FEUP Universidade do Porto
Faculdade de Engenharia

3 ao Quadrado - Agenda Web

Relatório de Desenvolvimento

Gestão de Projectos de Software - Grupo A - LEIC 2001/2002
http://gnomo.fe.up.pt/gps01A

João Montenegro - ei97023@fe.up.pt

André Teixeira - ei97024@fe.up.pt
César Rodrigues - ei97006@fe.up.pt
Nuno Dias - ei95030@fe.up.pt
Rui Soares - ei97026@fe.up.pt
Carlos Ribeiro - ei97043@fe.up.pt
Filipe Pinto - ei96036@fe.up.pt
Rúben Pereira - ei96033@fe.up.pt
Vânia Gonçalves - ei97011@fe.up.pt

Versão 1.0

22 de Dezembro de 2001

Conteúdo

Lista de Figuras

1 Introdução

Terminada a implementação da aplicação **Agenda Web**, torna-se necessário que exista um documento, ou um conjunto de documentos, que possibilitem no futuro, a manutenção, ou mesmo a evolução do sistema com o acrescento de novos módulos e funcionalidades.

1.1 Objectivo

Pretende-se com a apresentação deste relatório documentar o processo de desenvolvimento da aplicação **Agenda Web**. Para tal, serão apresentados vários diagramas, bem como as respectivas explicações. Para descrever a arquitectura lógica, será utilizado um diagrama de pacotes. Para a arquitectura física serão apresentados diagramas de componentes e distribuição. Será ainda apresentado o diagrama de classes da aplicação. Este é bastante relevante, visto que ilustra de uma forma muito condensada a estrutura da aplicação (ao nível da programação), bem como as relações entre classes e entre métodos. Serão ainda apresentados alguns detalhes de implementação da aplicação.

1.2 A aplicação

Pretende-se que a **Agenda Web**, desenvolvida pela **3 ao Quadrado**, seja uma ferramenta de fácil utilização no dia-a-dia, e que sirva para de alguma forma facilitar a vida ao utilizador, nomeadamente no que toca à marcação dos mais variados compromissos. Para tal, a **Agenda web** põe ao serviço do utilizador um conjunto de funcionalidades, das quais se destacam a marcação de compromissos entre várias pessoas, gerindo todo o processo que envolve confirmações, recusas, cancelamentos, alterações, etc. Podemos também destacar a marcação de compromissos ou disponibilidades periódicas (que se repetem com regularidade), com a possibilidade de definir excepções a essa periodicidade. Outra funcionalidade digna de relevo, é a de ser possível notificar o dono de uma agenda, sempre que tal se revele necessário.

Estas são algumas das funcionalidade que a **Agenda Web** põe ao dispor do utilizador.

1.3 A implementação

Convém referir que o processo de desenvolvimento da aplicação foi algo atribulado. Dado que se trata de um projecto de reengenharia, supostamente deveria ser possível reutilizar, ou desenvolver novas funcionalidades a partir da aplicação recebida. Tal não foi possível, em parte devido ao facto da tecnologia da aplicação recebida não atingir os níveis de estabilidade que são

exigíveis a uma aplicação desta natureza. Revelou-se então necessário proceder à mudança da tecnologia, o que contribuiu de forma decisiva para o atraso da implementação da aplicação. À parte de todos os percalços, é convicção da **3 ao Quadrado**, que foi conseguida uma aplicação de acordo com os requisitos mínimos negociados com a cliente, assegurando o nível de qualidade que é marca da empresa.

2 Arquitectura Lógica

No desenvolvimento da **Agenda Web**, utilizou-se uma arquitectura lógica definida em camadas e pacotes lógicos (tal como é definida no *Relatório de Projecto*). Conseguiu-se, assim, estruturar melhor a aplicação, beneficiando-se das vantagens inerentes à modularização (facilidade de alteração, manutenção, etc).

A arquitectura lógica está representada pelo seguinte diagrama de pacotes lógicos.

2.1 Diagrama de Pacotes

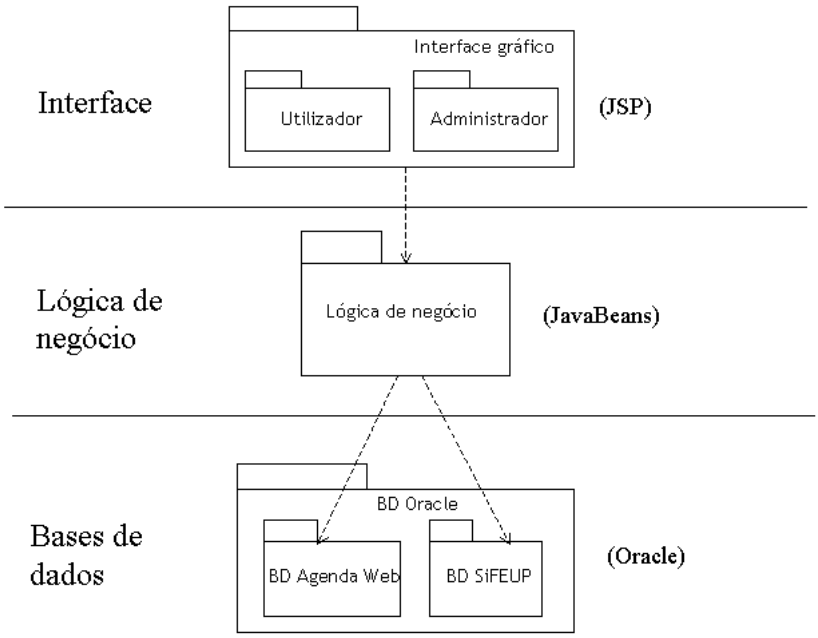

Figura 2.1: Diagrama de Pacotes.

2.2 Camada Interface

A camada **Interface** é composta por um pacote lógico: **Interface gráfico**. Dentro deste pacote existem dois outros pacotes: **Utilizador** e **Administrador**. Estes dois pacotes definem a interface gráfica com os dois actores do sistema[1]: utilizador e administrador. Esta camada lógica é implementada em *JSP*s[2]. Esta separação (da camada **Interface**) do resto da aplicação, permite uma maior facilidade de adaptação do sistema a possíveis futuras mudanças.

2.3 Camada Lógica de Negócio

A camada **Lógica de Negócio**, que contém o pacote **Lógica de Negócio**, utiliza componentes *JavaBeans* para fazer a ligação entre a camada superior (**Interface**) e a camada inferior (**Base de Dados**), assegurando o correcto funcionamento da *Agenda Web*. Implementa, portanto, todas as funcionalidades através de módulos de código reutilizáveis, chamados de componentes *JavaBeans*.

2.4 Base de Dados

Esta camada engloba as bases de dados **BD Agenda Web** e **BD SiFEUP**, no pacote lógico **BD Oracle**. Estas são as bases de dados utilizadas pela aplicação. A **BD Agenda Web** é a principal base de dados desta, sendo necessária para o funcionamento da aplicação. São aqui guardadas as informações relativas aos compromissos, disponibilidades, etc. A **BD SiFEUP** representa a base de dados do SiFEUP, existente na Faculdade de Engenharia da Universidade do Porto, da qual são obtidas informações de algumas tabelas.

[1] Ver a secção *casos de Uso* do *Relatório de Especificação de Requisitos*
[2] Ver secção *Tecnologias* na página 8

3 Tecnologias

Relativamente às três camadas lógicas, definidas na *Arquitectura Lógica*[3], foi escolhida a solução tecnológica que mais se adequava ao problema.

Começando pela camada de **Base de dados**, optou-se por implementar a base de dados **BD Agenda Web** (principal base de dados da aplicação) em *Oracle*, utilizando a versão 8.1.7, disponível na Faculdade de Engenharia da Universidade do Porto. Esta opção foi escolhida não só devido ao facto de ser uma solução disponível na Faculadade (e já ser utilizada na **BD SiFEUP**, da qual foram utilizadas algumas tabelas), como também, por suportar directamente *JavaBeans*.

Quanto à camada **Lógica de Negócio**, foram utilizados *JavaBeans*, (tecnologia desenvolvida pela *SUN*) e que traz várias vantagens relativamente à reutilização de código e modularização, devido a ser uma tecnologia baseada em componentes. A alternativa seria a utilização de *Enterprise JavaBeans* (*EJB*s), mas devido a problemas de integração desta tecnologia com a base de dados *Oracle*, esta foi colocada de parte. No entanto, a utilização de *EJBs* tornaria mais fácil a comunicação com a base de dados, sendo esta transparente. Na solução adoptada, foram implementados *beans* específicos para tratar dessa comunicação e oferecer o mesmo tipo de transparência à implementação da **lógia de negócio**

Por fim, na camada de **Interface**, foi utilizado *JSP* (Java Server Pages), que permite uma integração fácil com os *JavaBeans* e gerar dinamicamente as *páginas Web* de interface com os utilizadores. O código gerado pelos *JSP*s, é *Dynamic HTML*, ou seja, utiliza linguagens como *HTML*, *JavaScript* e *Cascading StyleSheets*. Para implementar esta solução utilizou-se as aplicações *Apache Web Server* com módulo *JServ*.

[3] ver secção *Arquitectura Lógica* na página 6

4 Projecto Detalhado

4.1 Interfaces

4.1.1 Sessão

A interface Web da agenda utiliza o sistema de sessão do servidor para transportar informação relevante de página em página durante a navegação. Todos os ficheiros *JSP* incluem o ficheiro "session.jsp" para que as variáveis de sessão sejam lidas, ou, no caso de ainda não existirem, serem inicializadas. São utilizadas três destas variáveis:

- `curDate`, do tipo `GregorianCalendar`: contém a data correspondente à vista actual, esta data pode ser diferente da data actual;
- `owner`, do tipo `User`: contém a informação e identificação do dono da agenda;
- `personal`, do tipo `boolean`: informa se o utilizador está a visitar a sua agenda pessoal, após se ter autenticado no SiFEUP.

As variáveis `curDate` e `owner` podem ser alteradas através da navegação, para que se possa visualizar os compromissos com uma data diferente da actual, ou pertencentes a outro dono. Esta alteração pode ser efectuada através da instanciação de variáveis na `Query String` do `URL` da página que se pretende visualizar. O ficheiro "session.jsp" verifica a `Query String`, actualizando a sessão se necessário. A lista seguinte apresenta os nomes das variáveis que podem ser instanciadas na `Query String`:

- Para a alteração da `curDate` (data de visualização):
  - `curDay`: inteiro com o dia do mês da nova data;
  - `curMonth`: inteiro com o mês;
  - `curYear`: inteiro com o ano.
- Para a alteração do owner:
  - `idDocente`: inteiro com o ID de um docente na base de dados;
  - `IdAluno`: string com o ID de um aluno na base de dados.

O seguinte URL é um exemplo do uso da funcionalidade da alteração do conteúdo sessão, neste caso o owner passará a conter a informação do docente com ID=124, e a vista apresentará os compromissos do dia 23/11/2001: http://forge.fe.up.pt/agenda/diaria.jsp?idDocente=124-&curDay=13&curMonth=11&curYear=2001

4.1.2 Vistas da Agenda

Os compromissos de uma agenda podem ser apresentados ao utilizador através de três tipos de vistas: Diária (ficheiro "diaria.jsp") que mostra os compromissos de um dia, Semanal (ficheiro "semanal.jsp") que mostra os compromissos de uma semana e Mensal (ficheiro "mensal.jsp") que mostra os compromissos de um mês. As vistas podem funcionar em dois modos: modo Pessoal, ou modo Visitante.

O modo **Pessoal** é activado sempre que o utilizador está a visitar a sua agenda após se ter autenticado no SiFEUP (variável de sessão `personal` igual a `true`), neste caso o utilizador está já identificado pelo sistema, sendo-lhe assim permitido alterar e apagar os compromissos da sua agenda.

O modo **Visitante** é activado sempre que o utilizador visita uma agenda que não a sua, ou quando visita a sua página sem se ter autenticado no SiFEUP. Neste segundo modo, o utilizador pode apenas ver os compromissos públicos, sem poder alterá-los ou apagá-los.

Todas as vistas estão inseridas no *layout* genérico do SiFEUP. É mostrado do lado direito um menu com variados *links* úteis à navegação:

- **Hoje** *link* para a vista diária do dia de hoje;
- **Novo Compromisso** *link* para o formulário de marcação de compromisso (apenas no modo **Pessoal**);
- **Nova Disponibilidade** *link* para o formulário de marcação de disponibilidade (apenas no modo **Pessoal**);
- **Configuração** *link* para o formulário de alteração das opções do utilizador (apenas no modo **Pessoal**);
- **Ver Outra Agenda** *link* que permite pesquisar uma pessoa e passar a visualizar a sua agenda;
- **Calendário** permite ver um pequeno calendário, que quando o utilizador clica num dia ou semana, passa a visualizar a respectiva vista da agenda nessa data;
- **Login** *link* para a página de autenticação do SiFEUP (apenas se o utilizador nao se tiver autenticado anteriormente).

Por baixo deste menu, por vezes poderá aparecer um "placard" com avisos que requerem a atenção do utilizador. Estes avisos referem-se a compromissos propostos ao utilizador que ainda não foram confirmados ou que foram alterados, e também compromissos que o utilizador aceitou e que se tornaram definitivos ou foram cancelados. Uma listagem mais extensa destes compromissos é vísivel na página "bigBoard.jsp".

Vista diária

A vista diária apresenta os compromissos de uma agenda para um dia através de um horário com entradas horizontais para cada meia hora (o utilizador pode definir nas suas opções pessoais a hora de início e a de fim de visualização, logo o número de meias-horas visíveis pode variar). Cada compromisso ocupa uma célula da tabela que constitui o horário, com um *rowspan* igual ao número de meias horas da sua duração. É utilizado um algoritmo de desenho da tabela que prevê a existência de dois ou mais compromissos sobrepostos (que ocupam uma ou mais meias-horas equivalentes).

Algoritmo de desenho da tabela do horário

- Criar o *Bean* `appApi` do tipo `AppointmentApi`;
- Executar o método `getDayAppointments` de `appApi`, passando o objecto `owner` e a data `curDay` (variáveis de sessão) como argumentos, recolhendo o vector (objecto do tipo Vector) com os compromissos (objectos do tipo `Appointment`) devolvido pelo método invocado;
- Criar uma matriz bidimensional de dimensões 10 (máximo de compromissos sobrepostos) por N (número de meias-horas apresentadas na vista), ou seja, `matriz[10][N]`. Esta matriz passará a conter a referências para os compromissos presentes no vector criado no passo anterior do algoritmo, distribuídas de forma a conter uma representação das células da tabela em *HTML* que será gerada;
- Para cada elemento do vector de compromissos, coloca-se uma referência para si nas células da matriz com linhas correspondentes às meias horas que este ocupa e a coluna o mais à esquerda possível;
- Percorrer a matriz obtida desenhando todas as células da tabela final, de acordo com o conteúdo da matriz.

Este algoritmo tenta encaixar os compromissos nas colunas mais à esquerda possível. O uso da matriz deve-se ao facto de não ser possível gerar uma tabela em *HTML* com esta característica, visto não ser possível saber quantas colunas é que já estão ocupadas numa dada linha, por causa do uso dos *rowspans*.

Vista semanal

Esta vista mostra os compromissos de uma dada semana. O algoritmo de desenho da tabela semanal segue os seguintes passos:

- Criar o *Bean* `appApi` do tipo `AppointmentApi`;
- Calcular o dia inicial e o final da semana da data `curDate` (da sessão);

- Executar o método `getWeekAppointments` de `appApi`, passando o objecto `owner` e a data de início da semana como argumentos, recolhendo o vector (objecto do tipo `Vector`) com os compromissos (objectos do tipo `Appointment`) devolvido pelo método invocado;
- Percorrer os dias da semana, desde o inicial até ao final, verificando quais os compromissos presentes nesses dias e desenhando a tabela nesta passagem.

Vista mensal

A vista mensal apresenta ao utilizador os dias do mês sob a forma de um calendário (ver figura). Cada célula correspondente a um dia do mês contem um *link* para a vista diária desse dia mostrandos os compromissos do dia. O dia `curDate` (presente na sessão) está assinalado com uma tonalidade mais clara.

É possível saber se um dia contém ou não compromissos invocando o método `getMonthAppointments` do objecto `appApi` (um *Bean* criado nesta vista do tipo `AppointmentApi`), que devolve um array do tipo *boolean* com uma entrada por cada dia do mês (igual a *true* se houver compromissos marcados para esse dia).

O algoritmo usado para o desenho do calendário é o seguinte:

- Criar o *Bean* `appApi` do tipo `AppointmentApi`;
- Calcular o dia inicial e o final do mês da data `curDate` (da sessão);
- Invocar o método `getMonthAppointments` do objecto `appApi`, que devolve o *array* de *boolean*;
- Desenhar células vazias na primeira linha do calendário até ao dia da semana do dia 1 do mês;
- Desenhar as células correspondentes aos dias do mês, criando os links para a vista diária de cada dia e assinalando os dias nos quais existem compromissos.
- Desenhar células vazias na última linha do calendário nos dias da semana após o último dia do mês.

4.1.3 Detalhes de Compromisso/Disponibilidade

A página **"`details.jsp`"** permite a visualização de todos os detalhes de um compromisso ou disponibilidade. À semelhança do formulário "`make_appointment.jsp`", este ficheiro aceita como argumentos no *URL* os parâmetros `load` e `loadDisp` com o valor do **ID** do compromisso ou disponibilidade pretendido. O compromisso ou disponibilidade com o **ID** referido é carregado da Base de Dados (por intermédio da função `fetchAppointment(int id)` da classe `AppointmentApi`) e colocado no objecto **`detail`** (da classe `Appointment`). Seguidamente a página é preenchida consoante os dados no *`Appointment`* `detail`.

No caso de se tratar de um compromisso, será mostrada a lista com todos os intervenientes e o seu estado. Caso o utilizador que está a visualizar o compromisso seja um dos participantes, o link para voltar poderá chamar o ficheiro "`confirmations.jsp`" passando como argumento o **ID** do compromisso, código do utilizador (`idDocente` ou `idAluno`) e a acção a tomar dependendo do estado actual do utilizador (ex: http://forge.fe.up.pt/agenda/confirmations.jsp?id=120-&idAluno=970509023&action=confirm). Existem quatro acções possíveis - Aceitar ou Rejeitar (se o estado do compromisso para o utilizador for pendente ou alterado); Confirmar (se o estado for Desmarcado ou Confirmado); ou Desmarcar (se o dono decidir cancelar o compromisso).

Se o utilizador que visualiza esta página for o dono do compromisso ou disponibilidade mostrado, existem ainda dois *links* para alterar compromisso ou disponibilidade (chamando "`make_appointment.jsp`"), ou elemina-lo (chamando "`confirmations.jsp`" com o parâmetro `action=cancel` no caso do compromisso, e "`del_disp.jsp`" no caso da disponibilidade).

Ficheiros adicionais - `confirmations.jsp`

Este ficheiro apenas chama a função **`changeUserState`** da classe `AppointmentApi` com os argumentos recebidos em `id`, `idDocente`, `idAluno`, e `action` de modo a actualizar o estado do utilizador. De seguida, o *browser* é redireccionado para a vista diária do utilizador.

4.1.4 Inserir/Editar Compromissos/Disponibilidades

Existe um formulário para a inserção e edição de compromissos e disponibilidades, implementado no ficheiro "`make_appointment.jsp`". Este formulário deverá ser chamado com argumentos passados no *URL* de modo a indicar a acção pretendida (inserir ou editar) e o compromisso ou disponibilidade a editar. Os dados do formulário são tratados por um *JavaBean* "`form_Appointment.java`", que estabelece a ligação entre o formulário em *HTML/JSP* e os JavaBeans da lógica de negócio.

Autenticação

Apenas o dono de uma agenda poderá inserir ou editar compromissos e disponibilidades, sendo que para usar este formulário, o utilizador tem que ter passado por um processo prévio de autenticação. No caso particular da edição, o utilizador apenas poderá editar um compromisso ou disponibilidade que lhe pertença, ou seja, que tenha sido ele a criar. Se qualquer um dos casos anteriores não se verificar, não será concedido acesso ao formulário.

Inserção

Para inserir um compromisso, o formulário poderá ser chamado sem argumentos (ex: http://-forge.fe.up.pt/agenda/make_appointment.jsp). No caso de uma disponibilidade, o formulário deverá ser chamado com o argumento `disp=true` (ex: http://forge.fe.up.pt/agenda/make_-appointment.jsp?disp=true).

Edição

Para editar um compromisso, terá que ser passado o argumento `load=id`, em que *id* é o **ID** do compromisso na Base de Dados (ex: http://forge.fe.up.pt/agenda/make_appointment.jsp?load=120). Para uma disponibilidade o processo é semelhante, com a diferença que se deverá usar o parâmetro `loadDisp` em vez de `load` (ex: http://forge.fe.up.pt/agenda/make_appointment.jsp?loadDisp=63).

Código JSP - `make_appointment.jsp`

O ficheiro "**`make_appointment.jsp`**" utiliza um *JavaBean* (classe `form_Appointment`) que é guardado na variável `formData`. Este objecto contem, para alem de todos os elementos relativos ao compromisso ou disponibilidade a tratar, métodos para validar dados e tratar diversas acções do formulário. Ao efectuar um *submit*, os resultados são passados para o *JavaBean* com a indicação da acção a executar (Adicionar interveniente, Remover interveniente e Inserir/Actualizar). Com esta informação o objecto `formData` devolve um *link* para a página para a qual o utilizador deverá ser redireccionado. Os dados são passados para o *Bean* como argumentos no *URL*, excepto ao entrar no formulário no modo de edição, em que ao incluir o parâmetro `load` ou `loadDisp` no *URL*, o compromisso ou disponibilidade é carregado da Base de Dados para o objecto `formData`. Os campos do formulário são preenchidos consoante os dados recebidos do *JavaBean*, e são gerados também outros campos escondidos com dados importantes para o tratamento do formulário (como a *flag* `disp` e os **ID**s do compromisso e dos intervenientes).

Outro aspecto de destaque é o tratamento dos utilizadores intervenientes de um compromisso. O formulário dispõe de um botão para acrescentar um interveniente que chama o módulo de pesquisa em "`search.jsp`". Os utilizadores podem pertencer a dois grupos distintos - docentes e alunos. Os identificadores na Base de Dados de cada utilizador participante são passados no *URL* nos *arrays* `itd[]` e `ita[]`, correspondendo aos códigos de docentes e alunos respectivamente. O formulário permite ainda que intervenientes sejam retirados da lista de intervenientes. Relativamente ao tratamento dos utilizadores participantes, a classe `form_Appointment` dispõe da funções `setItd(String[] it)` e `setIta(String[] it)` para receber os *arrays* correspondentes. Contudo, o servidor de *JSP/JavaBeans* utilizado não suporta a funcionalidade de receber *arrays* passados no *URL* directamente no *JavaBean*. Deste modo, resolvemos ir buscar estes *arrays* ao objecto `request` do *JSP*, executando posteriormente as funções referidas com os elementos recebidos.

A formatação do formulário também difere consoante se trate de um compromisso ou uma disponibilidade. Sendo que certos campos de um compromisso, não se aplicam nas disponibilidades. É o caso dos intervenientes e do local. Numa disponibilidade é também permitida a selecção de vários tipos, enquanto que um compromisso só pode ser de um tipo.

JavaBean - `form_Appointment.java`

Este *JavaBean* estabelece a ligação entre o formulário em "`make_appointment.jsp`" e as classes `Appointment` e `AppointmentApi` da camada da **Lógica de Negócio**. A funcionalidade principal da classe `form_Appointment` é receber os dados do formulário, e

adapta-los de modo a os colocar num objecto **`Appointment`**. O inverso também se verifica, e esta classe também devolve todos os campos do objecto `Appointment` com a formatação necessária para os campos do formulário. Este *Bean* também identifica a acção feita no formulário e disponibiliza um método que constrói o *link* para o qual o *browser* deve ser redireccionado de modo a executar essa acção. As acções e redireccionamento é feito da seguinte forma:

- **Adicionar Interveniente**: O utilizador é redireccionado para o módulo de pesquisa em "`search.jsp`", de modo a seleccionar o utilizador pretendido.
- **Remover Intervenientes**: O *browser* chama o ficheiro "`remove.jsp`" que retira os utilizadores seleccionados da lista de intervenientes do compromisso.
- **Inserir/Actualizar**: Se não ocorreram erros na inserção dos dados, o utilizador volta para a vista diária da sua agenda ("`diaria.jsp`").
- **Página de Erro**: Caso o utilizador insira dados inválidos, estes serão listados na página "`validate.jsp`".

No caso da inserção/actualização, antes do redireccionamento devem ser executadas as funções `validate()` e `update()` de modo a validar os dados a inserir, e inserir ou actualizar os dados na Base de Dados. Na validação de dados é verificado que o título do compromisso ou disponibilidade foi preenchido; que numa disponibilidade foi escolhido pelo menos um tipo; e que a duração de um compromisso ou disponibilidade não se prolonga para além da meia-noite. A inserção e actualização do `Appointment` é feita através dos métodos `insertAppointment` e `updateAppointment` da classe **`AppointmentApi`**.

Ficheiros adicionais - `remove.jsp`

Este ficheiro é usado para retirar utilizadores da lista de intervenientes. Devem ser acrescentados ao *URL* todos os parâmetros do formulário e ainda o parâmetro `backLnk` com o nome do ficheiro ao qual se deve voltar (neste caso "`make_appointment.jsp`") e também um *array* `itName[]` com os nomes dos intervenientes a remover. Os utilizadores participantes num compromisso são referenciados no formulário pelo seu código (*arrays* `itd[]` e `ita[]`). Contudo, o utilizador selecciona os intervenientes pelo seu nome. Por este motivo, este ficheiro verifica para cada código em `itd` e `ita` se o nome correspondente se encontra no *array* de nomes a retirar (`itName[]`). De seguida o *browser* é redireccionado de volta à página em `backLnk`, incluindo todos os argumentos passados no *URL* excepto os parâmetros `backLnk`, `del` (usado para chamar este ficheiro), `itName` e os elementos de `itd[]` e `ita[]` que foram retirados.

Ficheiros adicionais - `validate.jsp`

Caso se verifique alguma inconsistência nos dados, o utilizador será notificado de quais os erros cometidos. À semelhança do ficheiro anterior, o *URL* deverá incluir todos os parâmetros do formulário e ainda o parâmetro `backLnk`. Os erros detectados também devem ser acrescentados num *array* `error[]`.

4.1.5 Pesquisa de Utilizadores

Por vezes, o utilizador pode necessitar de referenciar outro utilizador. Como não seria muito prático para o utilizador introduzir o código do utilizador pretendido, foi desenvolvido um formulário para permitir a pesquisa de docentes e alunos pelo nome. Esta pesquisa é efectuada pelo ficheiro **"`edit_type.jsp`"**, usando o *JavaBean* **`UserSearch`** (objecto `searchData`). Este módulo deverá ser chamado com o parâmetro `backLnk` no *URL*, podendo ser acrescentados quaisquer outros parâmetros os quais serão adicionados ao valor de `backLnk` para construir os *links* para voltar à página pretendida. Os resultados da pesquisa são devolvidos no objecto `searchData` e são listados os nomes de todos os utilizadores do resultado, possuindo cada um o *link* construído anteriormente mais o identificador do utilizador listado. Este identificador é colocado nos parâmetros `itd[]` e `ita[]` quando o módulo é chamado do formulário de inserção/alteração de compromissos, ou nos parâmetros `idDocente` e `idAluno` quando a origem é outra.

4.1.6 Gestão de Tipos de Compromissos

O **administrador** do sistema dispõe de um formulário que permite a introdução de novos tipos de Compromissos. Este formulário está implementado no ficheiro **"`edit_type.jsp`"**. O formulário é relativamente simples e contem apenas caixas de texto com todos os tipos existentes, permitindo a alteração dos seus nomes, e também uma outra caixa de texto para acrescentar um novo tipo. Os dados são tratados através do *JavaBean* **`TypeApi`**.

4.2 Lógica de negócio

A camada da lógica de negócio é caracterizada por efectuar a ligação entre a camada do interface e da base de dados. Esta ligação é normalmente acompanhada por uma componente algorítmica, de modo a que os dados obtidos da camada a montante, sejam perceptíveis pelo interface. A descrição que se vai seguir não pretende ser muito exaustiva, visto que todas as classes tratadas aqui estão documentadas com a ferramenta *JavaDoc*, de modo que pode ser consultada para algum esclarecimento.

4.2.1 ApiBD

Esta classe existe para facilitar o acesso de outras classes à base de dados. Sempre que seja necessário executar uma *query SQL* é necessário criar um objecto desta classe. Ao criá-lo é automaticamente criada uma conecção à base de dados especificada pelo *uri*, para depois executar uma inicializações na própria base de dados, respeitante ao formato da data e da ordenação dos registos.

Esta classe disponibiliza três métodos com as seguintes funcionalidades:

- executar perguntas *SQL* de consulta de dados,
- executar expressões *DML* (*DELETE, INSERT* e *UPDATE*),
- desligar a ligacao existente.

Dos métodos listados apenas os dois primeiros têm valores de retorno, que são resectivamente:

- um objecto do tipo `ResultSet` que contém o conjunto de resultados obtidos,
- um inteiro indicando o número de linhas que foram afectadas.

Neste método há o cuidado especial detectar todas as situações de erro. Nestes casos é lançada uma execpção para a classe que invocou o método.

4.2.2 AppointmentApi

O objectivo principal desta classe é fornecer à camada superior, a do interface, a lista de compromissos e de disponibilidades de determinado utilizador, e caso este seja um docente, fornece também as suas aulas e exames. Também é possível encontrar métodos para a gestão de compromissos. Esta classe *extends* uma outra, `AppCore`, onde se encontram os acessos à base de dados. Toda a informação de determinado compromisso ou disponibilidade é representada através da classe `Appointment`, a qual tem os campos necessários para a definição de um registo da tabela.

A lista anteriormente referida é elaborada de acordo com a vista que está a ser gerada na interface. Para isso existem três métodos distintos: `getDayAppointments(User owner,`

`GregorianCalendar day)`, `getWeekAppointments(User owner, GregorianCalendar day)` e `getMonthAppointments(User owner, GregorianCalendar day)`. Todos estes métodos têm como valores de entrada a identificacao de um utilizador e um determinado dia. Os dois primeiros métodos têm como saída um vector de objectos Appointment, sendo que o método para a vista mensal apenas fornece um array de booleanos.

Todos estes métodos fazem a verificação se o utilizador em causa é um docento para que, como já foi dito, pesquisar as suas aulas e exames. De seguida, independentemente do utilizador, obtém os seus compromissos e disponibilidades. Os métodos utilizados para estas operações encontram-se na classe `AppCore`. Neste métodos nada mais é feito do que efectuar uma ou um conjunto de *querys SQL* de modo a obter os dados pretendidos, e mapear estes em vários objectos da classe `Appointment`.

Este mapeamento não existe quando se deseja a informação para determinado mês, já que, na camada de interface apenas é disponibilizada a informação se existe ou não algum tipo de compromisso. Assim, para o método `getMonthAppointment()` devolve um *array* de *boolean*, em que cada posição do mesmo indica se existe algum tipo de compromisso nesse dia ou não.

Na classe `AppointmentApi` ainda podemos encontrar métodos para fazer a gestão dos compromissos. Esta gestão engloba criar novos compromissos, modificar ou apagar os já existentes. Este conjunto de métodos, que têm como parâmetro de entrada um objecto da classe `Appointment`, nada mais fazem do que actualizar a base de dados de acordo com os dados presentes no argumento de entrada.

4.2.3 Menu

Este bean é responsável por gerar o código *HTML* de modo a construir o menu de opções que é disponibilizado ao utilizador quando este está a utilizar a aplicação. Independentemente de o menu aparecer ao não, tem de haver uma invocação desta classe por parte dos ficheiros jsp, para que seja gerado uma porção de código indispensável à correcta visualização da página. Para cada opção que se deseje ter no menu basta invocar o método correspondente. Ou criar um novo método para uma outra opção é necessário especificar o link para onde se pretende redireccionar o utilizador e o texto a aparecer na página. Com estas duas variáveis é criado o código *HTML* para este menu. Existe ainda um método que em vez de gerar uma nova opção gera "placard" com os vários avisos a mostrar ao utilizador. Este método faz uso das classes que a seguir se descrevem para obter a lista de compromissos "afixar".

4.2.4 AppBoard

Como já foi dito, na aplicação desenvolvida é possível avisar o utilizador da marcação de novos compromissos na sua agenda. A verificação da existência de compromissos que determinado utilizador necessita de verificar, devido a alterações, cancelamentos e adições, é da responsabilidade desta classe. Os dois métodos presentes nesta classe são bastantes semelhantes, sendo que no primeiro é vocacionado para "placar" que aparece nas várias vistas das agendas. O segundo é mais vocacionado para a página onde são visualizados os compromissos que estão em situações que necessitem de ter o conhecimento do utilizador.

O acesso há base de dados é feito a partir da classe `BoardCore`. Aqui o que apenas é feito é obter os dados de cada compromisso e mapeá-los em objectos `Appointment`.

4.2.5 TypeApi

Esta classe fornece os mecanismos necessários à gestão dos vários tipos de compromisso que podem existir. Essencialmente esta classe é usada pela parte do interface que corresponde ao administrador, já que este é o único que tem poderes para alterar os vários tipos possíveis. No formulário criado para o efeito, já foi referido que aparecem todos os tipos listados. Quando o administrador por fim faz a actualização dos dados, é passado para o método `setTypes(Stringg[] types]` um *array* de *Strings*, contendo cada uma um valor para o tipo. Esse *array* é tratado, de modo a que algum valor que tenha sido actualizado efectuar essa operação na base de dados.

Por fim, são também disponibilizados um conjunto de métodos de modo a que em várias partes da camada de interface possam saber toda a informação de um determinado tipo, através da indicação de um campo a partir do qual seja possível obter uma referência única. Todos estes métodos têm como resultado um objecto da classe `Type`, onde é mapeada toda a informação de um determinado tipo.

4.2.6 User

O bean `User` tem um conjunto de atributos que correspondem à campos correspondentes à tabela **Utilizadores** na base de dados. Tem também um pequeno conjunto de métodos que permitem de um modo transparante para as restantes classes obter toda a informção de determinado utilizador, bastando para isso ter um **id** do mesmo. Estes métodos verificam se esses dados já são conhecidos ou não. Se sim, são automaticamente retornados, se não é feita uma *query SQL* de modo a obter esses mesmos dados. Existem ainda um pequeno método para verificar se dois objectos do tipo `User` são iguais. Este método substitui o método homónimo da classe `Object`, já que faz a comparação através dos **id**'s dos utilizadores, em detrimento de fazer através de todos os campos da classe. Assim consegue-se que um utilizador que esteja definido apenas por este campo seja comparável com um outro que tenha todos os seus atributos preenchidos.

4.2.7 UserSearch

Como já foi visto no capítulo da interface, é disponibilizado ao utilizador uma pequena pesquisa de modo a que este possa seleccionar uma pessoa para ver a sua agenda ou para acrescentar à lista de intervenientes de um compromisso que está a construir. Assim temos que dependendo da escolha feita pelo utilizador entre pesquisar entre alunos ou docentes é feita uma pesquisa na tabela correspondente da base de dados. De salientar que esta pesquisa não é sensível a maíusculas, sendo que é irrelevante introduzir “pascoal”ou “PasCOAL”.

4.2.8 MyCalendar

Esta classe faz todo o conjunto de operações necessárias à renderização da janela de `pop-up` onde é visualizado um calendário. Existe um conjunto de funções `settable` para os dados necessários à contrução do calendário, e para poder obter um conjunto de dados relativos a datas. Existem dois métodos distintos para a geração de código HTML, que são o *drawLinks()* e *drawMonth()*. Como o próprio nome sugere, o primeiro "desenha" os botões de navegação do calendário. O segundo método elabora o mês, sendo que poderá ou não no início de cada linha a indicação da semana correspondente. Todos dos dias estão `linkados` para um campo de um formulário ou mesmo para uma página. Se a indicação da semana for visível, os números correspondentes encontram-se também `linkados`. Por fim esta classe fornece alguns pequenos métodos de verificação de datas, essenciais para o resto da aplicação.

4.3 Base de Dados

Nesta secção serão descritas as tabelas e respectiva informação que é utilizada para mostrar e inserir diferentes tipos de informação para diferentes utilizadores.

4.3.1 Agenda Web

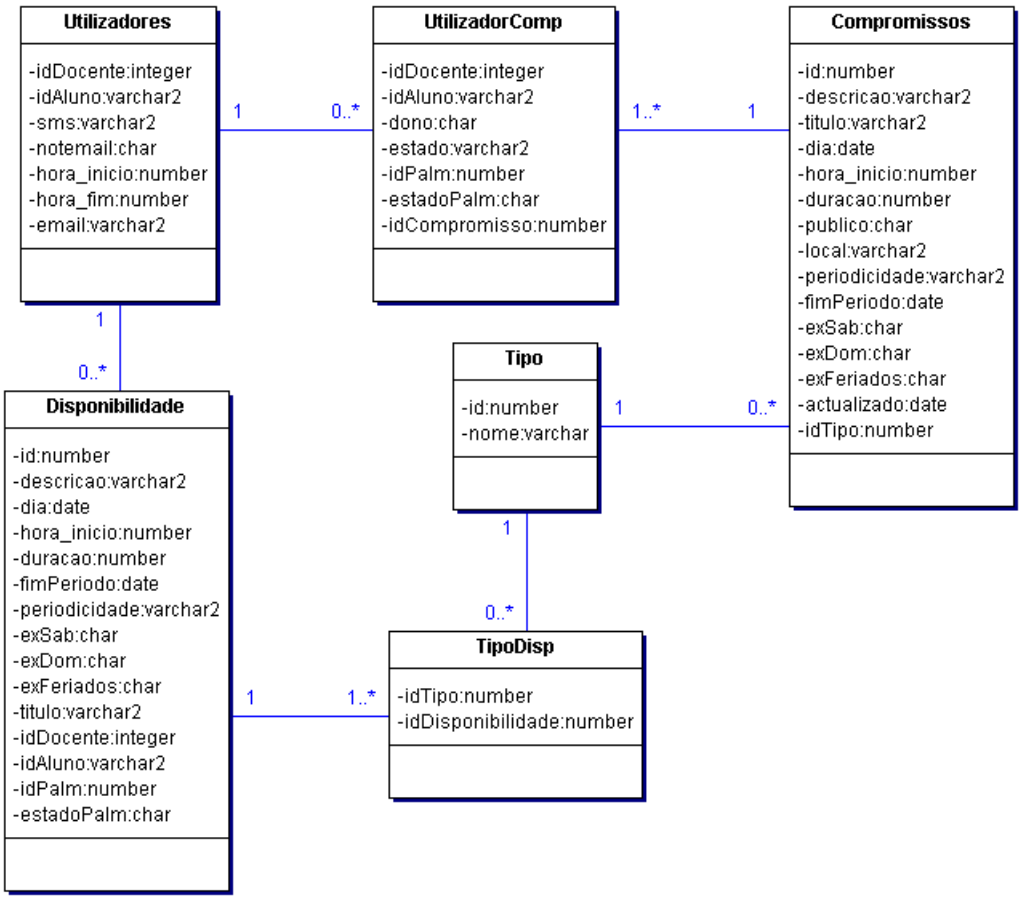

Figura 4.1: Modelo relacional da base de dados da **Agenda Web**.

As tabelas que fazem parte da base de dados Agenda Web contêm informação que permitem tanto a alunos como docentes inserir, modificar e eliminar compromissos e disponibilidades.

Esta informação permite que os utilizadores possam configurar as suas agendas ao nível do seu horário de trabalho ou notificações de ocorrência de compromissos. Permite ainda que o utilizador possa ser notificado quando se dão marcações, alterações ou rejeições a compromissos marcados pelo próprio ou por outros utilizadores.

A informação contida nesta base de dados permite ainda controlar a informação existente que deve ser descarregada para o Palm do utilizador.

De seguida, passa-se a descrever com maior pormenor o tipo de informação contida nas tabelas representadas na figura 4.1.

Utilizadores

A tabela **Utilizadores** guarda a informação que permite configurar a agenda do utilizador. Os campos **hora_inicio** e **hora_fim** permitem definir o horário de trabalho do utilizador. Os campos **email** e **sms** guardam a informação do email e número de telemóvel onde o utilizador receberá as notificações. O utilizador receberá sempre notificações sobre alterações aos compromissos em que é interveniente. No entanto, pode também querer receber notificações da aproximação da ocorrência de compromissos. Para tal, o campo **notemail** indica se o utilizador deseja receber estas notificações. O campo **sms** se estiver preenchido indica automaticamente que o utilizador deseja recber notificações por este meio.

Nesta tabela existem ainda dois campos separados para o número de docente e de aluno (iguais aos existentes no SiFEUP) dado que estes campos têm tipos de dados distintos.

Tipo

Esta tabela define os tipos de compromissos ou disponibilidades inseridas pelo administrador. Contém apenas os campos **id** e **nome**, correspondendo a um identificador e ao nome do tipo, respectivamente.

Disponibilidade

A tabela **Disponibilidade** guarda a informação necessária para marcar uma disponibilidade.

Esta tabela tem um campo **id** que é um número identificador da disponibilidade. Cada disponibilidade é obrigatoriamente caracterizada por um título, dia, hora de início e uma duração (campos **titulo**, **dia**, **hora_inicio**, **duracao**, respectivamente). Uma disponibilidade pertence sempre a um utilizador (**iddocente** ou **idaluno**) e é caracterizada por um ou mais tipos.

Uma disponibilidade pode ainda ser, opcionalmente, caracterizada por uma descrição (**descricao**) e uma periodicidade. Uma disponibilidade pode repetir-se diária, semanal, mensal ou anualmente (**periodicidade**) até uma determinada data escolhida pelo utilizador (**fimperiodo**). O utilizador pode ainda optar por repetir a disponibilidade até uma determinada data, mas excluindo os dias em que seja sábado, domingo ou feriado (campos **exsab**, **exdom** e **exferiados**, respectivamente).

Dado que uma disponibilidade pode ter vários tipos, por exemplo, aula de dúvidas e reunião, é necessária uma tabela intermédia entre esta tabela e a tabela **Tipo** de forma a implementar a relação “Muitos para Muitos”. Esta tabela chama-se **TipoDisp**.

Os campos **idPalm** e **estadoPalm** dizem respeito à possibilidade de se poder descarregar as disponibilidades para a agenda do Palm. O campo **estadoPalm** indica se a disponibilidade é nova ou não e se já foi descarregada ou não para o Palm. O campo **idPalm** só fica preenchido quando a disponibilidade é descarregada pela primeira vez para o Palm e corresponde ao **id** que fica registado na base de dados da agenda do Palm.

TipoDisp

A tabela **TipoDisp** é a tabela intermédia entre **Disponibilidade** e **Tipo** e contém apenas os identificadores do tipo e da disponibilidade.

Compromissos

A tabela **Compromissos** guarda a informação de um compromisso marcado por um utilizador.

O campo **id** é um número identificador do compromisso. À semelhança de uma disponibilidade, um compromisso tem exactamente os mesmos campos obrigatórios (**titulo**, **dia**, **hora_inicio** e **duracao**). Um compromisso refere-se apenas a um único tipo existindo, por isso, o campo **idtipo** que referencia a tabela **Tipo**.

Um compromisso pode ainda ser, opcionalmente, caracterizado por um local onde vai ocorrer e pela sua visibilidade para os outros utilizadores (público ou privado) (campos **local** e **publico**, respectivamente).

Mais uma vez, à semelhança de uma disponibilidade, um compromisso pode ser caracterizado por uma periodicidade, existindo por isso os campos **periodicidade**, **fimperiodo**, **exsab**, **exdom** e **exferiados**, com o mesmo significado que na tabela **Disponibilidade**.

Um compromisso tem um dono, que corresponde à pessoa que marcou o compromisso, e pode ter zero ou mais intervenientes, para além do dono. Desta forma, para implementar a relação "Muitos para Muitos" entre esta tabela e a tabela **Utilizadores**, existe a tabela **UtilizadorComp**.

O campo **actualizado** indica a data da última actualização do compromisso em função do estado em que o compromisso está para todos os intervenientes. Por exemplo, quando todos os utlizadores aceitam o compromisso, esta data é actualizada para a data em que o último interveniente aceitou o compromisso.

UtilizadorComp

Esta tabela permite guardar os intervenientes num compromisso, incluindo o dono. O utilizador tanto pode ser um aluno como um docente, daí os dois campos **iddocente** e **idaluno**.

O campo **dono** indica se esse utilizador é ou não o dono do compromisso marcado e o campo **estado** indica o estado em que o compromisso está para um determinado interveniente no compromisso. Este campo **estado** pode tomar os seguintes valores: **Pendente** quando determinado utilizador é interveniente num compromisso e não foi ele que marcou o compromisso; **Aceite** quando um interveniente aceita um compromisso ou o utilizador é o próprio dono do compromisso; **Definitivo** quando todos os intervenientes aceitam o compromisso; **Alterado** quando o dono do compromisso altera o compromisso; **Rejeitado** quando um dos intervenientes rejeita o compromisso; **Desmarcado** quando um interveniente rejeita o compromisso, todos os outros intervenientes passam a ver o compromisso neste estado e **Cancelado** quando todos os intervenientes têm conhecimento que o compromisso não se realizará.

Os campos **idPalm** e **estadoPalm** representam a mesma função que na tabela **Disponibilidade**.

4.3.2 SiFEUP

São usados vários esquemas relacionais do SiFEUP para mostrar informação distinta nas agendas dos docentes. São usadas as tabelas relativas aos horários, de forma a ser possível mostrar na agenda as aulas existentes num determinado semestre, e as tabelas relativas aos exames, de forma a mostrar os exames das disciplinas que o docente lecciona.

Aulas

Para determinar as aulas que um docente lecciona são usadas as seguintes tabelas: **C_Docentes**, **C_Docentes_C_Docentes**, **C_Turmas**, **C_Salas**, **Aulas**, **Tipos_de_Aula**, **Cadeiras**, **Ocorrencias**, **Periodos** e **Versoes**.

Para um dado semestre correspondente à data actual, são retiradas para um determinado docente as aulas que este lecciona. É especificamente retirada a disciplina, a turma, a sala e a hora e duração a que a aula corresponde.

Exames

Para determinar os exames das disciplinas que um determinado docente lecciona são usadas as seguintes tabelas: **Cadeiras**, **Reservas_Exame**, **Exames_Salas_Provisorio**, **Chamadas**, **Ocorrencias**, **Tipos_de_Aula**, **Distribuicoes_de_Servico**, **Docentes** e **Periodos**.

São apenas mostrados os exames das disciplinas que o docente lecciona e não os exames em que o docente é apenas vigilante.

Para cada exame é mostrada a sua hora de início, duração, sala, disiciplina, chamada e número de alunos previstos.

5 Funcionalidades Implementadas

Face aos requisitos funcionais, referidos no "Relatório de Especificação de Requisitos", foram implementadas as seguintes funcionalidades:

- Existe uma agenda para cada elemento da organização, sendo este o dono da agenda.
- A agenda é constituída por compromissos públicos, compromissos privados e disponibilidades
- Os compromissos são caracterizados por um só tipo, enquanto que as disponibilidades podem ser caracterizadas por vários tipos simultaneamente.
- Os compromissos/disponibilidades têm um título, descrição, hora de início e duração.
- O compromisso privado (tipo e descrição) só está visível para o dono da agenda. Para os outros elementos da organização, este compromisso tem apenas a indicação que é privado.
- Existe a possibilidade de se marcar um compromisso (público ou privado) em que participem vários intervenientes.
- O dono da agenda pode ter compromissos que se sobreponham.
- Só o dono da agenda pode marcar compromissos e disponibilidades.
- A agenda está disponível a todos os colaboradores da organização para visualização.
- Fica registado quem marcou um compromisso e quando.
- O administrador do sistema pode definir os tipos de compromissos possíveis.
- Utilizam-se intervalos de tempo constantes de 30 minutos. Assim, cada compromisso tem a duração mínima de 30 minutos, e a duração máxima de um múltiplo de 30 minutos.
- É possível serem marcados acontecimentos que acontecem periodicamente (por exemplo, a aula que ocorre todas as segundas-feiras).
- O interface da aplicação é acessível a partir de um *browser* para *Internet*.
- A aplicação utiliza/importa dados presentes no SiFEUP, nomeadamente os horários dos docentes.

- A aplicação avisa o dono de uma agenda, quando existir um novo compromisso marcado no qual ele é um interveniente, ou quando um compromisso marcado for aceite, rejeitado ou cancelado por outro interveniente do compromisso.
- O dono da agenda pode eliminar da sua agenda tanto os compromissos como as disponibilidades.
- O interface da aplicação segue o aspecto geral do interface do SiFEUP para ser possível uma futura integração no mesmo.
- É possível ao autor de determinada marcação efectuar alterações de vários parâmetros sobre essa marcação (horário, tipo, etc). Quando estas alterações são feitas, se se tratar de um compromisso com mais intervenientes, estes são avisados de tais alterações.

6 Funcionalidades Não Implementadas

Devido à escassez de tempo, certos requisitos não puderam ser cumpridos. Deparamos também com outras funcionalidades que se tornaram redundantes e problemáticas, não se justificando a sua implementação. Contudo tencionamos corrigir estas eventuais deficiências num futuro próximo.

Requisito 6

"É possível ao dono da agenda marcar um compromisso na agenda de outra pessoa, desde que na mesma altura não exista já um compromisso. No entanto, na sua própria agenda, o dono pode ter compromissos que se sobreponham."

Este requisito não foi inteiramente cumprido. O dono da agenda pode ter compromissos que se sobreponham, contudo, achamos pouco conveniente que um utilizador marque um compromisso na agenda de outra pessoa. Esta funcionalidade é desnecessária visto que o utilizador pode marcar o compromisso incluindo a outra pessoa como interveniente. Esta será então notificada do compromisso, e poderá aceitar ou recusá-lo. Caso o compromisso seja aceite por todos os intervenientes, este constará nas agendas de cada um dos intervenientes.

Requisito 12

"É possível serem marcados acontecimentos que acontecem periodicamente (por exemplo, a aula que ocorre todas as segundas-feiras)."

A marcação de compromissos e disponibilidades periódicas foi implementada, contudo a sua visualização na agenda é limitada. Um acontecimento deste tipo mostrado apenas no dia para o qual foi definido, não sendo visíveis na agenda as repetições periódicas do acontecimento.

Requisito 14

"Deve ser possível à aplicação utilizar/importar dados presentes no SiFEUP, nomeadamente os horários."

Este requisito foi cumprido mas apenas parcialmente. A aplicação importa dados do SiFEUP, nomeadamente dados sobre os docentes e alunos e os horários das aulas e dos exames (no ponto de vista dos docentes). Ficou em falta a importação dos horários de aulas e exames dos alunos.

Requisito 18 (não mínimo)

"Deve ser possível a partir da aplicação exportar os dados presentes numa agenda para um Palm."

Este requisito faz parte do trabalho para fora, encomendado à empresa Insane Software que faltou no cumprimento desta funcionalidade.

Requisito 19 (não mínimo)

"A aplicação poderá, opcionalmente, notificar o dono da agenda, da proximidade de ocorrência de um compromisso, através de *e-mail* ou *sms*."

Este requisito também foi parte do trabalho à empresa Insane Software, que cumpriu parcialmente esta funcionalidade. Contudo, devido à escassez de tempo este trabalho ainda não foi integrado na aplicação final.

7 Conclusão

A questão da integração da aplicação no SiFEUP foi uma das preocupações tanto do desenho da arquitectura, como da implementação. A nível da interface, tentou-se adoptar ao máximo o aspecto visual do SiFEUP. Para isso, foi consultado o designer do CICA para que as normas de imagem fossem seguidas o mais possível.

A nível de conteúdos, foi necessário usar a base de dados do SiFEUP para se poder obter a informação dos utilizadores da aplicação (docentes e alunos), assim como a informação sobre as aulas e datas de exames dos docentes. Esta integração foi muito facilitada pelo facto de um dos elementos da 3 ao Quadrado, por trabalhar no CICA, ter um acesso mais directo às pessoas que trabalham ao Sistema de Informação da FEUP.

A implementação deste projecto tomou um rumo diferente do planeado e especificado inicialmente. A impossibilidade de usar os *EJB* como tecnologia de suporte obrigou a uma reestruturação da arquitectura. Este processo introduziu mudanças a nível da base de dados, que passou a ter de ser implementada de raiz, e consequentemente a nível da lógica de negócios, com a necessidade de uma camada de ligação e acesso à base de dados. Este facto levou a alguns atrasos, e à data deste documento o código fonte não se encontra nas melhores condições, sendo a nossa intenção corrigir brevemente as eventuais deficiências na aplicação. Apesar disto, a interface não sofreu alterações. A tecnologia usada mostrou-se bastante flexível, visto ter como base o *Java*, que permitiu uma divisão bem definida entre a camada de interface (*JSP*) e a lógica de negócio (*JavaBeans*). A base de dados *Oracle* não apresentou problemas.