades principais do *desktop* do médico. Existem outras, contudo, que facilitam o trabalho do médico. Seleccionando um paciente a partir do *desktop* (um para o qual exista uma marcação no dia seleccionado), o médico pode aceder a todo o histórico de episódios desse paciente. Esta funcionalidade existia, na versão anterior, sob a forma de um

form separado, invocado a partir do *desktop* do médico. Nesta nova versão, este histórico está integrado no próprio *desktop*, de modo a facilitar a sua utilização e interligação com outras funcionalidades. O médico acede ao histórico pressionando o botão "Localizar". A interface do *desktop* é alterada, permitindo ao médico a inserção do código do doente. Para o doente indicado, obtém o seu histórico completo, organizado hierarquicamente. A vantagem da integração do histórico no *desktop* é que, tendo um paciente seleccionado no ecrã inicial (na *Consulta Externa, Internamento, Bloco Operatório ou Urgência*), o médico obtém imediatamente o seu histórico ao pressionar o botão "Localizar", não sendo necessário estar a introduzir os dados do paciente para o identificar.

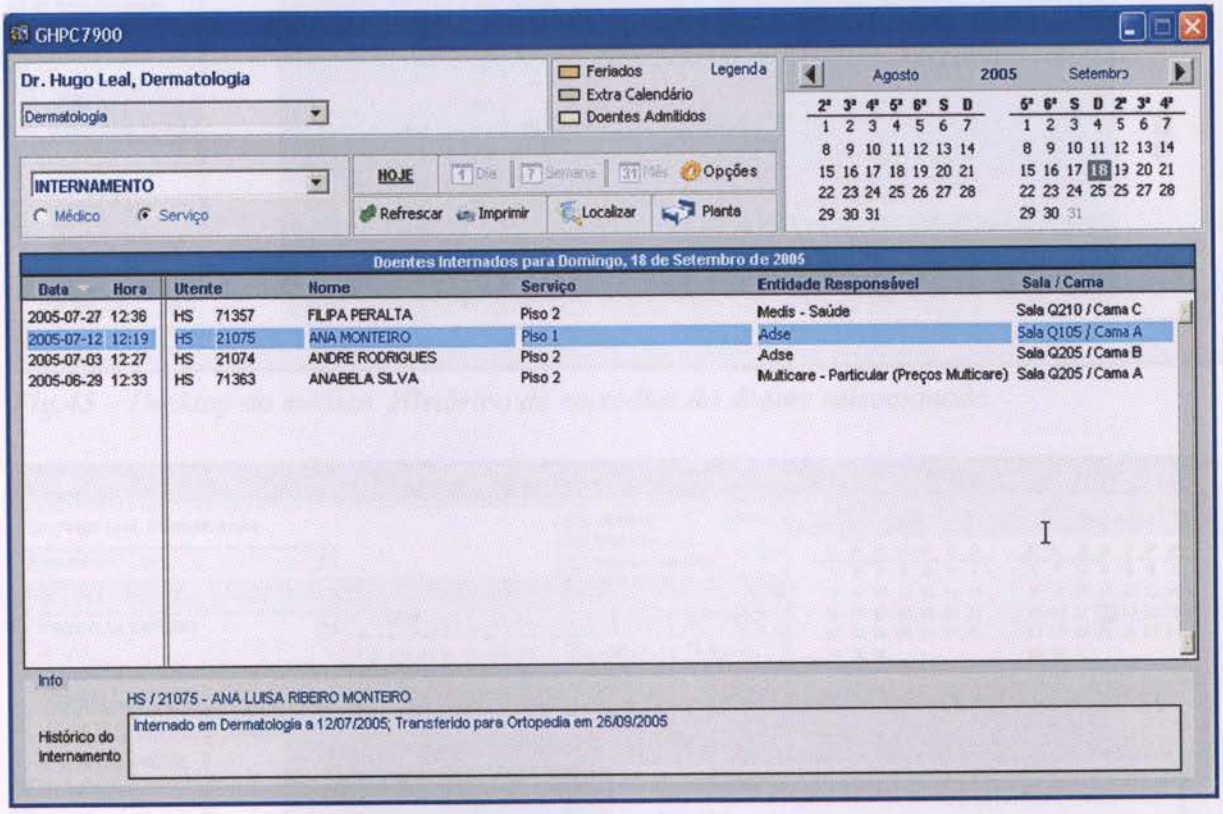


***Fig.44** – Desktop do médico. Internamentos em determinado dia.*

Um nova funcionalidade da nova versão do *desktop* é a criação de um resumo de informação clínica por episódio. Encontrando-se no modo do histórico (tendo pressionado o botão "Localizar"), o médico pode seleccionar qualquer episódio[24]. O procedimento convencional, aquele disponível na versão anterior, seria invocar o *form GHPC8000,* onde seria apresentada toda a informação clínica do episódio (e outra informação relevante de outros episódios). NA nova versão do *Processo Clínico* o médico pode aceder a um resumo da informação do episódio directamente a partir do *desktop*, obtendo imediatamente o resumo do episódio seleccionado, sem que ter que sair do *desktop*. Note-se que este resumo dos episódios no *desktop* do médico não substitui a utilização do *GHPC8000*. O resumo é apenas para consulta rápida. No *GHPC8000* é possível registar e modificar toda a informação clínica, e esta é aí apresentada numa forma de mais fácil leitura.

[24] Denomina-se por episódio qualquer acto que possa sr registado na aplicação, como uma consulta, uma entrada na urgência ou no bloco operatório, ou um internamento.

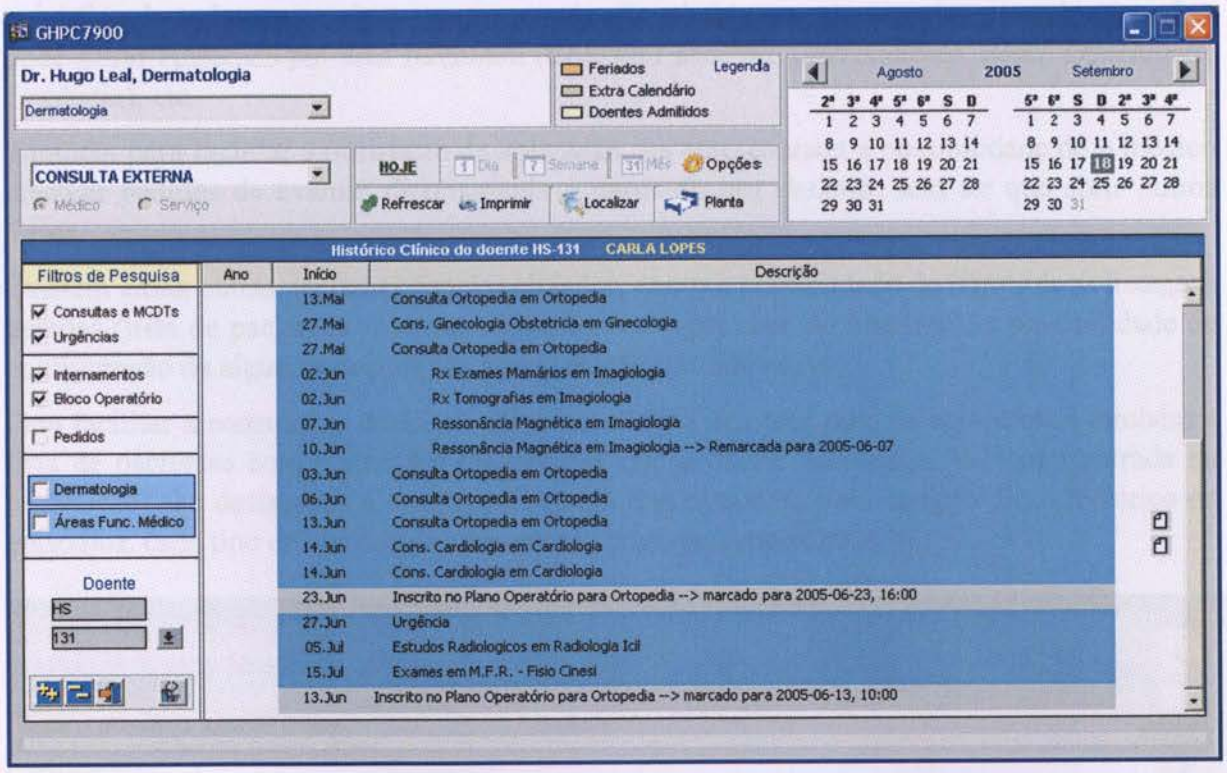


Fig.45 *– Desktop do médico. Histórico de episódios do doente seleccionado.*

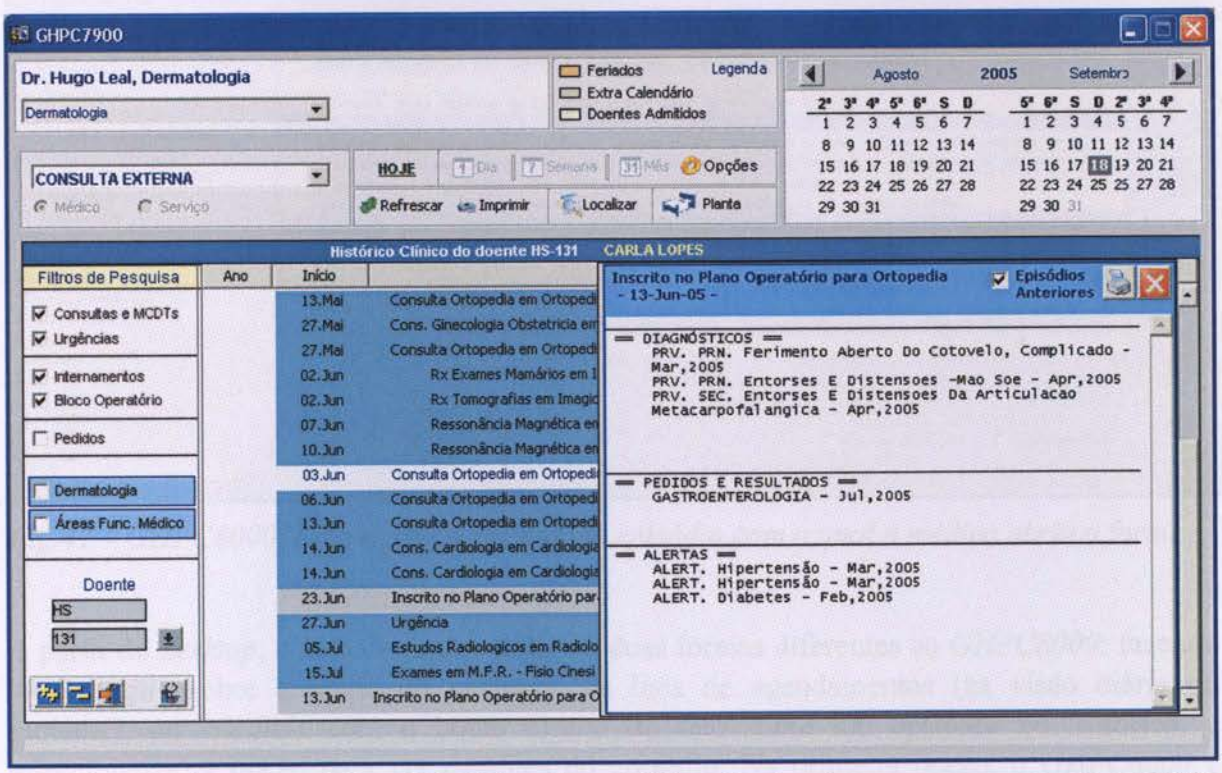


Fig.46 *– Desktop do médico. Histórico de episódios do doente, com o resumo do episódio seleccionado.*

Ainda no histórico de episódios do paciente, o médico tem disponíveis várias caixas de selecção, que lhe permitem adicionar vários filtros à lista de episódios. Este pode ver apenas os episódios da especialidade seleccionada (cardiologia, dermatologia, etc.), pode ver os

episódios de todos os serviços que exerce (e não só do que está actualmente seleccionado), pode filtrar episódios por área funcional (*Consulta Externa, Internamento, Bloco Operatório, Urgência*), etc.

Também para facilitar a utilização da aplicação, foi acrescentada a possibilidade de o médico efectuar pedidos de exames directamente a partir do seu *desktop*, sem ter que abrir outros *forms*.

Existem ainda outras pequenas funcionalidades, como a possibildade de imprimir as listagens geradas (lista de pacientes com marcação, lista de episódios do paciente), a possibilidade de configuração de algumas opções (de apresentação dos dados), etc.

Para facilitar a consulta, o *desktop* utiliza um código de cores para os episódios, e também a lista de pacientes com marcação. Os doentes que já foram admitidos (já deram entrada na instituição) são destacados a uma cor diferente dos restantes, por exemplo. E no histórico de episódios, cada tipo de episódio é apresentado com uma cor própria.

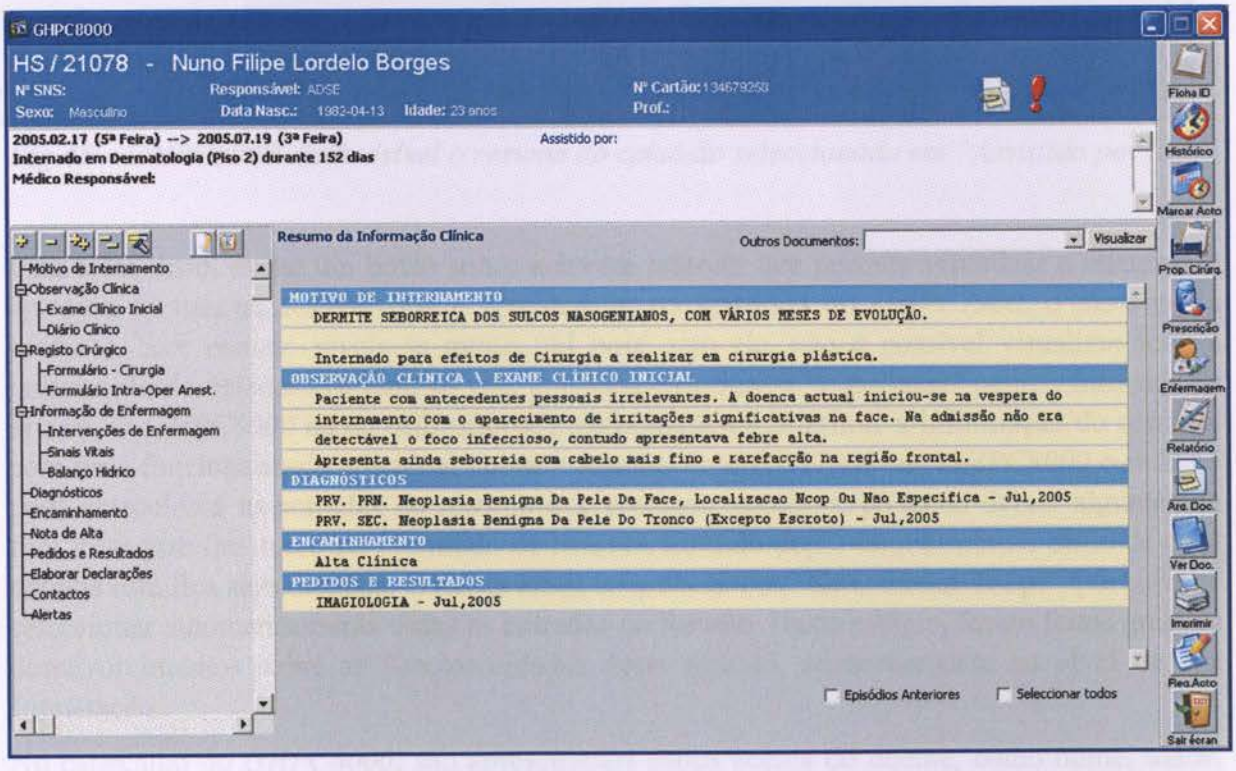


Fig.47 *– GHPC8000. Está visível o resumo do episódio com o qual o médico abriu o form.*

A partir do *desktop*, o médico pode aceder de duas formas diferentes ao *GHPC8000*: fazendo *duplo-clique* sobre o nome do paciente, na lista de agendamentos (na visão diária ou semanal), ou *clicando* com o botão direito do rato sobre um episódio no histórico, e seleccionando a opção para ver detalhes do episódio. Na primeira opção, o *GHPC8000* é aberto para o último episódio do doente seleccionado (o *GHPC8000* apresenta a informação clínica por episódio). Na segunda, abre com a informação do episódio seleccionado.

A lógica do funcionamento do *GHPC8000* é simples. De acordo com o tipo de episódio, é apresentada uma árvore no painel do lado esquerdo (ver figuras). Cada nó desta árvore corresponde a determinado tipo de informação, sendo aberto um módulo no centro do *GHPC8000* que permite gerir essa informação sempre que um nó é seleccionado.

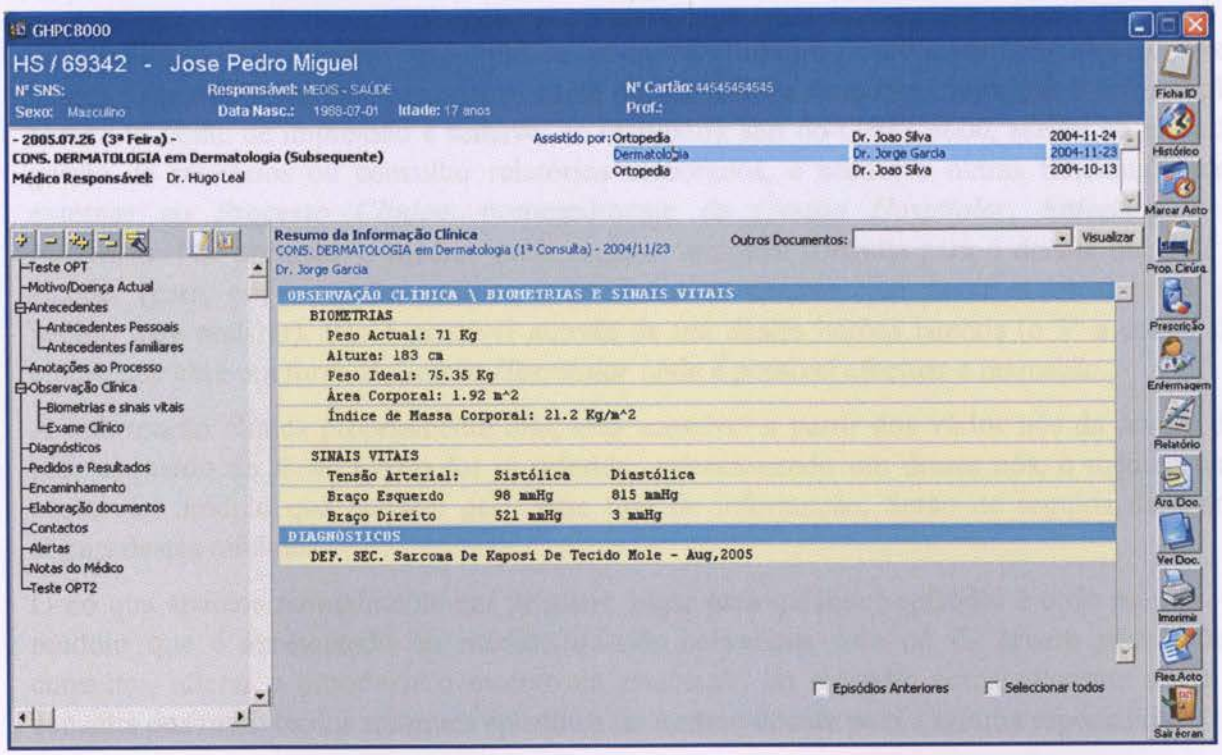


***Fig.48** – GHPC8000. Está visível o resumo do episódio seleccionado em "Assistido por".*

Para além disto, existe um botão sobre a árvore referida que permite visualizar o resumo do episódio (o mesmo resumo que é apresentado no histórico do *GHPC7900*, o *desktop* do médico). Este resumo revela-se muito útil pois, sem ele, não é possível visualizar toda a informação do episódio de uma só vez, pois esta encontra-se "espalhada" pelos vários nós da árvore. O *GHPC8000* dá ao utilizador a possibilidade de imprimir a informação do resumo, com uma funcionalidade adicional sobre o resumo do *GHPC7900*: no *GHPC8000* o médico pode seleccionar as entradas do resumo que pretende imprimir, podendo deixar algumas de fora, caso isso lhe interesse. No modo de resumo, fazendo duplo-clique sobre o título de cada entrada esta fica seleccionada. E existe ainda uma *check-box* "Seleccionar Todos" que permite seleccionar automaticamente todas as entradas do resumo. Neste estágio, foram feitos grandes desenvolvimentos sobre as funcionalidades deste resumo, nomeadamente ao nível da sua formatação.

No cabeçalho do *GHPC8000*, são apresentados dados acerca do doente, como nome, idade, profissão, etc. Por baixo da caixa onde estes dados aparecem, surge a informação do episódio cujos dados clínicos são apresentados. Para além da descrição do episódio, é aqui apresentada a data em que se realiza, assim como o médico responsável.

Ainda neste bloco de informação do episódio, do lado direito, é apresentada uma lista de outros episódios (com a legenda "Assistido por"). Tratam-se dos episódios do mesmo doente realizados antes do actual. Seleccionando um destes episódios, o médico tem acesso ao resumo desse episódio. Contudo, apenas o resumo desse episódios é acessível dessa forma, não sendo possível, por exemplo, alterar a informação clínica dos mesmos. O *GHPC8000* só permite a alteração e introdução de dados clínicos para o episódio para o qual foi aberto. A possibilidade de consultar o resumo dos episódio anteriores existe porque o médico pode ter necessidade de aceder a esses dados enquanto estiver a introduzir ou analisar a informação do episódio actual.

A interface do *GHPC8000* disponibiliza ainda uma barra de botões do lado direito. Estes botões disponibilizam funcionalidades como aceder à ficha do doente, imprimir a informação clínica (o botão de impressão é sensível ao contexto), sair do *GHPC8000*, aceder ao *form* de gestão de relatórios ou consultar relatórios elaborados, e aceder a outras funcionalidades externas ao *Processo Clínico*, nomeaadmente da *Gestão Hospitalar*, *Enfermagem* e *Farmácia*. Por exemplo, o médico pode registar um nova consulta para o doente que está a atender (para, por exemplo, analisar o resultado de exames que possa eventualmente ter pedido para realizar). Isto é possível através de um destes botões laterais (o 3° a contar de cima), que abre um *form* da *Gestão Hospitalar* onde é possível efectuar a marcação.

A informação clínica propriamente dita, está acessível a partir dos vários nós da árvore do lado esquerdo do *form*. Como foi já referido, seleccionando um destes nós, o médico tem acesso ao módulo que permite gerir esse tipo de informação. Serão de seguida descritos alguns destes módulos.

O nó que aparece normalmente em primeiro lugar para qualquer episódio é o do motivo. O módulo que é apresentado ao médico quando selecciona este nó da árvore permite-lhe consultar, alterar e introduzir o motivo da realização do episódio actual. Permite ainda a consulta dos motivos dos restantes episódios do mesmo doente para a mesma especialidade.

Outro módulo disponível na maioria dos tipos de episódio é o dos dados bio métricos (ver fig.49). A informação aqui apresentada é específica do doente, e não de um episódio em particular. Todos os episódios do mesmo doente apresentam a mesma informação neste nó.

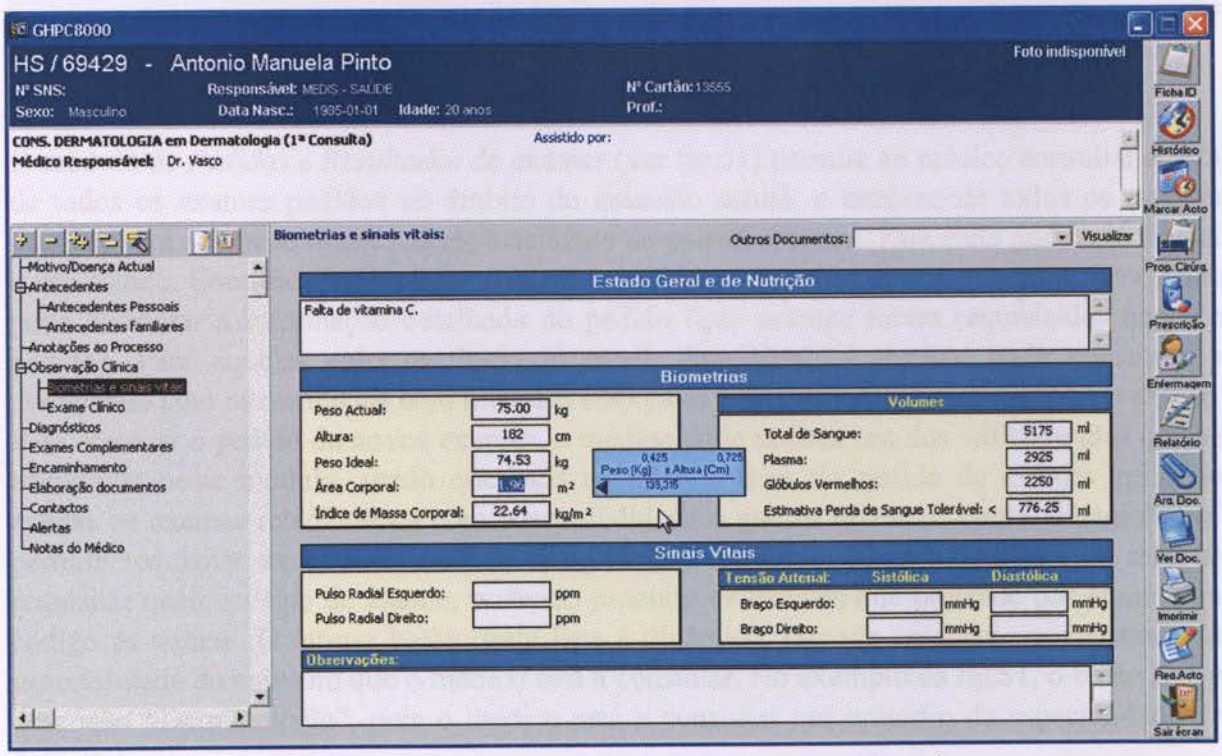


***Fig.49** – GHPC8000. Está visível o módulo "Biometrias e Sinais Vitais".*

O módulo dos diagnósticos (ver fig.50) permite ao médico gerir os vários diagnósticos atribuídos ao doente. Neste estágio foram desenvolvidas novas funcionalidades para esta módulo, e também alterados alguns comportamentos para facilitar a sua utilização. Na nova

versão é possível associar um diagnóstico a um documento, que consiste normalmente num conjunto de indicações para o doente seguir relativamente à condição que lhe foi diagnosticada.

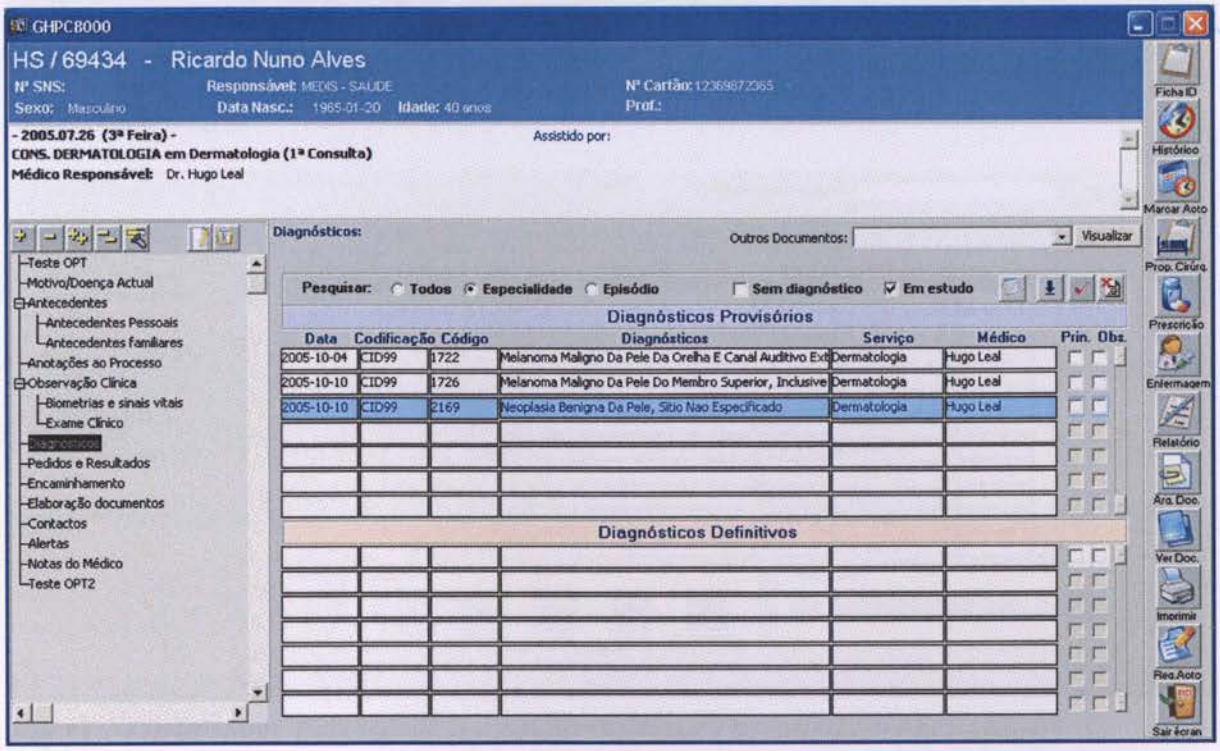


Fig.50 *– GHPC8000. Está visível o módulo "Diagnósticos".*

O módulo de *Pedidos e Resultados* de exames (ver fig.51) permite ao médico consultar a lista de todos os exames pedidos no âmbito do episódio actual, e também de todos os exames pedidos para o mesmo doente na especialidade do episódio actual. Para cada pedido, é visível o seu estado. Com um *duplo-clique* sobre um pedido, o médico acede a um novo *form*, onde pode consultar a informação detalhada do pedido (que exames foram requisitados naquele pedido). Para aqueles cujo resultado já esteja disponível, o médico pode consultar o documento com os resultados e/ou relatório elaborado pelo especialista que efectuou o exame. Para efectuar o pedido de novos exames, o médico pode utilizar um dos vários botões que se encontram neste módulo, sendo que cada um abre o *form* de pedido de exames exibindo apenas os exames relacionados com a especialidade a que se refere (o botão "Imagiologia" permite requisitar exames da área de imagiologia). O botão "Outros" permite ao médico requisitar qualquer tipo de exame, podendo procurar os exames que pretende por nome e/ou código de exame. O último botão desta lista é dinâmico, estando associado aos exames da especialidade do episódio que o médico está a consultar. No exemplo da fig.51, o botão tem a descrição "Dermatologia", pois o médico está a consultar um episódio da especialidade de dermatologia.

Uma nova funcionalidade desenvolvida neste estágio é a dos protocolos. Tratam-se de conjuntos de prescrições e outras acções predefinidas, que permitem ao médico acelerar o registo de exames e consultas para procedimentos padrão. Poderia ser criado um protocolo a utilizar quando o paciente evidenciasse determinado tipo de sintomas. O protocolo poderia compreender a realização de vários tipos de exame para datas predefinidas a contar do dia da

marcação, assim como a marcação de nova consulta para depois da realização dos exames, para que o médico pudesse analisar o resultado dos mesmos.

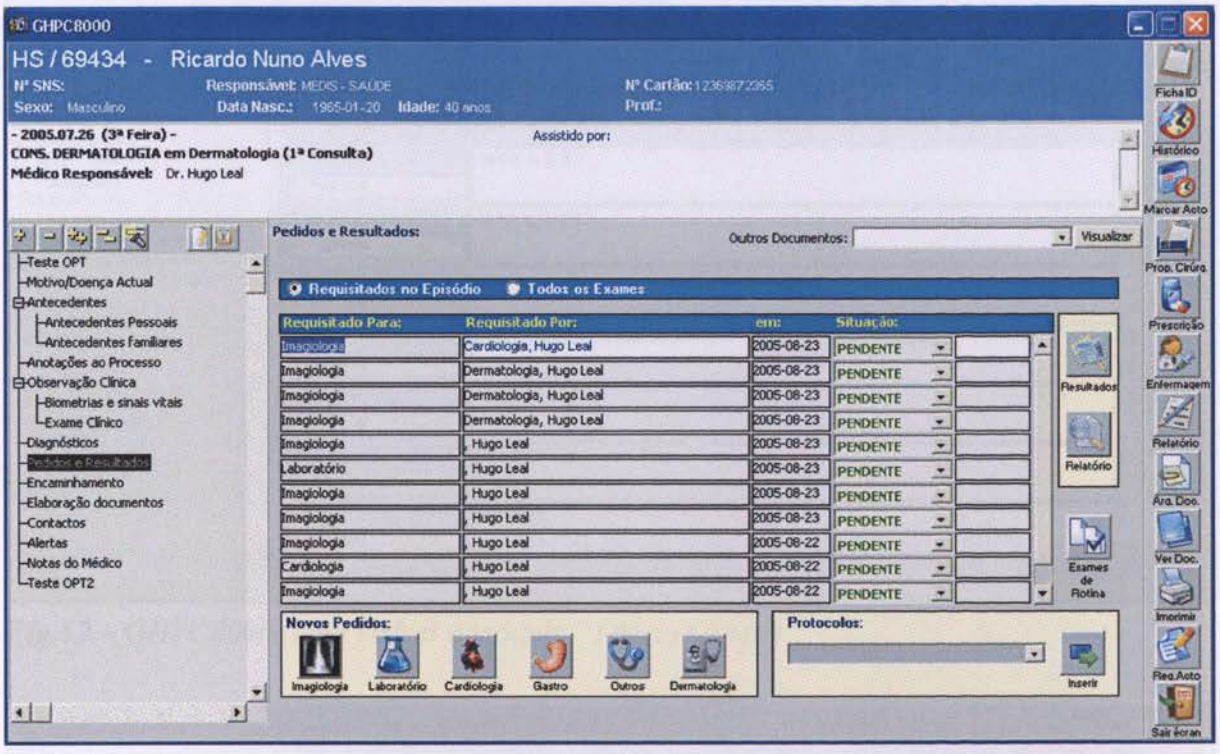


Fig.51 *– GHPC8000. Está visível o módulo "Pedidos e Resultados" (de exames).*

O módulo do *Diário Clínico* (ver fig.52), disponível apenas nos episódios do tipo internamento, permite ao médico registar (e consultar) notas clínicas durante o internamento de um paciente. Este módulo foi também alvo de grandes desenvolvimentos durante este estágio. Na versão inicial, as entradas do diário apresentadas eram apenas filtradas por doente, sendo que todas as notas registadas para qualquer internamento apareciam misturadas, o que dificultava a sua análise. A nova versão oferece a possibilidade de apresentar apenas as notas do internamento actual, ou de todos os internamentos do doente. Quando são apresentados todos os internamentos, o sistema de legendagem implementado permite facilmente saber a que internamento cada nota corresponde. Aparece uma barra colorida antes de cada registo, sendo que cada cor corresponde a um internamento. Na legenda que se encontra no fundo do módulo, encontram-se rectângulos coloridos, com as cores dos internamentos apresentados. Passando o ponteiro do rato sobre um desses rectângulos, torna-se visível a descrição do internamento: especialidade a que corresponde, data de internamento, data de alta (se já teve alta), sala e cama.

Outro módulo sobre o qual se efectuaram desenvolvimentos neste estágio é o das notas privadas. Os restantes nós da árvore permitem a gestão de informação que fica disponível para qualquer médico consultar. Mesmo que o médico não tenha permissões para alterar essa informação, pode sempre consultá-la no resumo dos episódios disponibilizado no *desktop* do médico (*GHPC7900*). Para dar ao médico a possibilidade de registar informação confidencial associada aos episódios, foi criado o módulo *Notas do Médico*. Esta informação só está acessível para o médico que a introduziu, e só é apresentada nos resumos dos episódios quando for esse médico a utilizar a aplicação.

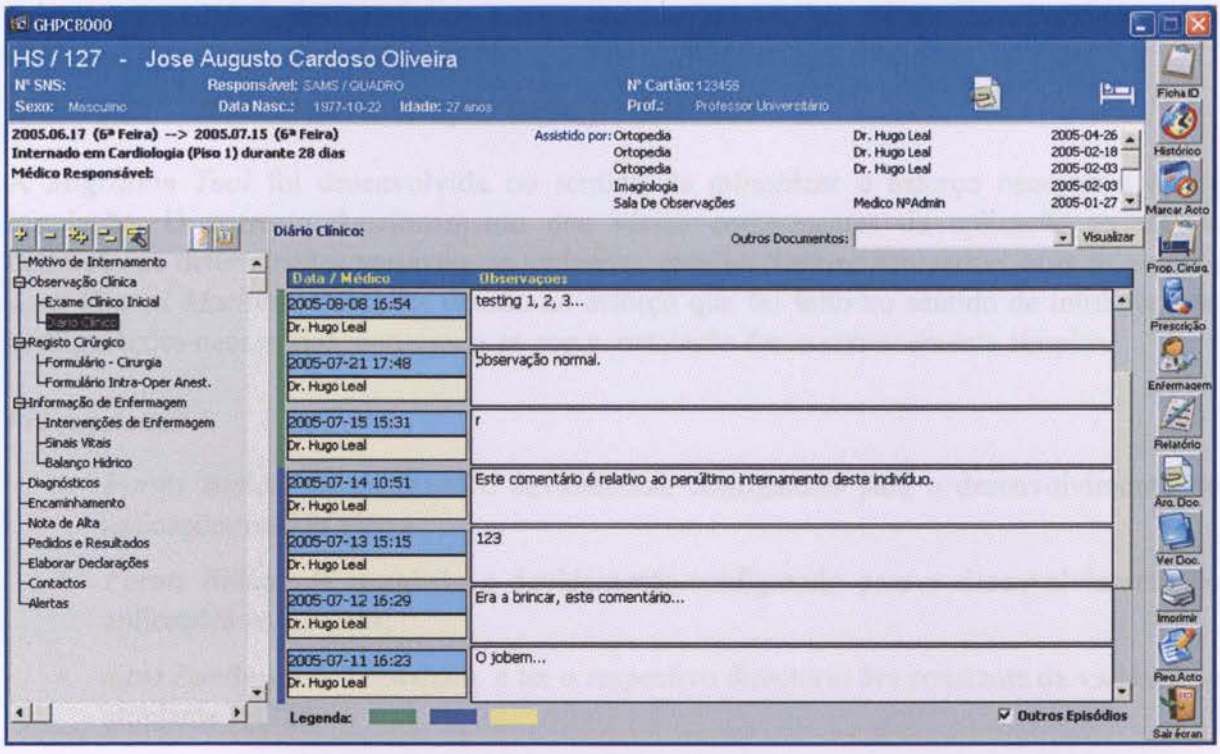


Fig.52 *– GHPC8000. Está visível o módulo "Diário Clínico".*

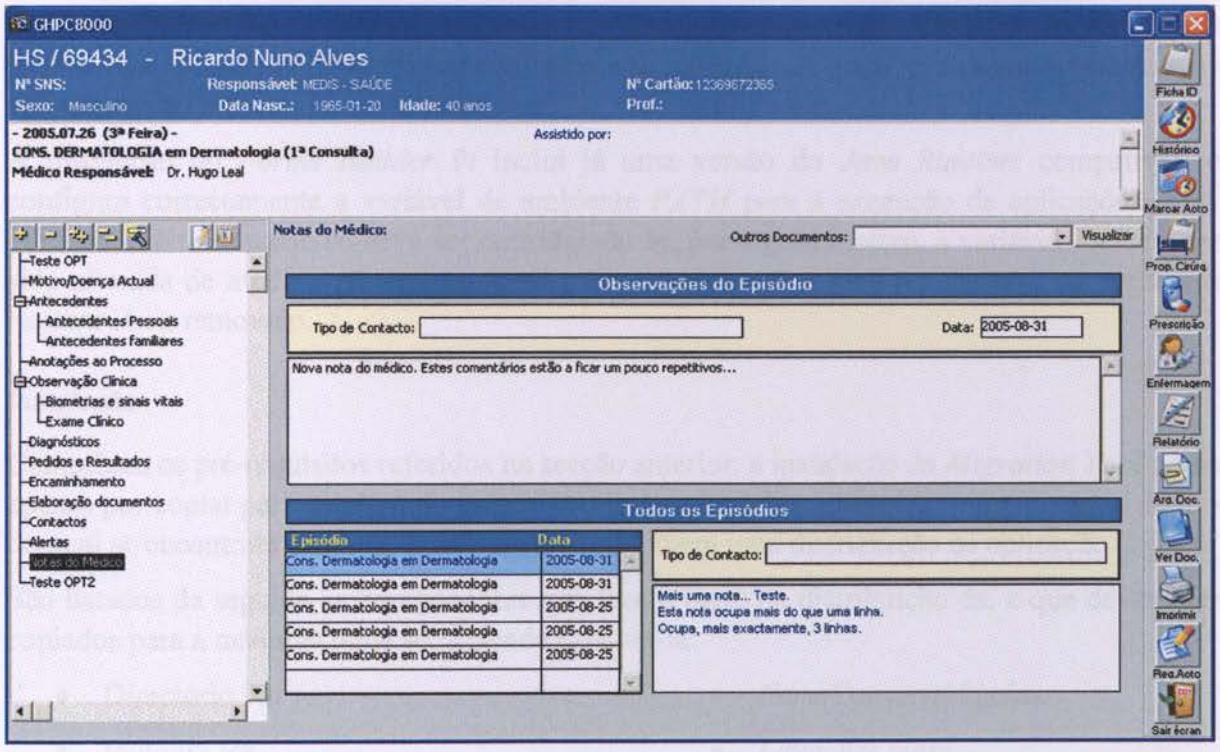


Fig.53 *– GHPC8000. Está visível o módulo "Notas do Médico".*

Neste módulo o médico pode consultar todas as notas que registou para qualquer episódio do doente, assim como alterá-las ou introduzir novas.

A identificação que é pedida ao médico quando este inicia a aplicação (através de *login* e *password*) garante a confidencialidade e integridade da informação disponibilizada no *Processo Clínico Electrónico.*

ANEXO E: Manual de Instalação da *Migration Tool*

A *Migration Tool* foi desenvolvida no sentido de minimizar o esforço necessário à sua instalação. O correcto funcionamento dos vários componentes da aplicação exigem a definição de determinadas variáveis de ambiente, quer do Sistema Operativo, quer da própria *Java Virtual Machine*. Contudo, devido ao esforço que foi feito no sentido de minimizar as configurações necessárias, conseguiu-se que a instalação fosse extremamente simples.

Pré requisitos

- *Forms Builder 6i* instalado e devidamente configurado para o desenvolvimento de aplicações na CPCHS;
- *Forms Builder 9i* instalado e devidamente configurado para o desenvolvimento de aplicações na CPCHS;
- *Java Runtime 1.4.2* instalado, e ter o respectivo directório *bin* constante da variável de ambiente *PATH*;

É necessário que a máquina onde a aplicação vai ser executada tenha devidamente instalados e configurados o *Forms Builder 6i* e o *Forms Builder 9i*. Não se tratam de exigências difíceis de cumprir, já que qualquer máquina de desenvolvimento da CPCHS dispõe destas ferramentas instaladas e configuradas, pois são necessárias para o desenvolvimento das aplicações da empresa.

A instalação do *Forms Builder 9i* inclui já uma versão do *Java Runtime* compatível, e configura correctamente a variável de ambiente *PATH* para a execução de aplicações Java. Por isso, o último passo só deve ser considerado se, por algum motivo, a variável *PATH* tiver sido alterada de modo a não ser possível executar aplicações Java no sistema, ou o próprio *runtime Java* removido.

Instalação

Cumpridos os pré-requisitos referidos na secção anterior, a instalação da *Migration Tool* passa apenas por copiar para a máquina onde se pretende utilizá-la o directório m*igrationtool* dentro do qual se encontram todos os ficheiros que constituem uma distribuição da aplicação.

São listados de seguida os componentes que fazem parte da distribuição da, e que devem ser copiados para a máquina onde se pretende executá-la:

- Directório *bin*
- *f60jdapi.jar*
- *f90jdapi.jar*
- *migracao.jar*
- *FormConverter9i.class*
- *Launcher.class*
- *Launch.bat*
- Ficheiros de configuração (*Options.pref* e *Libraries.pref*)

Execução

Para iniciar a aplicação basta executar o ficheiro *Launch.bat*.