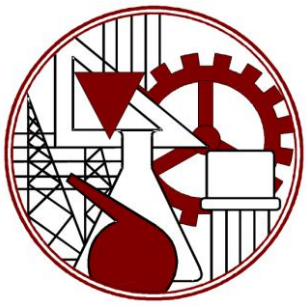
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

DEPARTAMENTO DE ENGENHARIA DE ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES



Plataforma de Desenvolvimento de Aplicações *Web* Orientadas a Mensagens

Rui Manuel Vieira Dias

(Licenciado)

Trabalho de projecto realizado para obtenção do grau de Mestre em Engenharia Informática e de Computadores

Orientador:

Professor Doutor Luís Filipe Graça Morgado, ISEL

Júri:

Presidente: Professor Adjunto Pedro Pereira, ISEL

Vogais:

Professor Doutor Luís Filipe Graça Morgado, ISEL

Professor Doutor Porfírio Pena Filipe, ISEL

Setembro de 2010

Resumo

O projecto apresentado neste relatório consiste na implementação de uma plataforma de desenvolvimento de aplicações *Web* orientada a mensagens, capaz de facilitar a construção de páginas *Web* e de automatizar o acesso a dados com base nos requisitos do problema.

O foco da solução é baseado nos requisitos que forem definidos, de modo a desenvolver uma aplicação *Web* que responda a esses requisitos. A estratégia adoptada visa reduzir o acoplamento e aumentar a coesão dos módulos funcionais.

Para reduzir o acoplamento e aumentar a coesão dos módulos funcionais separou-se a apresentação dos dados, permitindo assim que o desenvolvimento e os testes sejam realizados independentemente um do outro.

O desenvolvimento é baseado em normas padronizadas (*standards*) de modo a facilitar a integração com outras tecnologias. Para o desenvolvimento do trabalho foi adoptada uma metodologia ágil. A gestão de projecto foi realizada com recurso a uma plataforma de gestão de projecto, que permitiu registar e organizar todas as informações e tarefas realizadas ao longo do projecto.

Após a conclusão da plataforma e como caso de estudo, foi realizada uma aplicação baseada num cenário real de gestão académica (Portal Académico) para aferir os conceitos envolvidos.

Palavras-Chave

Análise de Requisitos, Arquitectura de Aplicações *Web*, Arquitectura de Acesso a Dados, *ACL*, Metodologia de Desenvolvimento Ágil.

Abstract

The project presented in this document is the implementation of a message-oriented platform for Web application development to facilitate the construction of web pages and automate data access based on the requirements of the problem.

The focus of the solution is based on the requirements that were defined, in order to develop a web application that meets these requirements. The proposed strategy aims to reduce the coupling and increase the cohesion of functional modules.

To reduce the coupling and increase the cohesion of the functional modules the presentation was separated from data. Thereby, the development and testing can be achieved independently of one another.

The development is based on standards to facilitate integration with other technologies. For the development of the work an agile methodology was adopted. The project management was done using a project management platform, which supported the recording and organization of all the information and tasks performed throughout the project.

Upon completion of the platform and as a case study, a prototype application was developed based on a real scenario of academic management (Academic Portal) to evaluate the concepts involved.

Keywords

Requirements Analysis, Web Application Architecture, Data Access Architecture, ACL, Agile Development Methodology.

Agradecimentos

Quero agradecer ao Instituto Superior de Engenharia de Lisboa (ISEL), em particular ao Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores (DEETC), por todos os meios que colocou à disposição para a minha formação.

Ao professor Luís Morgado, orientador deste projecto, que desde início manifestou uma grande disponibilidade e pelo apoio prestado.

Gostaria também de agradecer a todos os meus professores que ao longo destes anos contribuíram para a minha valorização pessoal e profissional.

A todos os meus colegas e amigos, pela sua pronta disponibilidade, incentivo, apoio e pelas inúmeras discussões de ideias que permitiram dispor de vários pontos de vista.

À equipa de informática da Unidade Complementar de Informática do ISEL, ao Gabinete de Apoio ao Aluno, ao Gabinete de Avaliação e Qualidade pelos esclarecimentos prestados, simpatia e disponibilidade demonstradas.

Para finalizar, um agradecimento muito especial aos meus pais e à minha família pela excepcional paciência, incentivo e apoio dado ao longo desta etapa académica, nunca permitindo baixar os braços ou desistir de enfrentar o obstáculo seguinte.

Um agradecimento muito especial à minha namorada Cláudia, por toda a paciência demonstrada, pelo apoio incondicional, pelos incentivos, e por estar sempre presente para dar uma palavra amiga. Ela foi fundamental para ultrapassar muitas das dificuldades que surgiram ao longo da realização deste projecto.

Índice

Índice de Figuras

Índice de Exemplos

Índice de Tabelas

Lista de Acrónimos

ACL	*Agent Communication Language*
AJAX	*Asynchronous Javascript And XML*
API	*Application Programming Interface*
ASP	*Active Server Pages*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
FIPA ACL	*Foundation for Intelligent Physical Agents - Agent Communication Language*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
JSON	*Java Script Object Notation*
JSP	*Java Server Pages*
LDAP	*Lightweight Directory Access Protocol*
MVC	*Model-View-Controller*
ODBC	*Open Data Base Connectivity*
PC	*Computador Pessoal*
PHP	*PHP: Hypertext Preprocessor (*originalmente *"Personal Home Page")*
SGA	*Sistema de Gestão Académica*
SQL	*Structured Query Language*
SOA	*Service-Oriented Architecture*
SSL	*Secure Sockets Layer*
URL	*Uniform Resource Locator*

WML	*Wireless Markup Language*
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*

1. Introdução

O desenvolvimento de aplicações para a *World Wide Web*, também conhecida como *Web*, é realizado de diferentes formas, umas mais complexas que outras, sendo um dos principais objectivos garantir que esse desenvolvimento seja realizado de forma eficiente, de modo a dar origem a aplicações no mínimo tempo possível. Tendo em conta esta premissa, pretende-se aumentar a rapidez de construção de aplicações *Web*[1], criando para isso uma plataforma que permita automatizar esse processo.

A plataforma desenvolvida permite acelerar a construção de aplicações *Web* através da interpretação do conteúdo de mensagens originadas a partir de requisitos.

O desenvolvimento é baseado em normas padronizadas (*standards*) de modo a facilitar a integração com outras tecnologias. Para o desenvolvimento do trabalho foi adoptada uma metodologia ágil.

Após a conclusão da plataforma e como caso de estudo foi realizada uma aplicação baseada num cenário real de gestão académica (Portal Académico) para aferir os conceitos envolvidos.

1.1. Âmbito

Este projecto foi desenvolvido no âmbito da unidade curricular de Dissertação/Projecto, do Mestrado em Engenharia Informática e de Computadores do Instituto Superior de Engenharia de Lisboa (ISEL), sendo analisados, no âmbito deste, os requisitos e funcionalidades do sistema de gestão académica do ISEL e as aplicações que lhe dão suporte (*back office* e *front office*). Pretende-se que a plataforma suporte a construção de aplicações *Web* que permitam responder aos requisitos pretendidos.

[1] **Aplicação *Web*** - Por definição, é algo mais do que apenas um *website*. É uma aplicação cliente / servidor que utiliza um navegador *Web* como o seu programa cliente, e executa um serviço interactivo através da ligação com um servidor sobre a *Internet* (ou *Intranet*) [1]. Uma aplicação *Web* apresenta conteúdos de forma dinâmica com base em pedidos do utilizador.

1.2. Motivação

A área do desenvolvimento de aplicações *Web* não pára de evoluir, estando sempre a aparecer novas soluções. A grande concorrência entre as várias soluções tem levado a que estas se tornem cada vez mais complexas, pelo que a sua utilização tem associado um custo de aprendizagem significativo. Ou seja, muitas das soluções exigem períodos de formação e conhecimentos específicos em determinadas tecnologias o que origina um período de adaptação à nova solução.

O que se pretende com este projecto é aumentar a rapidez de construção de aplicações *Web* utilizando uma plataforma que requer um custo de aprendizagem relativamente pequeno. Pretende-se que a utilização da plataforma seja intuitiva e acessível para o utilizador, bastando seguir os passos necessários para o seu funcionamento. Para isso, a plataforma foi desenvolvida de modo a minimizar o conhecimento das tecnologias, sendo da responsabilidade da plataforma a construção dinâmica das aplicações *Web*.

Tendo em conta algumas limitações do actual Portal Académico do ISEL achou-se interessante construir um novo portal utilizando a nova plataforma. No novo portal pretende-se promover uma experiência eficaz e de qualidade na interacção com os utilizadores.

1.3. Formulação do problema

No âmbito das aplicações *Web* é possível identificar dois tipos principais de *websites*: aqueles com conteúdo estático e aqueles com conteúdo gerado dinamicamente. Este trabalho foca-se na apresentação de conteúdos dinâmicos, pretendendo-se automatizar o modo de apresentação de conteúdos deste tipo. Pretende-se com esta solução reduzir o tempo de desenvolvimento permitindo com isso que haja um foco maior na lógica de negócio, nomeadamente, na análise de requisitos funcionais e de dados. Sendo os requisitos funcionais e de dados o núcleo de uma aplicação, a ideia é definir muito bem esses requisitos. Havendo requisitos bem definidos, compete à plataforma realizar todo o trabalho necessário para corresponder aos requisitos.

Actualmente o ISEL possui uma aplicação que representa a aplicação *front office* de um modelo de dados relacionado com gestão académica (Portal Académico). Tendo em

conta muitas limitações desta aplicação, surgiu a ideia de implementar um novo Portal Académico utilizando a nova plataforma.

1.4. Objectivos

O objectivo principal deste projecto é implementar uma plataforma de desenvolvimento de aplicações *Web* que permite:

- Vertente *Web*:
  - Construir páginas de interacção com o utilizador a partir dos requisitos do problema;
  - Construir dinamicamente vários tipos de elementos *HTML* de forma a tornar a aplicação mais rica;
  - Validar automaticamente os elementos de um formulário consoante o tipo de dados.
- Vertente de Acesso a Dados:
  - Automatizar o acesso a dados com base nos requisitos do problema.

Para atingir estes objectivos é necessário uniformizar a interacção entre os vários componentes da plataforma de modo a que todos eles consigam comunicar entre si de forma eficiente. Para além disso, é necessário definir os requisitos da aplicação e as funcionalidades a disponibilizar.

Para validar a plataforma a desenvolver é objectivo deste trabalho o desenvolvimento de um Portal Académico experimental para acesso via *Internet*.

1.5. Enquadramento

O presente projecto está enquadrado no âmbito do projecto *eISEL* (ISEL Electrónico), sendo este um projecto de modernização da infra-estrutura informática de suporte do ISEL. No âmbito do projecto *eISEL* foram desenvolvidos outros projectos de Mestrado, com vista a explorar soluções para a implementação de uma plataforma de integração dos vários serviços e sistemas do instituto, entre eles:

- Sistema Modular de Gestão Académica;

- Gestão Académica no Contexto do Processo de Bolonha.

O projecto "Sistema Modular de Gestão Académica" tem como protótipo uma aplicação de *front office* de Gestão Académica para o ISEL (Portal Académico) e o projecto "Gestão Académica no Contexto do Processo de Bolonha" tem como protótipo uma aplicação de *back office* de Gestão Académica. Este último protótipo nunca chegou a ser implementado devido ao facto do projecto "Gestão Académica no Contexto do Processo de Bolonha" não ter sido concluído.

A relação funcional e arquitectural entre os vários projectos é apresentada na Figura 1.1. O presente projecto tem em conta a informação dos dois projectos anteriores, disponibilizando uma plataforma capaz de desenvolver os respectivos protótipos, considerando as abordagens seguidas em cada um dos projectos. O projecto *"Sistema Modular de Gestão Académica"* utilizou uma abordagem alternativa, baseada na linguagem *Prolog*, o projecto *"Gestão Académica no Contexto do Processo de Bolonha"* não foi concluído, no entanto forneceu informação útil para este projecto, nomeadamente, o conhecimento de processos relacionados com a gestão académica.

É de realçar que o projecto ao qual o presente relatório se refere é autónomo, podendo ser utilizado no seguimento dos projectos anteriores, desde que os requisitos definidos para cada projecto sejam traduzidos no formato das mensagens reconhecidas pela plataforma desenvolvida e se realizem alguns passos de configuração.

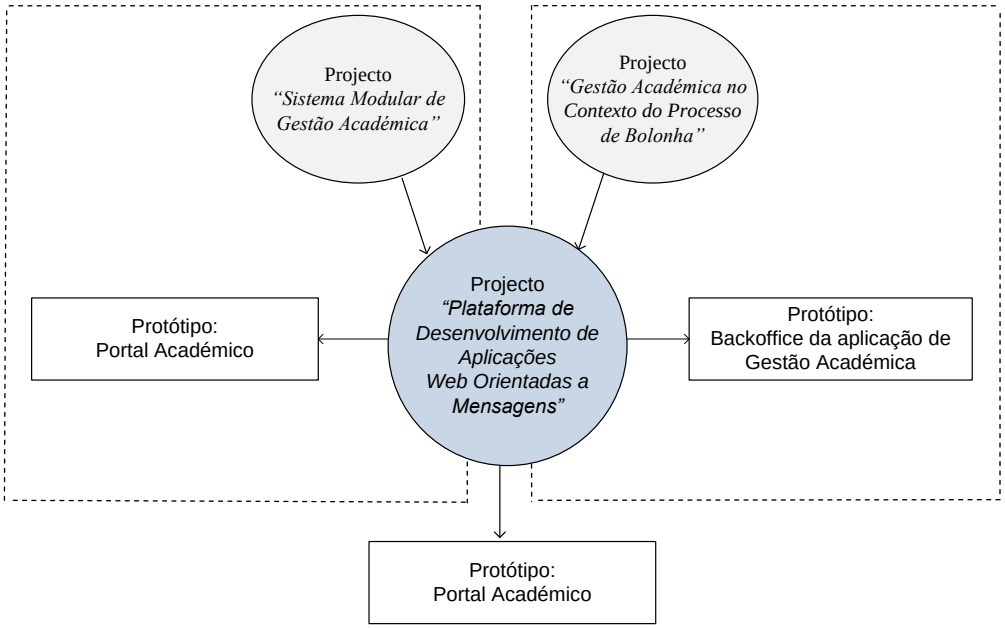


Figura 1.1 – Relacionamento entre projectos eISEL / Enquadramento

1.6. Organização do Documento

O presente documento encontra-se organizado em nove capítulos, cujos âmbitos se detalham de seguida.

O capítulo *"1. Introdução"* apresenta o projecto, as motivações do mesmo, os objectivos a alcançar e o enquadramento.

O capítulo *"2. Abordagens para o Desenvolvimento de Aplicações Web"* apresenta várias abordagens utilizadas para o desenvolvimento de aplicações *Web*.

O capítulo *"3. Enquadramento Tecnológico e Estratégia de Acção"* apresenta as opções tecnológicas que foram adoptadas no desenvolvimento da plataforma e aborda as características e os conceitos que foram adoptados no desenvolvimento do projecto. São apresentadas formas de controlar o acesso do utilizador à aplicação, as abordagens para o desenvolvimento de aplicações *Web* e é descrita a importância de separar o conteúdo da apresentação.

O capítulo *"4. Arquitectura da Plataforma"* apresenta a arquitectura da plataforma desenvolvida, como esta se encontra organizada, e alguns aspectos de segurança.

O capítulo *"5. Implementação e Configuração"* descreve como é que a plataforma na vertente *Web* e na vertente de acesso a dados foi implementada de modo a dar suporte aos objectivos propostos.

O capítulo *"6. Protótipo (Portal Académico)"* apresenta os passos necessários para o desenvolvimento de uma aplicação *Web* (Portal Académico) com a plataforma.

O capítulo *"7. Processo de Desenvolvimento"* descreve a abordagem do problema, a metodologia adoptada no processo de desenvolvimento e as actividades realizadas em cada fase. É apresentada a plataforma utilizada para a gestão do projecto e as aplicações utilizadas no desenvolvimento.

O capítulo *8* apresenta *conclusões* que se obtiveram durante a realização do projecto e descreve o trabalho futuro que pode ser realizado de forma a tornar a plataforma desenvolvida ainda mais rica.

2. Abordagens de Desenvolvimento de Aplicações *Web*

Neste capítulo serão analisadas várias abordagens para o desenvolvimento de aplicações *Web*, tendo sido seleccionadas para análise um conjunto de abordagens que reflectem as principais tendências actuais no desenvolvimento de aplicações *Web*.

Uma plataforma de aplicações *Web* oferece uma abordagem consolidada para construir aplicações *Web* dinâmicas. A plataforma deve dar aos programadores e aos *designers* de páginas uma arquitectura consistente para a construção e acesso a elementos que podem ser incorporados dentro da página. A plataforma deve incluir suporte para o estado e gestão de sessão e autenticação, bem como o acesso a dados.

As plataformas de desenvolvimento de aplicações *Web* têm como objectivo proporcionar um suporte adequado à realização de uma arquitectura adequada ao contexto da *Web*, nomeadamente no que se refere a separar o conteúdo da apresentação, fazendo os programadores responsáveis pela lógica de programação e acesso a conteúdos e dando aos *designers* das páginas o controle sobre a formatação da apresentação. Esta separação permite que os *designers* e os programadores não interfiram com o trabalho uns dos outros.

O espectro de abordagens para o desenvolvimento de aplicações *Web* pode ser dividido em quatro grandes categorias [1]:

1- abordagens baseadas em *scripting* ou programáticas;
2- padrões de páginas (*templates*);
3- abordagens híbridas;
4- plataformas.

Embora haja alguma sobreposição (bem como alguma discussão sobre se determinadas abordagens pertencem a este esquema de categorização), as abordagens mais comuns encaixam numa destas categorias. As diferenças residem nos objectos designados para conter a "fonte" para as páginas geradas, e no grau de apoio fornecido pela infra-estrutura para o desenvolvimento de aplicações avançadas e escaláveis.

De seguida, será realizada uma análise das abordagens descritas anteriormente. Não será realizada uma análise muito detalhada porque não é do âmbito deste trabalho, pelo que, para se obter uma análise mais detalhada deverá ser consultada a referência [1].

2.1. Abordagens Programáticas

Nas abordagens baseadas em *scripting* ou programáticas, a fonte associada com o objecto da página consiste predominantemente em código escrito em *Perl*, *Python*, ou *Java*. O código pode ser intercalado com formatação. Naturalmente, tais abordagens dificultam a tarefa dos programadores. A maior parte do objecto da página é composto pela lógica da aplicação, enquanto que a formatação da página (por exemplo, *HTML*) é geralmente produzido com instruções de saída associadas à linguagem de programação utilizada. Entre as abordagens que se encaixam nesta categoria estão os *scripts CGI* e *Servlets*. Para um melhor detalhe destas abordagens consultar [2] e [3], respectivamente.

O maior problema com as abordagens programáticas para desenvolvimento de aplicações *Web* é que código associado à lógica de negócio e à apresentação estão misturados. O código *HTML* (com formatação de outras construções) é embutido dentro da lógica do programa. O código *HTML* é produzido a partir de instruções de saída (por exemplo, '*print*') associado com a linguagem de programação de origem. Isso limita a contribuição criativa que os *Web designers* podem ter no *layout* final da página. A intervenção de programadores é necessária quando se pretende modificar qualquer aspecto da página, esteja essa modificação relacionada com a lógica de programação ou com o *layout* de apresentação.

2.2. Padrões de Páginas (*Templates*)

Um *template* utiliza um objecto de origem (o modelo), que consiste predominantemente em estruturas formatadas, com uma zona limitada que permite adicionar programação. O foco do objecto de origem é a formatação, e não a lógica de programação. Naturalmente, esta abordagem apela aos autores de páginas *Web* e *designers* gráficos muito mais do que a abordagem de *scripting*, ou programática.

Conforme mencionado na secção anterior, as abordagens de *scripting* / programáticas são centradas no código e os objectos de origem associados com a geração

da página são *scripts* ou programas. O *template*, por outro lado, gira em torno da estrutura da página e da formatação das *tags*, não em torno do código. O mecanismo *Server-Side Includes* (*SSI*) foi uma das formas utilizadas para adicionar a funcionalidade de um *template* simples às páginas *Web*. Outros *templates* bem conhecidos são *Allaire's Cold Fusion* (explicado a seguir) [4] e *WebMacro / Velocity* (consultar [3] para melhor detalhe).

2.2.1. *Cold Fusion*

A abordagem *Cold Fusion* representa uma das primeiras abordagens de *templates* comerciais para a geração dinâmica de páginas *server-side*, fornecendo um conjunto de marcas que suportam a inclusão de recursos externos, apresentação de resultados interactiva e acesso a dados. A abordagem *Cold Fusion* deve muito do seu sucesso a dois aspectos:

1. *Querys* de acesso a dados fáceis de criar e usar;
2. Cada forma de acesso a dados funciona como uma consulta.

As *querys* sobre a base de dados são construídas usando o elemento *<CFQUERY>*, referenciando uma fonte de dados *ODBC* com o código *SQL* embutido entre as *tags* *<CFQUERY>* e *</CFQUERY>*. Os resultados podem ser percorridos iterativamente usando o elemento *<cfoutput>*, com cada coluna disponível para a variável de substituição (Figura 2.1).

```
<CFQUERY DATASOURCE="oracle-prod" NAME="dbquery1">
   SELECT  NAME,  ADDRESS,  PHONE
     FROM CUSTOMERS
    WHERE LAST_PURCHASE_DATE < '2001-01-01';
</CFQUERY>
   ...
<TABLE>
<TR>
   <TD ALIGN="center"><B>Name</B></TD>
   <TD ALIGN="center"><B>Address</B></TD>
   <TD ALIGN="center"><B>Phone</B></TD>
<TR>
<CFOUTPUT QUERY="dbquery1">
   <TD>#NAME#</TD>
   <TD>#ADDRESS#</TD>
   <TD>#PHONE#</TD>
</CFOUTPUT>
</TR>
</TABLE>
```

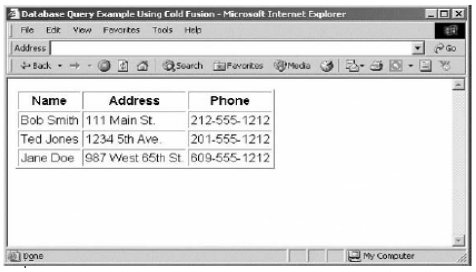


Figura 2.1 – Exemplo de uma aplicação *Cold Fusion* com uma *query* à base de dados

A abordagem *Cold Fusion* fornece acesso a variáveis de ambiente (por exemplo, parâmetros da *query*, componentes do *URL* e os dados da sessão). Ao evoluir, a plataforma *Cold Fusion* acabou por sucumbir à pressão para fornecer capacidades de *scripting* em *templates*. Esta capacidade não é usada pela maioria das aplicações *Cold Fusion* desenvolvidas porque leva facilmente à criação de uma mistura não estruturada de código e de formatação.

Apesar do *Cold Fusion* oferecer muitas das características associadas com um *template* sólido para o desenvolvimento de aplicações *Web*, tem alguns pontos negativos, como é o caso de ser um produto de *software* proprietário, bem como o *Cold Fusion Markup Language (CFML)* [5], que é propriedade intelectual da *Macromedia*.

A abordagem *Cold Fusion* pode ser aceitável se o objectivo for desenvolver uma aplicação simples com um pequeno número de utilizadores e que não exija o desempenho de uma plataforma robusta.

2.3. Abordagens Híbridas

As abordagens híbridas combinam elementos de *script* com estruturas *template*. Este tipo de abordagens têm mais poder programático do que os *templates* puros porque permitem embutir blocos que contêm *scripts*. O resultado é uma página orientada à estrutura combinada com o poder adicional de programação. Exemplos desta abordagem são, principalmente, *PHP*, *Sun Java Server Pages (JSP)* e *Microsoft's Active Server Pages (ASP)*.

A mistura de blocos de *script* com formatação de apresentação constitui uma violação grave do princípio da separação entre o conteúdo e a apresentação. A questão de quem "possui" o objecto de origem torna-se muito confusa. Se os *designers* e os programadores trabalharem com a mesma fonte de objectos pode dar origem a conflitos e colisões quando as mudanças de código quebrarem a formatação *HTML*, ou quando as alterações feitas por *designers* inadvertidamente introduzem erros no código.

A maioria destes sistemas foram concebidos para traduzir objectos híbridos em código.

2.3.1. *PHP: Hypertext Preprocessor (PHP)*

O *PHP* permite que os programadores introduzam código em *templates HTML*. O objecto fonte é estruturado como uma página *HTML*, mas a geração de conteúdo dinâmico é programática.

Por exemplo, o fragmento de código *PHP*:

```
<B>
<?php
    if ($xyz >= 3) { print $myHeading; }
    else {
?>
DEFAULT HEADING
<?php
    }
?>
</B>
```

Exemplo 2.1 – Fragmento de código *PHP* (extraído de [1])

Pode ser traduzido para:

```
<?php
    print "<B>";
    if ($xyz >= 3) { print $myHeading; }
    else { print "DEFAULT HEADING"; }
    print "</B>"
?>
```

Exemplo 2.2 – Código *PHP* com instruções *print* (extraído de [1])

Em outras palavras, texto inserido dentro de blocos *<?php. . . ?>* são processados utilizando a linguagem *PHP* nativa, enquanto o texto fora destes blocos é tratado como argumentos passados para instruções *'print'*.

Enquanto outras abordagens baseadas em *templates* fornecem diversos elementos distintos concebidos para executar tarefas específicas, no *PHP* há um bloco, *<?php...?>*, que serve como um *container* para o código *PHP*. Apesar dos *scripts PHP* serem muitas vezes referidos como *templates*, que dependem do código para executar a maioria dos trabalhos relacionados com a geração de páginas dinâmicas, tornam o *PHP* mais próximo de uma abordagem de *scripting* do que uma abordagem *template*.

Em termos da plataforma desenvolvida, que será apresentada mais à frente, foi escolhido o *PHP* pelo facto de ser uma linguagem orientada a objectos e de ser fácil de lidar com servidores de base de dados, como *MySQL*, *PostgreSQL*, *Microsoft SQL Server e Oracle*.

Como se pretende que a plataforma possa suportar vários sistemas de bases de dados o *PHP* revela-se uma escolha adequada.

2.3.2. *Java Server Pages (JSP)*

O *Java Server Pages* (*JSP*) é uma tecnologia utilizada no desenvolvimento de aplicações *Web*, similar às tecnologias *PHP* ou *Active Server Pages* (*ASP*) da *Microsoft* [1]. Como acontece com *PHP*, o suporte *JSP* foi implementado através de um pré-processador que transformou objectos da página com blocos de código embutidos em *servlet source code* [3]. A página em *JSP* do Exemplo 2.3, será traduzida para código *servlet* semelhante ao mostrado no Exemplo 2.4. A primeira linha do fragmento *JSP* do Exemplo 2.3 é a directiva da página para importar classes do pacote *java.io*. As próximas três linhas representam uma declaração de variável. Os blocos de código *Java* são delimitados por '<%' e '%>'. O código *HTML* fora destes delimitadores é traduzido em instruções '*print*'. A página inteira é convertida numa classe *Java* que é compilada pelo servidor.

```
<%@ page import="java.io.*" %>
<%!
  private CustomObject myObject ;
%>
  <h1>My Heading</h1>
<%
  for(int i = 0; i < myObject.getCount(); i++) {
%>
  <P>Item #<%= i %> is '<%= myObject.getItem(i) %>'.</P>
<%
  }
%>
```

Exemplo 2.3 – Página em *JSP* (extraído de [1])

O *JSP* representa ainda outra abordagem para converter estruturas híbridas de páginas em código que é então compilado e executado. No caso do *JSP*, o código é traduzido num *servlet Java* que é compilado e executado pelo motor *servlet* do servidor *Web* [3].

```
package jsp. myapp ;
import java.io.* ;
import java.util.* ; import javax.servlet.* ;
import javax.servlet.http.*; import javax.servlet.jsp.* ;
public class mypage extends HttpJspBase {
   private CustomObject myObject;
   public void jspService(HttpServletRequest req, HttpServletResponse
resp)
   {
      ServletConfig config = getServletConfig() ;
      ServletContext application = config.getServletContext() ;
      Object page = this ;
      PageContext pageContext =
      JspFactory.getDefaultFactory().getPageContext(this, req, resp,
      null, true, 8192, true) ;
      JspWriter out = pageContext.getOut() ;
      HttpSession session = request.getSession(true) ;
      out.print("<h1>My Heading</h1>") ;
      for(int i = 0; i < myObject.getCount(); i++) {
          out.print("<P>Item #" + i + " is '" +
          myObject.getItem(i) + "'.</P>") ;
      }
    }
}
```

Exemplo 2.4 – Tradução para a página em *JSP* do exemplo 2.3 (extraído de [1])

2.4. Separação do Conteúdo da Apresentação

Do que se observa da análise anterior nenhuma das abordagens cumpre um dos principais requisitos de uma boa plataforma de aplicações *Web*: a verdadeira separação do conteúdo da apresentação.

A separação do conteúdo da apresentação é importante para dar flexibilidade à aplicação e para dividir as responsabilidades entre quem está responsável pelos conteúdos e quem está responsável pela apresentação, aspectos de seguida abordados.

2.4.1. *Flexibilidade da aplicação*

Numa aplicação *Web*, a componente de apresentação consiste na organização e na definição da estrutura dos conteúdos no formato desejado, ou seja, é uma vista sobre os dados ou conteúdos. O conteúdo pode ser apresentado de muitas maneiras diferentes. A escolha do modo de apresentação deve ser separada das opções feitas para aceder aos

dados, de modo a que qualquer conteúdo possa ser apresentado por uma linguagem capaz de apresentar conteúdos (por exemplo, *HTML*, *WML*[2], etc.).

Não importa se o seu conteúdo foi lido a partir de um ficheiro ou se foi extraído de uma base de dados através de uma consulta. O que importa é que o modelo de dados deve ser aberto para que ele seja usado por uma variedade de vistas e que deve existir um mecanismo de controlo que será a cola que recupera os conteúdos com o formato adequado para a apresentação, ou seja, respeitando o padrão de desenho Modelo-Vista-Controlador, no original *Model-View-Controller* (*MVC*), ver Figura 2.2.

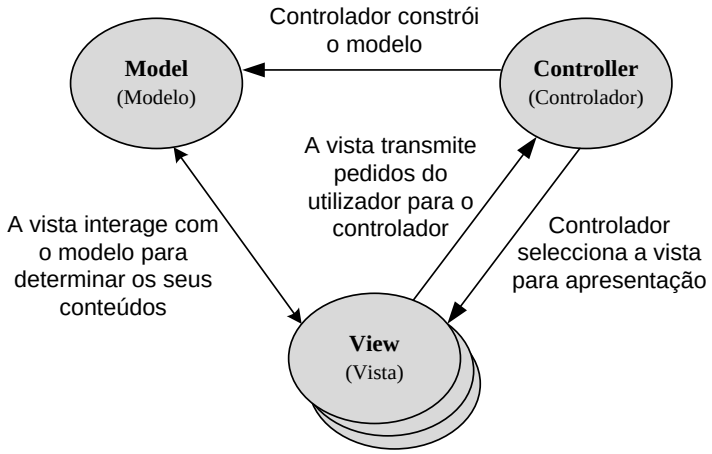


Figura 2.2 – Padrão de desenho *Model-View-Controller*

O controlador recebe o pedido do utilizador, constrói o modelo associado ao pedido e selecciona a vista (estrutura) para apresentar os conteúdos. A vista comunica com o modelo para determinar o seu conteúdo, e apresenta esse conteúdo para o utilizador no formato desejado. A vista também funciona como uma *interface* para transmitir novos pedidos do utilizador para o controlador. Este padrão é concebido para facilitar a verdadeira separação do conteúdo da apresentação, permitindo o desenvolvimento de aplicações que possam dinamicamente adaptar-se e personalizar a sua apresentação baseada nas preferências do utilizador. O modelo de dados não está vinculado a um único formato de apresentação, o que permite uma maior flexibilidade da aplicação.

[2] O ***Wireless Markup Language*** (***WML***) é um formato de conteúdo, baseado em *XML*, para dispositivos que utilizam *WAP*, como telefones móveis. O *WML* é muito semelhante ao *HTML* fornecendo suporte de navegação, entrada de dados, *hyperlinks*, apresentação de imagem, texto e formulários.

2.4.2. *Divisão de responsabilidade por módulos*

Outra razão para a separação do conteúdo da apresentação é o facto das pessoas responsáveis por esses dois aspectos de uma aplicação terem capacidades muito diferentes. Os especialistas da apresentação são os *designers* com capacidades que se baseiam em linguagens de formatação, como o *HTML*, ferramentas de *design* de páginas, como *Macromedia Dreamweaver* ou o *Microsoft Frontpage*, entre outras. Eles não são programadores, portanto, a sua especialidade não está na área da programação e na lógica da aplicação, mas sim na capacidade para projectar e implementar o *layout* das páginas. O acesso ao conteúdo é da responsabilidade dos programadores de aplicações.

Uma das razões pelas quais a separação entre o conteúdo e a apresentação é tão importante, é que desta forma é possível garantir a divisão de responsabilidades entre aqueles que têm acesso aos conteúdos (e aos processos) e aqueles que apresentam os conteúdos. São poucas as pessoas que têm todas as capacidades necessárias para realizar todas as tarefas associadas à geração dinâmica de páginas. *Designers* e programadores têm orientações diferentes, capacidades diferentes e requisitos diferentes. As colisões entre os esforços dos *designers* e dos programadores para modificar a mesma página ocorrem com muita frequência.

Quando se permite a mistura de blocos de código (acesso e dados e lógica de negócio) com a estrutura da página (forma de apresentação) existe a tentação de tornar toda a página num bloco contíguo de código. Além do mais, essa mistura torna o código de difícil leitura, quer para o *designer*, quer para o programador.

Uma abordagem baseada em *MVC* torna possível combinar a flexibilidade da aplicação com a divisão de responsabilidade adequada. Os programadores são responsáveis pelo componente controlador. Este é um módulo leve que delega o processamento para as tarefas adequadas, também criadas por programadores. Estas tarefas são responsáveis por aceder aos dados e por construir o modelo. Os especialistas de apresentação são responsáveis pela construção das vistas (*interfaces*). O controlador pode determinar dinamicamente que vista irá apresentar os dados associados com o modelo.

A ideia não é encapsular uma página num único módulo, mas sim relacionar os componentes responsáveis por cada tarefa. A utilização de normas bem definidas para relacionar os componentes é necessária para facilitar a compreensão. A secção seguinte

apresenta algumas plataformas utilizadas para o desenvolvimento de aplicações *Web* que incorporam os aspectos atrás referidos.

2.5. Plataformas

Nesta secção são apresentadas algumas plataformas de desenvolvimento de aplicações *Web*, nomeadamente, a plataforma *DIF* (*Digitalis Internal Framwork*), a plataforma *OutSystems* e a plataforma *Echo*.

O interesse pela plataforma *DIF* foi devido ao facto de esta ser a plataforma utilizada para o desenvolvimento de aplicações *Web* pela empresa que fornece as aplicações de gestão académica do ISEL. Sendo o actual Portal Académico do ISEL desenvolvido através desta plataforma, considerou-se interessante analisar a forma como isso é conseguido.

A plataforma *OutSystems* permite criar e modificar aplicações num tempo muito reduzido, adoptando metodologias ágeis de desenvolvimento. Estas aplicações podem ser facilmente alteradas durante todo o seu ciclo de vida, com um risco de projecto muito reduzido. Esta flexibilidade garante o alinhamento dos sistemas com as necessidades reais de negócio. Estas características encaixam no que se pretende para a plataforma a desenvolver, sendo portanto, uma plataforma interessante a analisar.

O interesse pela plataforma *Echo* foi devido ao facto de esta ser uma plataforma *open source* e por estar associada ao desenvolvimento de aplicações *Web* com características de clientes ricos.

2.5.1. *Plataforma DIF (Digitalis Internal Framework)*

A plataforma *DIF* é uma plataforma criada pela empresa *Digitalis*, que é uma empresa de *software* e serviços com o objectivo de auxiliar as Instituições de Ensino Superior a tirarem o maior partido e a extraírem o maior valor da tecnologia aplicada à gestão e distribuição do ensino. Esta empresa é a fornecedora das aplicações de gestão académica do ISEL.

Foi observado que a maioria das aplicações não corresponde ao que é pretendido pelo ISEL em termos de funcionalidade e apresentam erros de utilização que dificultam

a utilização das aplicações. Face a isto, foi efectuada uma análise da plataforma de desenvolvimento *DIF* de modo a perceber a sua arquitectura e o seu funcionamento.

A *DIF* é uma plataforma de desenvolvimento integrado para aplicações *Web* que pretende responder aos desafios tecnológicos actuais na *Web*, bem como aos desafios futuros com as novas gerações de dispositivos que abrirão novos caminhos no campo do ensino, nomeadamente no ensino à distância e na disponibilização de consultas e serviços *online* de conteúdos e alertas para toda uma nova geração de utilizadores globais.

Trata-se de uma plataforma integrada de desenvolvimento, uma ferramenta que permite a construção de aplicações e o desenvolvimento de serviços de qualquer tipo, nomeadamente na área do ensino, e da publicação dos seus conteúdos para diversos dispositivos e *interfaces*. A figura seguinte apresenta o diagrama interno da plataforma *DIF*.

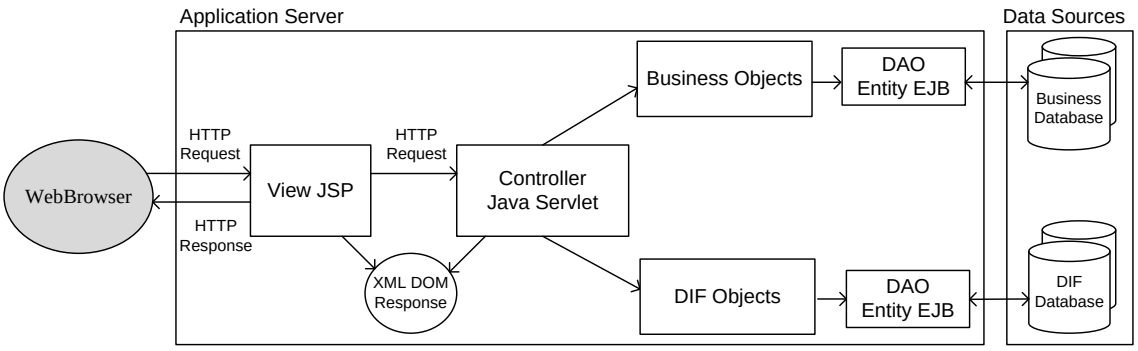


Figura 2.3 – Diagrama interno da plataforma *DIF*

Apesar da empresa *Digitalis* não fornecer documentação sobre a sua plataforma foi realizada uma análise e conclui-se que a plataforma respeita o padrão de desenho *Model-View-Controller* (*MVC*), explicado anteriormente.

O componente *View JSP* funciona como uma *interface* para transmitir os pedidos do utilizador para o componente *Controller Java Servlet*. Este último componente recebe o pedido do utilizador e constrói o modelo associado ao pedido. O componente *View JSP* consome a resposta construída pelo componente *Controller Java Servlet* e apresenta esse conteúdo para o utilizador no formato desejado.

O componente *Controller Java Servlet* pode instanciar os serviços associados à lógica de negócio que se encontram no componente *"Business Objects"* ou objectos da própria plataforma *DIF* que se encontram no componente *"DIF Objects"*. Ambos os

componentes, *"Business Objects"* e *"DIF Objects"*, possuem objectos específicos de acesso a dados, representados na Figura 2.3 pelo componente *"DAO Entity EJB"*.

Da análise efectuada, concluiu-se que a plataforma é complexa tecnologicamente. Esta exige um esforço de aprendizagem para que possa ser utilizada correctamente e bons conhecimentos em *Java*.

A própria *Digitalis* exige formação específica para se iniciar o desenvolvimento de aplicações suportadas pela *DIF*, disponibilizando para o efeito muita documentação o que demonstra bem da complexidade da plataforma [6].

Verifica-se que esta plataforma de desenvolvimento não está orientada aos requisitos, ou seja, com base nestes não é possível automatizar a construção de aplicações *Web*, obrigando a que novas funcionalidade tenham que ser implementadas por um programador com um relativo esforço de desenvolvimento e com um tempo de desenvolvimento significativo. No contexto da plataforma desenvolvida, o tempo de desenvolvimento é um ponto que se pretende minimizar.

2.5.2. *Plataforma OutSystems*

A tecnologia *OutSystems* combina a produtividade oferecida por ferramentas de modelação visual com o uso de plataformas de desenvolvimento. A plataforma *OutSystems* fornece o que se necessita para criar e disponibilizar rapidamente as aplicações e mudar essas aplicações em qualquer altura do seu ciclo de vida. A base assenta num ambiente visual altamente intuitivo e numa tecnologia de publicação (*deployment*[3]) que ajuda na entrega das aplicações e serviços em semanas, em vez de meses ou anos [7].

A plataforma *OutSystems* inclui os seguintes componentes [7] e [8]:

- ***OutSystems Service Studio*** **-** é uma ferramenta de desenvolvimento visual que permite aos programadores criarem, alterarem e publicarem aplicações. A implementação da *interface* gráfica, da lógica da aplicação e dos fluxos de interacção com o utilizador é realizada recorrendo a um processo simples e intuitivo de "arrastar e largar" (*drag-and-drop*).

[3] Todas as actividades que tornam as aplicações disponíveis para utilização.

- ***OutSystems Hub Server*** **-** é uma plataforma de execução que orquestra todas as actividades de execução, de publicação e de gestão de cada aplicação criada no *Service Studio*.
- ***OutSystems Service Center*** **-** é uma aplicação que coordena todas as operações de gestão, monitorização e publicação das aplicações *OutSystems* e componentes integrados de uma maneira centralizada e consistente.
- ***OutSystems Integration Studio*** **-** permite incorporar na plataforma *OutSystems* elementos externos, como por exemplo, bases de dados. Os componentes integrados ficam num catálogo de componentes que podem ser reutilizados pelas diversas aplicações.

A Figura 2.4 ilustra a arquitectura de componentes da plataforma *OutSystems*.

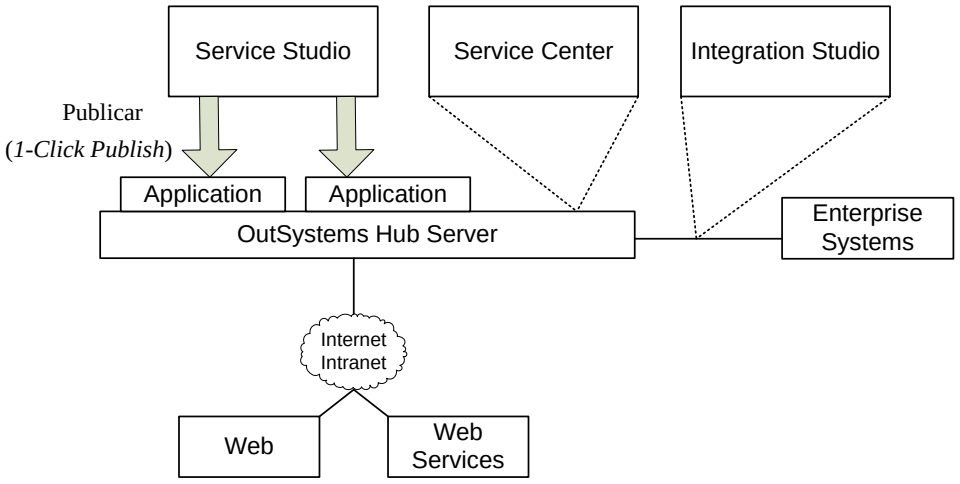


Figura 2.4 – Arquitectura de componentes da plataforma *OutSystems*

O componente *Web* representa uma página *Web* e o componente *Web Services* permite disponibilizar funcionalidades através de serviços *Web*.

Para executar uma aplicação criada no *Service Studio* é necessário estar ligado ao *OutSystems Hub Server*, onde a aplicação será publicada e alojada. A operação "*1-Click Publish*" fornece o processo completo para publicar automaticamente as aplicações. A Figura 2.5 ilustra os passos associados à operação "*1-Click Publish"* e a arquitectura do *Hub Server*.

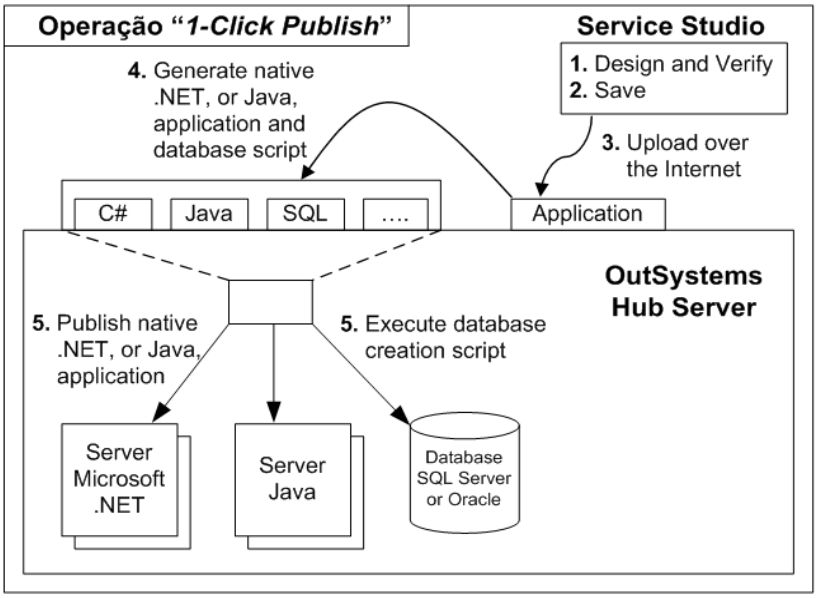


Figura 2.5 – Passos associados à operação "*1-Click Publish*" e arquitectura do *Hub Server*

A arquitectura típica da infra-estrutura publicada inclui [9]:

- Servidores *standard .NET* ou *Java* onde as aplicações são publicadas e executadas;
- Uma base de dados (baseada em *SQL Server* ou *Oracle*) que mantém os dados usados por todas as aplicações;
- e o *OutSystems Service Center* que é responsável por automatizar a publicação da aplicação, por fornecer todas as funcionalidades de monitorização, e fornecer uma consola de gestão para gerir toda a infra-estrutura.

A plataforma *Outsystems* apesar de bastante completa exige o conhecimento da própria tecnologia, o que significa que para a sua utilização é necessário um período de aprendizagem e de adaptação às características tecnológicas desta plataforma. Face a isto, pretende-se que a plataforma desenvolvida minimize o conhecimento tecnológico, abstraindo o utilizador desses aspectos.

Para além disso, para uma utilização mais completa da plataforma *Outsystems* obriga à necessidade de licenciamento, visto a tecnologia ser proprietária, o que origina algumas limitações.

2.5.3. *Plataforma Echo*

A *Echo* é uma plataforma *open source* para desenvolvimento de aplicações *Web* com características de clientes ricos [10]. Na perspectiva do programador, a plataforma *Echo* comporta-se como uma ferramenta de desenvolvimento visual, como o *Swing* [11], o *Eclipse Standard Widget Toolkit (SWT)* [12], ou como o *Service Studio* da *OutSystems* explicado anteriormente. A tecnologia *AJAX* é aplicada para permitir aos clientes *Web* uma experiência aproximada à utilização de aplicações *desktop*. O *AJAX* permite ao utilizador não esperar que uma página *Web* se recarregue ou que o processo seja terminado para continuar usando a aplicação. Cada informação é processada separadamente, de forma assíncrona, evitando assim recarregar a página a cada clique.

2.5.3.1. Visão geral sobre a arquitectura

A plataforma *Echo* pode ser pensada como sendo duas plataformas numa: uma plataforma cliente *Javascript* e uma plataforma servidora *Java*. Se desejado os programadores podem escolher uma das duas plataformas e criar aplicações sem nunca ter aprendido a outra [13].

Ambas as plataformas têm aproximadamente a mesma *API*. Ou seja, cada uma tem os objectos que representam o mesmo conjunto de componentes, eventos e propriedades com as quais as aplicações são construídas. As *APIs* diferem para atender às necessidades de cada linguagem (*Javascript* e *Java*), mas conceptualmente são quase idênticas.

Plataforma Cliente *Javascript*

A versão *Javascript* do lado do cliente do *Echo3* pode ser executado por si só, se assim for desejado, uma vez que não depende de um servidor.

Todo o código escrito na plataforma cliente *Javascript*, *Echo3*, é executado no navegador do cliente remoto. Se o código necessitar de interagir com um servidor, ele pode fazer isso usando *HTTP*. Como acontece com qualquer código que está sendo executado num cliente remoto, a segurança precisa ser tratada exclusivamente no servidor, devido ao facto do cliente remoto não ser confiável.

Plataforma Servidora *Java*

A versão *Java* do lado do servidor do *Echo3* corre em *Java Servlet Container* (como este assunto não é do âmbito deste trabalho para mais detalhes consultar [14] e [15]). O primeiro acto do servidor é, ao receber um visitante enviar uma aplicação construída na versão cliente do *Echo*. Esta aplicação cliente permitirá activar o servidor para remotamente exibir o estado da aplicação que está actualmente a ser executada no servidor a partir do navegador *Web* do utilizador. O servidor irá comunicar com a aplicação cliente usando *XML* sobre *HTTP*, serializando as mudanças de estado, processando o input do utilizador e apresentando as actualizações.

O código da plataforma servidora *Echo3* é escrito em *Java* e é executado no servidor. Nunca é transmitido para o cliente, apenas o estado da *interface* é apresentado ao utilizador. As aplicações *Echo* do lado do servidor podem usar todos os meios de acesso aos dados disponíveis para uma aplicação *Java Servlet* [15], por exemplo, *SQL*, ou *XML / HTTP*.

2.5.3.2. Aplicações

Os programadores que escrevem componentes personalizados terão que interagir o seu código *HTML* / *Javascript* com a *API* da plataforma *Echo* [15].

As aplicações *Echo* podem ser criadas totalmente em código *Java* do lado do servidor ou como aplicações cliente escritas em *Javascript* (*Echo3*) [10]. Para isso, é necessário criar classes específicas que se encontram definidas quer para a plataforma cliente *Javascript* [17], quer para a plataforma servidor *Java* [18].

O desenvolvimento de aplicações no lado do servidor exige o conhecimento da linguagem *Java*, enquanto o desenvolvimento de aplicações no lado do cliente existe conhecimento de *Javascript*. Conhecimentos básicos de *AJAX* e de *XML*, ou *JSON*, serão benéficos se for necessário comunicar com serviços do lado do servidor.

A utilização da plataforma *Echo* para além de exigir conhecimentos nas linguagens referidas anteriormente também exige o conhecimento dos módulos e das especificidades que fazem parte da plataforma, o que origina um esforço significativo na utilização da plataforma.

Verifica-se que esta plataforma de desenvolvimento não está orientada aos requisitos, ou seja, com base nestes não é possível automatizar a construção de aplicações *Web*, obrigando a que novas funcionalidades tenham que ser implementadas por um

programador com conhecimentos tecnológicos em *Javascript* ou *Java*, ou ambos. No contexto da plataforma desenvolvida, o nível de conhecimento tecnológico para a utilização da plataforma é um ponto que se pretende minimizar.

3. Enquadramento Tecnológico e Estratégia de Acção

Neste capítulo é apresentado o enquadramento tecnológico e a estratégia de acção que foi tida em consideração no desenvolvimento deste projecto.

3.1. Enquadramento Tecnológico

As tecnologias de suporte à aplicação são um ponto importante a definir. É a escolha destas tecnologias e as opções tomadas neste âmbito que vão delimitar as capacidades futuras da solução alcançada e a sua usabilidade e interoperabilidade.

Um ponto importante na construção de aplicações *Web* é a adopção de normas de desenvolvimento (*standards*). Este ponto é importante devido à disseminação existente de vários navegadores *Web*[4], levando a que a utilização da aplicação seja feita através de várias plataformas, com comportamentos e potencialidades distintas. No entanto, algo que une todas estas ferramentas são os *standards*, levando a que uma aplicação baseada nestes seja correctamente apresentada e funcione de acordo com o esperado em qualquer navegador *Web* que lhes dê suporte.

As tecnologias a utilizar como base para a programação de aplicações *Web* no lado cliente são o *Javascript*, *DOM* [5] e *CSS*. Com estas é possível controlar o comportamento e visualização dos componentes e páginas.

[4] **Navegador *Web***, também conhecido pelos termos ingleses *web browser* ou simplesmente *browser*, é um programa que habilita os seus utilizadores a interagirem com documentos virtuais da *Internet*, também conhecidos como páginas *Web*.

[5] ***Document Object Model (DOM)*** é uma norma da *W3C*, independente da plataforma e da linguagem, que especifica o modo de dinamicamente alterar e editar a estrutura, conteúdo e estilo de um documento.

3.1.1. *Utilizadores Alvo*

Por observação do *website* do *W3schools* [1] é possível consultar as estatísticas de utilização dos vários navegadores *Web* ao longo dos anos. As estatísticas do último ano são apresentadas na Tabela 3.1. Através da tabela verifica-se que o navegador *Web* com mais impacto no mercado é o *Firefox*, tornando a compatibilidade com este navegador *Web* imperativa, seguida das duas versões do *Internet Explorer* utilizadas pelos utilizadores do site actual.

Perante estes dados verifica-se que deverá ser mantida compatibilidade, em versões actuais do site, com a seguinte lista de navegadores *Web*:

- *Internet Explorer 7*
- *Mozilla Firefox 3*

Desta forma garante-se que as alterações não vão ter um impacto negativo nos utilizadores, sendo garantida a sua aceitação, pelo menos no que se refere ao navegador *Web*.

2010	IE8	IE7	IE6	Firefox	Chrome	Safari	Opera
July	15.6%	7.6%	7.2%	46.4%	16.7%	3.4%	2.3%
June	15.7%	8.1%	7.2%	46.6%	15.9%	3.6%	2.1%
May	16.0%	9.1%	7.1%	46.9%	14.5%	3.5%	2.2%
April	16.2%	9.3%	7.9%	46.4%	13.6%	3.7%	2.2%
March	15.3%	10.7%	8.9%	46.2%	12.3%	3.7%	2.2%
February	14.7%	11.0%	9.6%	46.5%	11.6%	3.8%	2.1%
January	14.3%	11.7%	10.2%	46.3%	10.8%	3.7%	2.2%
2009	**IE8**	**IE7**	**IE6**	**Firefox**	**Chrome**	**Safari**	**Opera**
December	13.5%	12.8%	10.9%	46.4%	9.8%	3.6%	2.3%
November	13.3%	13.3%	11.1%	47.0%	8.5%	3.8%	2.3%
October	12.8%	14.1%	10.6%	47.5%	8.0%	3.8%	2.3%
September	12.2%	15.3%	12.1%	46.6%	7.1%	3.6%	2.2%
August	10.6%	15.1%	13.6%	47.4%	7.0%	3.3%	2.1%
July	9.1%	15.9%	14.4%	47.9%	6.5%	3.3%	2.1%
June	7.1%	18.7%	14.9%	47.3%	6.0%	3.1%	2.1%
May	5.2%	21.3%	14.5%	47.7%	5.5%	3.0%	2.2%
April	3.5%	23.2%	15.4%	47.1%	4.9%	3.0%	2.2%
March	1.4%	24.9%	17.0%	46.5%	4.2%	3.1%	2.3%
February	0.8%	25.4%	17.4%	46.4%	4.0%	3.0%	2.2%
January	0.6%	25.7%	18.5%	45.5%	3.9%	3.0%	2.3%

IE	Internet Explorer
Firefox	Firefox (identified as Mozilla before 2005)
Chrome	Google Chrome
Mozilla	The Mozilla Suite (Gecko, Netscape)
Safari	Safari (and Konqueror. Both identified as Mozilla before 2007)
Opera / O	Opera
N	Netscape (identified as Mozilla after 2006)
AOL	America Online (based on both Internet Explorer and Mozilla)

Tabela 3.1 – Percentagens de utilização dos vários navegadores *Web* (extraído de [1])

3.1.2. *Web 2.0*

O conceito de *Web* 2.0 foi introduzido numa conferência organizada pela *O'Reilly* e a *MediaLive International* [20]. A *Web* 2.0 é um conjunto de abordagens a problemas e modelos já existentes, um conjunto de princípios e práticas que contribuem em conjunto para levar a informação, aplicações e funcionalidades à maioria das pessoas.

O ponto mais importante na definição da *Web* 2.0 é a utilização da *Web* como plataforma e não como tinha sido feito até então, em que esta era um mecanismo ou uma funcionalidade utilizada: passar do navegador *Web* como ferramenta executora para ferramenta de suporte para acesso a aplicações na *Web*.

A *O'Reilly* identificou conceitos base que identificam uma aplicação *Web* 2.0, que são apresentados em seguida.

3.1.2.1. Conceitos base

Segundo os documentos [20] e [21] a *Web* 2.0 pode ser classificada com base em vários conceitos. Também é de notar que para uma aplicação ou plataforma ser classificada como *Web* 2.0 não terá que explorar todos os conceitos apresentados, podendo mesmo explorar apenas um deles de forma mais abrangente.

No âmbito deste trabalho vão ser abordados apenas quatro conceitos que são os que foram tidos em conta no desenvolvimento da plataforma.

3.1.2.2. A *Web* como Plataforma

Este será o conceito base que terá que ser correspondido. Actualmente, a filosofia predominante será a de utilizar a *Web* como plataforma das aplicações.

A disponibilização de serviços, tal como acontece com o *Google*, que já é abrangente o suficiente para substituir com os serviços disponibilizados muitas aplicações para utilização local (pacote de *Office*, visualização de *PDFs*, lembretes, dicionários, etc.), tudo acessível graças à plataforma utilizada (a *Internet*) de uma forma simples, em qualquer PC e utilizado por qualquer pessoa [22].

Este é o conceito fundamental visto ser o grande motivador da mudança e no qual se baseiam os restantes. Se não pensarmos na *Web* como a plataforma das aplicações não vamos fazer aplicações para todos, vamos levar a aplicação a muito menos utilizadores e estará sempre dependente das especificidades dos mesmos.

Segundo esses princípios, os *softwares* são desenvolvidos de modo que fiquem melhores à medida que são usados mais vezes, pois os utilizadores podem ajudar a torná-lo melhor.

3.1.2.3. Alteração do Ciclo de Produção de *Software*

Na *Web* 2.0 não devem ser disponibilizadas aplicações fechadas, mas sim prestados serviços, serviços estes que deverão manter-se em constante actualização, adaptando-se às necessidades e inovando através da disponibilização de novas funcionalidades. Assim sendo, as aplicações deverão continuar o seu ciclo de desenvolvimento, não respeitando o ciclo de desenvolvimento clássico.

3.1.2.4. Modelos de Programação Leves

Visto ser algo a disponibilizar a um público mais largo e tentar fornecer serviços que possam ser utilizados por outros para agregarem nas suas aplicações, aumentando o seu valor e contribuindo para a grande comunidade da *Web* 2.0, não se pode basear a construção das aplicações em protocolos complexos.

Começaram-se a desenvolver *softwares* que são usados pela *Internet* e vendidos não em pacotes mas como serviços, pagos mensalmente. Além disso, mudou-se a forma de fazer *software*. Definiu-se então que quanto mais simples e modular for a programação, melhor. Assim, é fácil tirar ou acrescentar uma funcionalidade ou partilhar uma parte do *software* com outro *software*.

Os módulos podem ser reutilizados em diversos *softwares* ou partilhados para serem usados por programas de terceiros.

Desta forma, a modularidade foi um conceito que foi considerado no desenvolvimento da plataforma e que será explicado no ponto *3.2.1 Conceitos base*.

3.1.2.5. Experiências de Utilização Ricas

As *interfaces* gráficas para as páginas *Web* têm vindo a evoluir cada vez mais, existindo um crescente número de formas de disponibilizar ao utilizador *interfaces* dinâmicas, inovadoras e que conduzam a um maior entrosamento do utilizador com a aplicação.

As *interfaces* ricas, que se assemelham um pouco a aplicações *stand-alone* para o *PC*, mas misturando os conceitos de navegabilidade da *Internet*, tornam a experiência do utilizador mais rica, com *interfaces* rápidas e muito fáceis de usar.

É através das *interfaces* que se faz a agregação dos serviços e que se disponibilizam as funcionalidades. A capacidade de levar *interfaces* cada vez mais completas ao utilizador para a utilização de serviços é que permite a construção e disponibilização de serviços que substituam as aplicações que correm nos nossos *PCs* e mesmo os sistemas operativos.

Para desenvolver *interfaces* ricas recorre-se a uma combinação de tecnologias, que incluem, por exemplo, *Web Services* e *AJAX*. Estas tecnologias aumentaram a velocidade e a facilidade de uso das aplicações *Web*.

O *AJAX* é um componente chave das aplicações *Web* 2.0, sendo utilizado em algumas aplicações do *Google*, como por exemplo, o *Gmail*, entre outros [20].

3.1.3. *Javascript*

O *Javascript* é uma linguagem de programação *client-side* que surgiu com o objectivo de tornar as páginas da *Web* mais interactivas. O *Javascript* pode ser utilizado para validar dados e executar processos antes de fazer uma nova solicitação ao servidor, aliviando processamento e tráfego de rede do servidor. O grande problema do *Javascript* é que nem todos os navegadores tem o mesmo suporte, interpretam o código da mesma maneira ou tem os mesmos recursos. Há muito pouco tempo era comum criar várias versões do mesmo código para que o *site* funcionasse em mais de um navegador. Isso implica um maior esforço de desenvolvimento e uma dificuldade acrescida em actualizar o código.

Com o grande crescimento da *Web* surgiu a necessidade de normalizar os seus recursos. A padronização obriga directamente os navegadores a se adequarem às suas normas. Isso simplifica o desenvolvimento porque torna desnecessário o trabalho de criar várias versões do mesmo *site*, mas ainda há muitos conflitos que não podemos deixar para trás. Estes conflitos têm que ser tidos em consideração durante o desenvolvimento de modo a evitar surpresas no futuro.

3.1.4. *Open Data Base Connectivity (ODBC)*

O *ODBC* é um padrão para acesso a sistemas de bases de dados. Este padrão define um conjunto de *interfaces* que permitem o uso de linguagens de programação como *PHP*, *Visual Basic*, *Delphi*, *Visual C++*, entre outras capazes de utilizar estas *interfaces*, para ter acesso a bases de dados distintas sem a necessidade de codificar métodos de acesso especializados.

O *ODBC* possui uma implementação específica da linguagem *SQL* com a qual a aplicação pode comunicar com a base de dados de forma transparente, permitindo, por exemplo, que um mesmo programa possa utilizar simultaneamente o *MySQL, o Access, o SQL Server e o Oracle* sem a necessidade de mudanças na sua camada de dados.

Devido a estas características foi escolhido este padrão para acesso a sistemas de bases de dados tendo sido utilizado na *Plataforma de Acesso a Dados* de forma a permitir uma abstracção.

A utilização destas *interfaces* está condicionada à existência de *drivers ODBC* específicos para as bases de dados que se deseja aceder.

3.1.5. *Agent Communication Language (ACL)*

Um dos objectivos principais da plataforma desenvolvida é que seja modular e adaptativa. Para isso, a comunicação com a plataforma é realizada com base em mensagens. A utilização de mensagens possibilita um desacoplamento entre partes da aplicação, tornando-a mais modular e adaptativa. Para dar sustentabilidade ao modelo de mensagens utilizado foi escolhido o modelo de mensagens definido pela *FIPA ACL* [23].

A estrutura das mensagens *ACL* foi definida de forma a abranger diferentes tipos de aplicações (ver Tabela 3.2), no caso da plataforma a desenvolver foi definida uma mensagem aplicacional tal como apresentado na Tabela 4.1.

Parâmetro	Descrição
Performative	Indicação do tipo de pedido.
Sender	Identificação do componente que envia a mensagem.
Receiver	Identificação do componente que recebe a mensagem.
Reply-to	Este parâmetro indica que as mensagens posteriores à conversação são para ser direccionadas para o agente nomeado neste parâmetro, em vez do agente nomeado no parâmetro "*Sender*".
Content	Conteúdo da mensagem.
Language	Linguagem em que se encontra o conteúdo.
Encoding	Indica a codificação do conteúdo.
Ontology	Indica a ontologia usada para dar um significado aos símbolos usados no conteúdo da mensagem.
Protocol	Protocolo de comunicação.
Conversation-id	Identificador da conversação.
Reply-with	Serve para introduzir uma expressão que será utilizada pelo agente de resposta para identificar esta mensagem.
In-reply-to	Indica uma expressão que faz referência a uma acção anterior em que a mensagem é uma resposta.
Reply-by	Indica um tempo e, ou, uma data que indica o último tempo pelo qual o agente de envio gostaria de receber uma resposta.

Tabela 3.2 – Mensagem *ACL*

Poderá ser obtida uma explicação mais pormenorizada dos parâmetros da mensagem *ACL* consultando a referência [23].

Tendo por base o formato geral de uma mensagem *FIPA ACL*, foi definido um formato de mensagem específico, de modo a suportar as necessidades específicas da plataforma, o qual será apresentado em *4.2 Mensagens Aplicacionais*.

3.1.6. *Java Script Object Notation (JSON)*

O *JSON* é um formato leve para troca de dados computacionais. É baseado num subconjunto da linguagem de programação *JavaScript, Standard ECMA-262 3rd Edition* -

December 1999 [24]. *JSON* é um formato de texto que é completamente independente da linguagem, mas usa convenções que são familiares para os programadores de linguagens da família *C*, incluindo *C, C + +, C #, Java, JavaScript, Perl, Python*, e muitos outros.

O *JSON* é uma alternativa ao *XML* mas das vantagens reivindicadas do *JSON* sobre o *XML* como um formato para troca de dados, é o facto de ser muito mais fácil escrever um analisador *JSON*, sendo esta uma característica que levou à escolha do *JSON*.

Mas o facto de o *JSON* ser uma alternativa ao *XML* não invalida que ambos possam ser usados na mesma aplicação.

Outro factor para a escolha do *JSON* tem a ver com o facto deste formato ser tipicamente usado em ambientes onde se verifica um grande fluxo de dados entre o cliente e o servidor (daí a sua utilização por exemplo no *Google, Yahoo,* etc.) [24].

3.2. Estratégia de Acção

Um dos principais objectivos de um sistema de informação é a satisfação adequada dos requisitos de negócio, garantindo assim o correcto alinhamento com a estratégia da organização/cliente. Nesse sentido, um dos aspectos principais da plataforma desenvolvida é o foco da solução ser baseado nos requisitos (Figura 3.1) de modo a desenvolver-se uma aplicação *Web* que responda a esses requisitos.

Foram definidos três modelos principais que enquadram o desenvolvimento de uma aplicação, nomeadamente:

- Modelo de Informação - especifica as entidades de informação derivadas dos requisitos a utilizar por cada aplicação.
- Modelo de Apresentação – especifica o formato de apresentação da informação aos utilizadores.
- Modelo de Dados – especifica o esquema de dados e o acesso aos dados.

A Figura 3.1 ilustra a relação entre estes modelos e a especificação de requisitos.

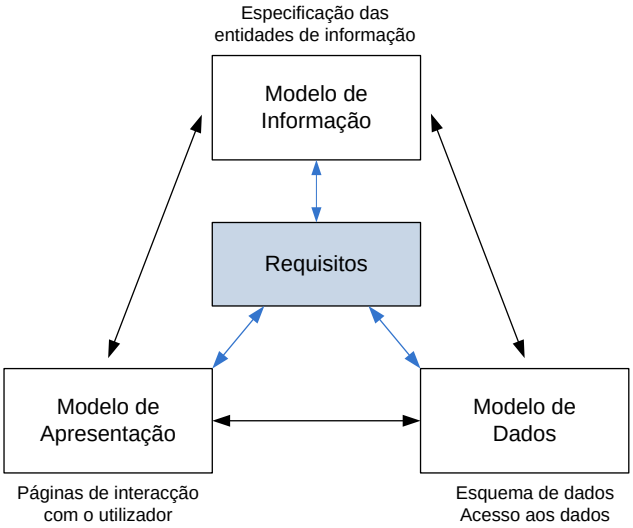


Figura 3.1 – Relação entre modelos e os requisitos

A estratégia adoptada visa reduzir o acoplamento entre os vários subsistemas e aumentar a coesão dos módulos funcionais.

A redução do nível de acoplamento permite:

- Maior facilidade de desenvolvimento, instalação, manutenção e expansão;
- Melhor escalabilidade, devido à possibilidade de distribuição e replicação de módulos que prestem serviços, sem que isso tenha um impacto significativo nos clientes desses subsistemas/módulos;
- Maior tolerância a falhas, logo maior robustez, uma vez que a falha de um subsistema/módulo tem um impacto restrito.

Aumentar a coesão leva a que, em caso de necessidade de alteração de um subsistema, o número de módulos afectados seja minimizado. Um módulo com um nível de coesão baixo é mais complexo, logo mais difícil de conceber e de testar.

Para reduzir o acoplamento entre os vários subsistemas e aumentar a coesão separou-se o modelo de apresentação do modelo de dados (Figura 3.2) permitindo assim, que o desenvolvimento e os testes de cada um dos modelos fossem realizados independentemente um do outro, simplificando desta forma o ambiente de desenvolvimento e de operação.

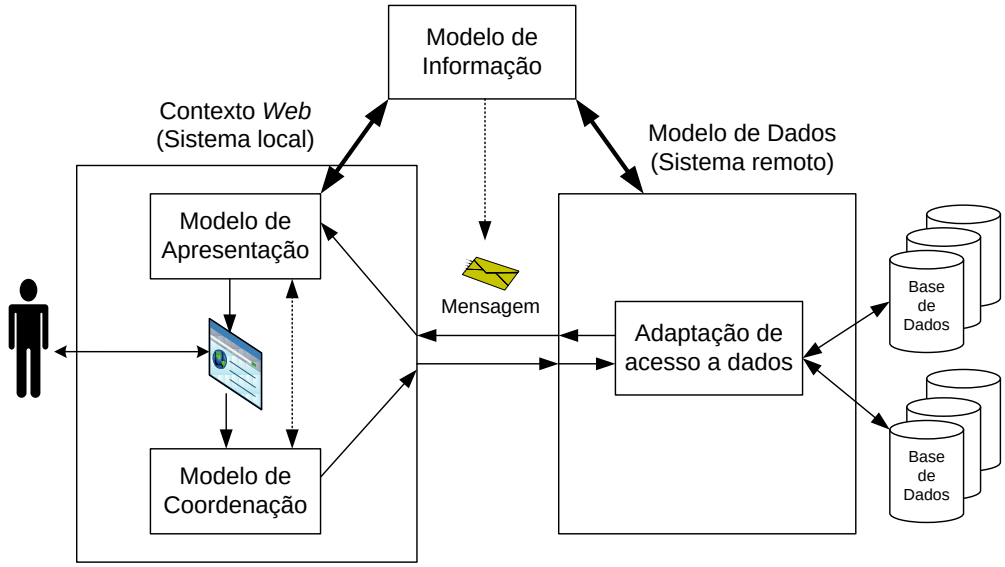


Figura 3.2 – Separação do Modelo de Apresentação do Modelo de Dados

O contexto *Web* para além de ser constituído pelo modelo de apresentação que permite construir as *interfaces* de utilização *Web* também é constituído pelo modelo de coordenação responsável pelo fluxo de páginas. O Modelo de Apresentação está relacionado com o Modelo de Informação para se especificar a estrutura das mensagens a ser utilizada, suportando assim a adaptação automática do formato dos dados, tal como definido pelo Modelo de Dados, ao formato de apresentação, definido pelo Modelo de Apresentação. Todos os modelos mencionados anteriormente são alimentados pelos requisitos, tal como ilustrado na Figura 3.1.

O método de trabalho adoptado ênfatiza as seguintes características de desenvolvimento:

- Normalização, reutilização e abstracção (ver definição na secção seguinte);
- Obter versões funcionais em menos tempo;
- Requisitos próximos da operação:
    - Especificação de contextos de utilização, ou seja, definição de acções a realizar pelo utilizador;
    - Ênfase na *interface Web* e nos dados.
- Complexidade incremental;
- Usabilidade (ver secção seguinte).

No que diz respeito ao desenvolvimento da plataforma, ela foi concebida tendo em conta todas as premissas referidas anteriormente, no sentido de garantir modularidade (ver secção seguinte), simplicidade e uma organização funcional coesa.

3.2.1. *Conceitos base*

De seguida, são apresentados os conceitos / princípios que foram tidos em consideração durante o desenvolvimento da plataforma.

A **abstracção** é um dos princípios para se gerir a complexidade no desenvolvimento de *software*. Ela permite analisar o problema com um nível de generalização, independentemente dos detalhes de implementação [25], ou seja, a abstracção é a representação concisa duma ideia ou objecto mais complexa, incidindo sobre as características essenciais do objecto [26].

A **modularidade** é outro dos princípios para se gerir a complexidade no desenvolvimento de *software*. Modularidade é a decomposição lógica e física de conceitos em unidades mais elementares, de forma a facilitar a aplicação dos princípios da engenharia de *software* [26]. A modularidade contribui para uma melhor compreensão dos sub-problemas que o *software* é suposto resolver, para uma melhor integração, e para facilitar a introdução das alterações em módulos específicos, o que reduz o impacto de falha nos restantes módulos e consequentemente em todo o sistema [25].

A **usabilidade** é um dos factores de qualidade mais importantes para as aplicações *Web*. Aplicações difíceis de utilizar levam os utilizadores a rejeitá-las. O objectivo é, portanto, projectar aplicações *Web* com as quais os utilizadores possam atingir os seus objectivos de forma eficaz, eficiente e satisfatória. Para este fim, no desenvolvimento de aplicações *Web*, tem que se considerar os contextos de utilização e as necessidades dos utilizadores.

De acordo com [1], existe um conjunto de orientações que devem ser seguidas para se desenvolver aplicações com boa usabilidade. De seguida, serão apresentadas algumas delas que foram tidas em consideração no desenvolvimento:

- **Interacção Eficiente** - *interfaces* eficientes devem minimizar o número de acções por parte do utilizador. Os itens de maior interesse devem ser atingidos

com o mínimo de acções possível. O tamanho dos elementos não deve ser inferior a um certo mínimo para se garantir que os utilizadores conseguem atingir os elementos com precisão.

- **Estrutura da Página** – para assegurar uma orientação fácil na página, deve ser evitada a necessidade de realizar *scrolling* horizontal porque na leitura linha a linha o utilizador teria que permanentemente mover a janela da esquerda para a direita e vice-versa, o que tornaria a leitura muito difícil.
- **Estrutura de navegação** – a navegação através de um *website*/aplicação representa um aspecto particularmente importante para a usabilidade do *website*/aplicação. O sistema de navegação deve ter uma estrutura clara e lógica, de forma a permitir ao utilizador uma navegação simples. Para isso, o sistema de navegação deve possibilitar ao utilizador obter a sua localização (onde estou?), informações claras sobre a página corrente (o que posso fazer ou encontrar nesta página?) e quais os itens que é possível atingir após a interacção (para onde posso ir?).

A **prototipagem** é um método de desenvolvimento que permite ao cliente visualizar e validar um modelo da *interface* (e das funcionalidades) que irá ter disponível, reduzindo também os problemas da comunicação. A utilização de técnicas de prototipagem logo no início do processo de desenvolvimento possibilita ir ao encontro das expectativas do cliente mais rapidamente.

Antes da prototipagem é necessário realizar uma análise de requisitos. Após esta análise, são realizados rascunhos das *interfaces* necessárias para responder aos requisitos. A vantagem de se realizar rascunhos das *interfaces* tem a ver com o facto das *interfaces* estarem sujeitas a algumas alterações e de ser mais fácil realizar essas alterações nos rascunhos do que no código. O tempo utilizado nas revisões das *interfaces* pode eliminar horas/semanas de codificação [27].

Como não há "rosas sem espinhos", é preciso neste caso ter um cuidado acrescido de forma a não ceder às naturais pressões do cliente, no sentido deste passar a utilizar de imediato o protótipo. A satisfação da pretensão do cliente significaria comprometer a qualidade do produto em detrimento de "ter qualquer coisa a funcionar". Existe ainda o perigo de se perder a visão mais global e abrangente do sistema e respectivas

funcionalidades em detrimento de uma visão mais restrita e imediatista baseada em sequências de ecrãs.

O protótipo é normalmente constituído por uma sequência de ecrãs, através dos quais é possível ver como uma interacção pode funcionar, como se pode navegar num *website* através dos ecrãs, ou como os elementos se comportam. Nos protótipos não importa o modo como são construídos, se são realmente bons, ou se eles são simulações exactas de um projecto futuro. O que importa é demonstrar que o seu funcionamento vai ao encontro dos requisitos definidos.

O principal benefício de um protótipo é possibilitar ao cliente visualizar a sequência de ecrãs da aplicação e interagir com os elementos que fazem parte dos ecrãs. Esta interacção permite verificar o comportamento da aplicação e assim obter uma noção real do que a aplicação irá fazer.

Além do protótipo permitir ao utilizador usar a aplicação, o protótipo pode ser usado para obter *feedback* sobre a *interface* antes de serem efectuados ajustes profundos. Escrever código é dispendioso, e realizar alterações numa aplicação leva tempo valioso e energia. Fazer alterações substanciais no final do desenvolvimento faz com que essas alterações sejam difíceis de realizar e demoradas, podendo dar origem a exceder o prazo estabelecido. Por outro lado, mudar o comportamento de uma interação num protótipo é uma tarefa rápida e fácil. De facto, podem ser realizadas várias alterações em pouco tempo, explorando diferentes soluções para cada problema.

Ao realizar as alterações num protótipo é importante obter novamente *feedback* do cliente para verificar se as alterações vão ao encontro do que é pretendido. Se o cliente não ficar satisfeito com a nova versão do protótipo, facilmente é alterado para a versão anterior.

Outra vantagem chave para a prototipagem é que se tem uma *interface* ao longo de todo o processo de desenvolvimento, contribuindo desta forma para limitar a adição de novos elementos que possam interferir com a visão do cliente e consequentemente dispersá-lo. A *interface* também fornece uma imagem clara de como o projecto final irá funcionar e o que precisa de ser realizado. Trabalhar com uma *interface* desde muito cedo possibilita que possa ser melhorada gradualmente.

4. Arquitectura da Plataforma

Este capítulo descreve a arquitectura da plataforma, como esta se encontra organizada e os aspectos de segurança.

A arquitectura da plataforma divide-se em duas áreas principais, uma associada à vertente *Web*, plataforma cliente, e outra associada à vertente de acesso a dados, plataforma servidora, como se encontra apresentado na Figura 4.1.

Na vertente *Web* estão associados dois subsistemas, o *Subsistema de Apresentação* e o *Subsistema de Coordenação*. O *Subsistema de Apresentação* é responsável pela construção de páginas de interacção com o utilizador com todas as características associadas a uma página *Web*, nomeadamente, com validação de formulários, com filtros de pesquisa quando a página assim o justifique, com a possibilidade de ordenar colunas de tabelas, entre outras características. O *Subsistema de Coordenação* é responsável por disponibilizar as funcionalidades de acordo com o perfil de utilizador, tendo em consideração os requisitos definidos e por gerir o fluxo entre as várias páginas construídas pela plataforma.

A plataforma servidora é constituída pelo *Gestor de Acessos*, pelo *Subsistema de Acesso a Dados* e pelo *Subsistema de Autenticação e Gestão de Sessões*.

O *Gestor de Acessos* tem a responsabilidade de registar as acções/serviços possíveis da plataforma tendo em consideração cada perfil de utilizador e por validar se os pedidos efectuados são válidos tendo em conta o perfil que solicitou a acção/serviço.

O *Subsistema de Acesso a Dados* é responsável por interpretar o conteúdo das mensagens, por realizar a respectiva tradução para *SQL* e por traduzir o resultado de uma *query SQL* para o conteúdo de uma mensagem.

O *Subsistema de Autenticação e Gestão de Sessões* é responsável pela autenticação dos utilizadores e por gerir o estados das sessões.

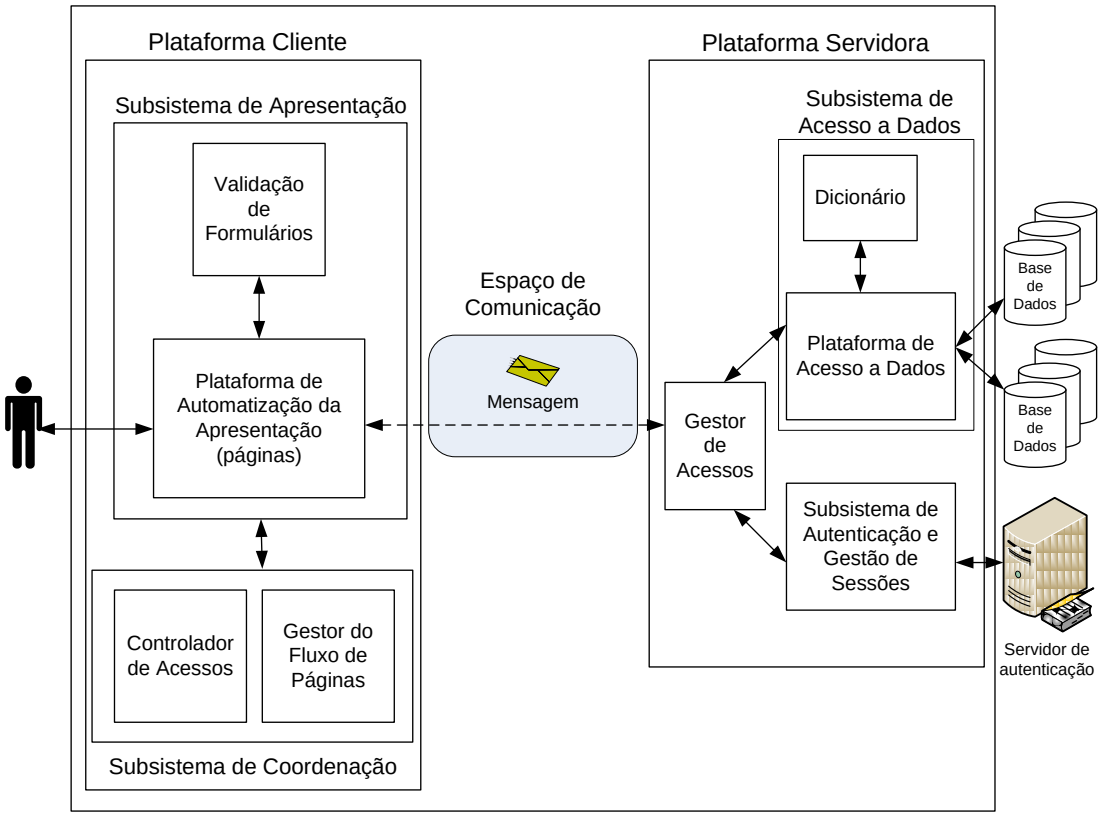


Figura 4.1 – Arquitectura da Plataforma

A comunicação entre os vários componentes da plataforma é feita com base em mensagens. A utilização de mensagens possibilita um desacoplamento entre partes da aplicação, tornando-a modular e adaptativa. Em particular, a utilização de mensagens para a comunicação entre componentes tem como benefícios não fazer um componente depender da existência de outro e poder dissociar completamente os componentes da origem dos seus dados.

Todos os parâmetros da mensagem aplicacional são utilizados na *Plataforma de Automatização da Apresentação*, mas apenas os parâmetros *Subject*, *Content* e *Result* são usados na *Plataforma de Acesso a Dados*.

Para permitir que a plataforma tenha a capacidade de executar múltiplas acções (leitura, escrita, actualização e remoção) em simultâneo associadas a uma mensagem, dotou-se a *Plataforma de Acesso a Dados* com essa capacidade. Para isso, o parâmetro *Subject* mostrou-se importante porque, para além de indicar o assunto da mensagem, permitiu na *Plataforma de Acesso a Dados* definir as operações que estão associadas ao respectivo assunto da mensagem. Na secção *5.2 Plataforma Servidora* é descrito como isso é conseguido.

A comunicação entre a *Plataforma de Automatização da Apresentação* e a *Plataforma de Acesso a Dados* é feita por objectos *JSON*, que representam as mensagens *ACL*. Compete à plataforma que pretender enviar a mensagem o papel de codificar a mensagem para *JSON* e à plataforma que recebe a mensagem o papel de descodificar a mensagem *JSON*.

No caso de serviços já existentes será necessário adaptá-los para que funcionem com a plataforma, ou que seja criada uma camada de tradução capaz de comunicar com os serviços já existentes.

Para além da troca de informação em *JSON* a plataforma tem a possibilidade de permitir trocar informação em *XML*, ou outros formatos, desde que o componente que recebe essa informação esteja preparado para a processar. Embora se crie o vínculo da necessidade da interpretação de mensagens neste formato (nas várias linguagens possíveis), a comunicação torna-se independente de forma temporal e funcional, pelo que todas as acções poderão ser consideradas como acções disjuntas, libertando o programador de toda a lógica de estado, sendo esta lógica processada no serviço utilizado.

Como foi referido anteriormente, a comunicação entre os vários componentes da plataforma é feita com base em mensagens originadas a partir de requisitos. Os requisitos são descritos em termos de funcionalidade e de estrutura de informação. A descrição funcional é feita com base em casos de utilização. A descrição da estrutura de informação é feita com base em entidades de informação (Modelo de Informação, que se encontra explicado na secção seguinte).

Os casos de utilização dão origem a páginas (vertente de apresentação) e a serviços (vertente de lógica aplicacional).

A plataforma desenvolvida suporta a construção de páginas através da interpretação de mensagens (ver secção *4.2 Mensagens Aplicaciona*) e recorrendo ao *Subsistema de Apresentação*. A disponibilização de serviços é conseguida através da sua configuração na plataforma servidora (ver secção *4.4 Plataforma Servidora*).

4.1. Modelo de Informação

Tal como apresentado anteriormente, o modelo de informação especifica as entidades de informação derivadas dos requisitos de informação. Uma entidade de

informação é uma representação de um tipo de informação que se pretenda descrever. Um exemplo disso será a entidade de informação *Pauta*, que é representada por um conjunto de atributos tais como, unidade curricular, código da unidade curricular, ano lectivo, período lectivo, responsável da unidade curricular e pela entidade de informação *AlunoPauta* que representa os alunos que fazem parte da pauta. A entidade *AlunoPauta* é representada pelos atributos número do aluno, nome do aluno, turma, classificação e data da avaliação. A figura seguinte apresenta as duas entidades de informação referidas anteriormente e a relação entre elas. O símbolo * representa uma ou mais entidades de informação do tipo indicado.

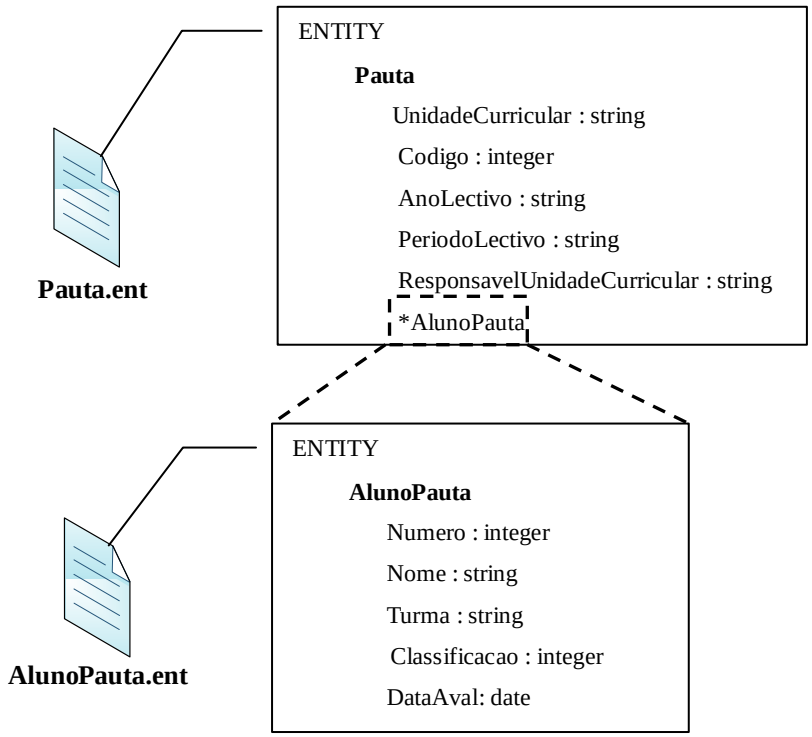


Figura 4.2 – Relação entre entidades de informação

A mensagem que será utilizada pela plataforma desenvolvida pode incluir dados referentes a diferentes entidades de informação (ver secção seguinte). Uma vez que cada caso de utilização tem associado uma (ou mais) entidade(s) de informação, e tendo em conta que é a partir desta que é construída a mensagem, foi implementado um módulo (*EntityInformation2Message.js*) capaz de construir a mensagem através da descrição da entidade. A forma de representação da entidade de informação pretende-se que seja o mais simples possível, de modo a que seja realizada e interpretada por qualquer pessoa mas que possua ao mesmo tempo toda a informação necessária para que o módulo

referido anteriormente consiga construir a mensagem correctamente. O módulo pode ser visto como um interpretador e um tradutor. Interpretador porque interpreta o conteúdo da entidade de informação e tradutor porque traduz esse conteúdo para o formato correspondente na mensagem *ACL*.

O desenvolvimento da plataforma foi efectuado considerando as mensagens *ACL* no formato estabelecido (*JSON*).

Cada caso de utilização do sistema está associado a uma mensagem que corresponde aos requisitos pretendidos, e essa mensagem irá dar origem a uma página *Web*. Por exemplo, o caso de utilização *"Consultar Pauta de Alunos"* tem associada a mensagem *"ConsultarPauta"* que é constituída pela informação que permite visualizar a pauta de alunos.

4.2. Mensagens Aplicacionais

Na definição das mensagens aplicacionais teve-se em consideração alguns aspectos de modo a que a plataforma fornecesse determinadas funcionalidades, nomeadamente, a validação de campos da aplicação *Web*, a tradução de campos da aplicação para um idioma, a criação dinâmica de tabelas, que desse a possibilidade ao utilizador de este indicar se as tabelas criadas são tabelas que permitem seleccionar linhas, entre outras funcionalidades que irão ser explicadas de forma mais pormenorizada na secção *5.1.1.1 Componentes Gráficos* e Funcionalidades Desenvolvidas. Para a validação de campos da aplicação *Web* definiu-se o parâmetro *Types*, para a tradução de campos da aplicação para um idioma definiu-se o parâmetro *Translate* e para a criação dinâmica de tabelas definiu-se o parâmetro *Inner*. Na Tabela 4.1 encontra-se uma explicação mais pormenorizada de cada parâmetro.

Parâmetro	Descrição
Performative	Indica o tipo de pedido (modo leitura ou modo de edição).
Subject	Assunto da mensagem.
Translate	Idioma para o qual se pretende traduzir os atributos.
Types	Tipos associados aos atributos existentes na mensagem. Para além do tipo do atributo, é indicado a respectiva dimensão, se é um tipo obrigatório e qual o tipo de componente (simples, seleccionável, ou outro).
Outer	Atributos da mensagem. Estes atributos podem conter atributos filhos que se encontram descritos no parâmetro *Inner*.
Inner	Atributos da mensagem que são filhos de outros atributos.
Content	Conteúdo da mensagem.
Result	Indica se a acção realizada sobre o serviço foi bem sucedida, ou não. Pode conter uma mensagem informativa para ser apresentada ao utilizador.
TablesWithMultipleChoice	Identificadores das tabelas que permitem seleccionar múltiplas linhas. Assim, é possível especificar quais as tabelas da página em que é possível seleccionar múltiplas linhas.

Tabela 4.1 – Formato base das mensagens aplicacionais utilizadas na plataforma

A Figura 4.3 apresenta o formato das mensagens aplicacionais que são utilizadas na plataforma e a relação entre os atributos.

Após o levantamento dos requisitos, o que origina as entidades de informação e os casos de utilização, é necessário relacionar esses requisitos com a mensagem aplicacional. Para isso, é indicado no parâmetro ***"subject"*** o caso de utilização, sendo os requisitos associados a esse caso de utilização indicados no parâmetro ***"outer"***. Se existirem requisitos que estão associados a outras entidades de informação é necessário especificar esses requisitos com a marca *"set"*, e indicar o nome da entidade de informação correspondente, como é o caso do *"atributo 2"* que está associado à entidade de informação *"Ent. Informação 2"* (Figura 4.3). Os requisitos associados a esta entidade de informação são especificadas no campo ***"inner"*** e para cada requisito é especificado o seu tipo que se encontra definido no parâmetro ***"types"***. Os requisitos que

não estiverem associados a entidades de informação são especificados com a marca *"type"* e com o respectivo tipo.

Todos os parâmetros das mensagens aplicacionais são utilizados pela plataforma cliente mas apenas os parâmetros ***"subject"***, ***"content"*** e ***"result"*** são usados na *Plataforma de Acesso a Dados*.

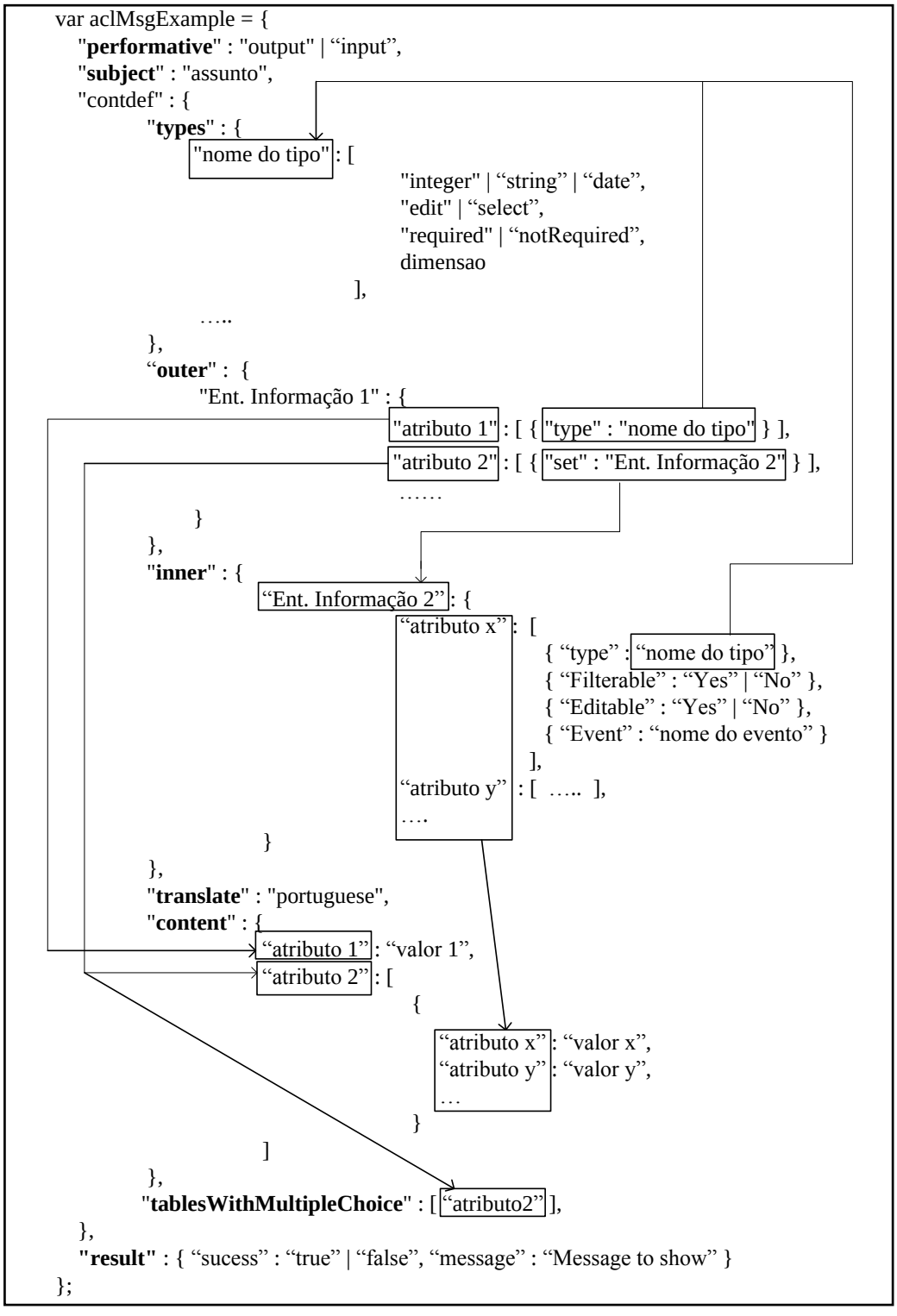


Figura 4.3 – Formato base das mensagens aplicacionais

De seguida, são explicados os parâmetros da mensagem (identificados a ***bold***):

- Parâmetro **"*performative*"**:

  Este parâmetro indica se a página é para ser construída em modo leitura (*"output"*) ou em modo de edição (*"input"*).

- Parâmetro **"*subject*"**:

  Indica o assunto da mensagem e está associada a um caso de utilização.

- Parâmetro **"*types*"**:

  Indica os tipos associados aos atributos existentes na mensagem. Estes tipos encontram-se definidos no componente *ValidateInput* e é este que é responsável por validar cada tipo.

```
"types" : {
   "nome do tipo" : [
         "integer" | "string" | "date",
         "edit" | "select",
         "required" | "notRequired",
         dimensao
   ]
   ...
 }
```

Exemplo 4.1 – Parâmetro *types* da mensagem aplicacional

Cada tipo tem associado:

- 1ª posição - o seu tipo (*integer*, *string*, *date*, ou outro);
- 2ª posição - o tipo de componente a apresentar em *HTML* (caixa de texto, *combo box*, *textarea*, ou outro);
- 3ª posição - a indicação se é um campo obrigatório;
- 4ª posição - número de caracteres do valor.

- Parâmetro ***"outer"***:

```
"outer" :  {
   "Ent. Informação 1" : {
        "atributo 1" : [ { "type" : "nome do tipo" } ],
        "atributo 2" : [ { "set" : "Ent. Informação 2" } ],
         ……
    }
}
```

Exemplo 4.2 – Parâmetro *outer* da mensagem aplicacional

O parâmetro ***"outer"*** é constituído por entidades de informação que por sua vez possuem os respectivos atributos. No exemplo anterior existe apenas uma entidade de informação que tem o nome *"Ent. Informação 1"* e os seus atributos são *"atributo 1"* e *"atributo 2"*. O atributo *"atributo 2"* por sua vez corresponde a outra entidade de informação (*set*), nomeadamente, *"Ent. Informação 2"*, cujos atributos encontram-se definidos no parâmetro "***inner"***, explicado a seguir. Atributos do tipo *set* serão apresentados na página *Web* em forma de tabela.

- Parâmetro ***"inner"***:

```
"inner" : {
   "Ent. Informação 2" : {
      "atributo x" : [
                          { "type" : "nome do tipo" },
                          { "Filterable" : "Yes" | "No" },
                          { "Editable" : "Yes" | "No" },
                          { "Event" : "nome do evento" }
       ],
       "atributo y"  : [  ...  ]
   }
}
```

Exemplo 4.3 – Parâmetro *inner* da mensagem aplicacional

O parâmetro ***"inner"*** é constituído por entidades de informação e os respectivos atributos. No Exemplo 4.3, a entidade de informação *"Ent. Informação 2"* tem associados os atributos *"atributo x"* e *"atributo y"*. As entidades de informação definidas no parâmetro ***"inner"*** serão apresentadas na

página *Web* em forma de tabela. Cada atributo representa uma coluna na tabela da página *Web*, e tem associado:

- 1ª posição ("*type*") – o tipo do atributo que se encontra definido no parâmetro ***"types"***;
- 2ª posição ("*Filterable*") – indica se o atributo está associado a um filtro de pesquisa, ou seja, se o valor for *true* é para criar uma *combo box* que permite filtrar a tabela tendo em conta o valor seleccionado.
- 3ª posição ("*Editable*") – indica se o campo é editável mesmo quando o ***"performative"*** associado à mensagem é *output* (modo de leitura). Se o valor for *true*, é criado um botão que permite editar o campo.
- 4ª posição, opcional ("*Event*") – indica o evento associado ao atributo. Isto faz com que seja criado um *link* para o atributo.

- Parâmetro ***"translate"***:

Indica em que idioma se pretende traduzir os termos (*labels*, nome das tabelas, cabeçalhos das tabelas, etc.) a apresentar na página *Web*.

- Parâmetro ***"content"***:

```
"content" : {
   "atributo 1" : "valor 1",
   "atributo 2" : [
                     {
                         "atributo x" : "valor x",
                         "atributo y" : "valor y",
                         ...
                     }
                  ]
}
```

Exemplo 4.4 – Parâmetro *content* da mensagem aplicacional

O parâmetro ***"content"*** é constituído pelos atributos definidos no parâmetro ***"outer"*** e ***"inner"***. Para os atributos definidos no parâmetro ***"inner",*** e que correspondem a entidades de informação, é atribuído para cada atributo o respectivo valor.

Os valores associados a cada atributo são os que serão apresentados na página *Web*.

- Parâmetro ***"tablesWithMultipleChoice"***:

  Este parâmetro permite especificar as entidades de informação que correspondem a tabelas na página *Web* que se pretende que permitam seleccionar múltiplas linhas. Para isso, apenas é necessário indicar no parâmetro ***"tablesWithMultipleChoice"*** o nome da entidade de informação que se encontra definido no parâmetro ***"inner"***.

- Parâmetro ***"result"***:

  Este parâmetro permite indicar se a operação foi realizada com sucesso e pode conter uma mensagem que será apresentada ao utilizador na página *Web*.

4.3. Plataforma Cliente

A plataforma cliente é a responsável por automatizar a apresentação da aplicação *Web*, ou seja, é responsável pela construção das páginas que fazem parte da aplicação *Web*. A construção de cada página é realizada com base numa mensagem *ACL*, o que significa que cada mensagem *ACL* origina uma página.

O fluxo de páginas e a validação dos dados introduzidos pelo utilizador são acções que são executadas pela plataforma cliente, por componentes que irão ser apresentados ao longo desta secção.

No contexto da plataforma global, os serviços que serão invocados pela plataforma cliente correspondem às acções associadas aos casos de utilização que se obtiveram dos requisitos (operações de leitura, de escrita, de actualização, de remoção, entre outras).

4.3.1. *Organização de Componentes da Plataforma de Automatização da Apresentação*

A *Plataforma de Automatização da Apresentação*, que compõe a plataforma cliente, é constituída pelos componentes, *PlatformLoader*, *InterpreterMessage*, *BuildPage* e *HTML_Component*.

A Figura 4.4 apresenta a organização dos componentes referidos anteriormente.

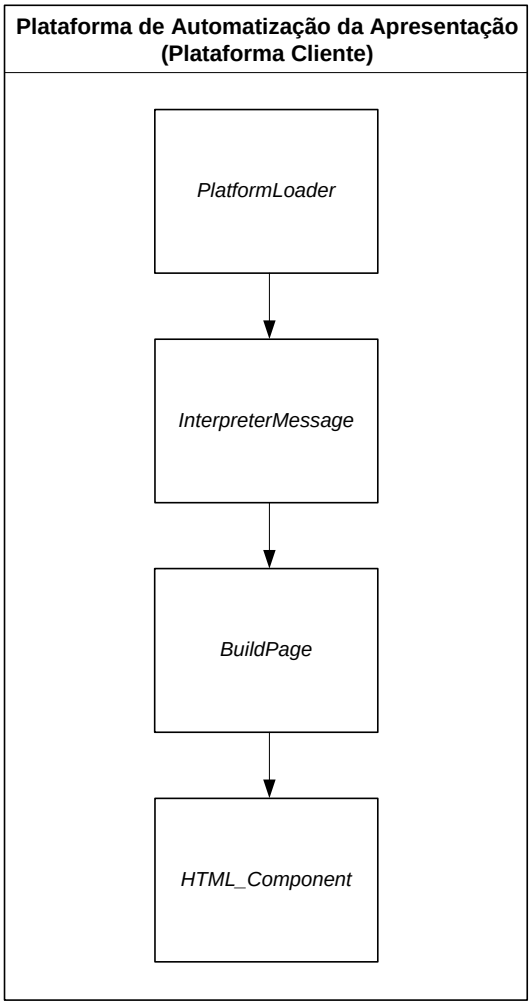


Figura 4.4 – Organização dos componentes da *Plataforma de Automatização da Apresentação*

Para que a utilização da plataforma seja mais facilitada foi criado o componente *PlatformLoader* que efectua o carregamento da plataforma através da inclusão de um único ficheiro *Javascript* na página.

O componente *InterpreterMessage* interpreta a mensagem e com base no formato definido constrói os componentes *HTML* através do componente *HTML_Component*. Este componente contém métodos de criação de componentes base definidos a partir dos requisitos elaborados iterativamente durante o desenvolvimento. Para além da criação de componentes também disponibiliza métodos que permitem atribuir funcionalidades aos componentes, por exemplo, o componente tabela tem associado a funcionalidade de

ordenação, de pesquisa, entre outras. Estas funcionalidades encontram-se descritas na secção *5.1.1.1 Componentes Gráficos e Funcionalidades Desenvolvidas*.

O componente *BuildPage* será o responsável por construir a estrutura da página tendo em conta o formato da mensagem, sendo este que irá construir todos os componentes *HTML* necessários recorrendo ao componente *HTML_Component*.

Na *Plataforma de Automatização da Apresentação* são utilizados os *performatives output* e *input*, para permitir criar a página em modo leitura ou em modo de edição, respectivamente. É com base no *performative* que a plataforma cliente irá "saber" se as páginas da aplicação *Web* são para ser construídas em modo de leitura ou em modo de edição.

Associada a esta plataforma existe o componente responsável por efectuar a validação de formulários (campos) da página *Web* (ver Figura 4.1) que é responsável por validar os valores introduzidos pelo utilizador e apresentar mensagens consoante o erro verificado. Esta validação é baseada no tipo associado ao campo que se encontra definido na mensagem.

4.3.2. *Organização de Componentes do Subsistema de Coordenação*

O *Subsistema de Coordenação* é responsável por disponibilizar as funcionalidades de acordo com o perfil de utilizador e por gerir o fluxo entre as várias páginas construídas pela plataforma.

O *Subsistema de Coordenação* é constituído por dois componentes, o *Controlador de Acessos* (*AccessController*) e o *Gestor do Fluxo de Páginas* (*FlowManager*).

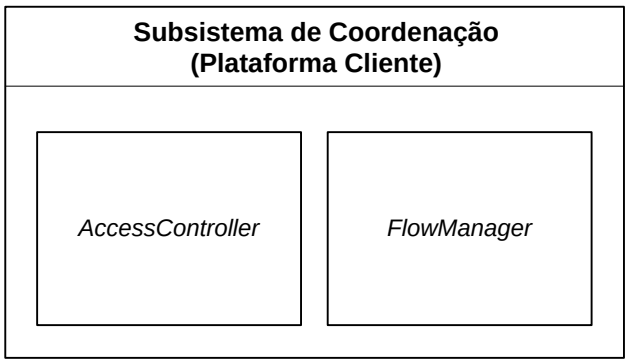


Figura 4.5 – Subsistema de Coordenação (Plataforma Cliente)

O componente *Controlador de Acessos* é responsável por definir as funcionalidades para cada perfil de utilizador, tendo em conta os requisitos definidos, e por disponibilizar essas mesmas funcionalidades de acordo com o perfil de utilizador.

É este componente que disponibiliza as opções dos *menus* consoante o contexto onde o utilizador se encontra e o perfil do utilizador. A forma como são definidas as funcionalidades e como são disponibilizadas as opções dos *menus* é explicado na secção *5.1.2 Subsistema de Coordenação*.

O componente *Gestor do Fluxo de Páginas* é responsável por gerir o fluxo entre as várias páginas construídas pela plataforma. É este componente que interpreta a interacção do utilizador e redirecciona o utilizador para a página pretendida.

4.4. Plataforma Servidora

A arquitectura servidora definida é uma arquitectura orientada a serviços (*SOA*), baseada em *endpoints HTTP*. Não tendo sido implementado um serviço de descoberta ou gestão de serviços, todos os que se pretendem utilizar terão de ser pré-configurados na plataforma (ver secção *5.3 Configuração da Plataforma*).

Os serviços correspondem às acções associadas aos casos de utilização que se obtiveram dos requisitos (operações de leitura, de escrita, de actualização, de remoção, entre outras).

Para o funcionamento da plataforma é necessário responder às *interfaces* externas para compatibilidade com a plataforma, já que a sua implementação pode ser feita na linguagem que se achar conveniente e o seu comportamento programado consoante a situação específica.

Para além do serviço de acesso a dados foi implementado um serviço de autenticação *LDAP* [28].

De seguida, irá apresentar-se a *interface* das mensagens quer para o serviço de acesso a dados, quer para o serviço de autenticação *LDAP*. As *interfaces* das mensagens têm de ser respeitadas para a fácil integração com a plataforma.

4.4.1. *Serviço de acesso a dados*

Neste serviço é necessário realizar a adaptação entre o Modelo de Informação e o Modelo de Dados, e vice-versa, possibilitando desta forma adaptar o acesso a dados. Isto é conseguido recorrendo a um dicionário que permite associar os requisitos com a especificação da base de dados (ver secção *4.4.3 Organização de componentes*).

Para a utilização do serviço disponibilizado pela *Plataforma de Acesso a Dados* é necessário que esta receba uma mensagem com os parâmetros *subject*, *content* e *result*.

O modelo de mensagens definido encontra-se apresentado na Tabela 4.2.

Assunto	Mensagem
Acção a realizar	{ "subject": "acção a realizar", "content": { "attr1" : "value1", "attr2" : "value2", ……. }, "result": { "success" : "true" | "false", "message" : "Message to show" } }

Tabela 4.2 – Modelo de mensagens do serviço de acesso a dados

Na mensagem o parâmetro *subject* indica a acção que está associada a um caso de utilização resultante do levantamento de requisitos.

O parâmetro *content* é constituído pelos elementos que se pretende corresponder com a especificação da base de dados. Na Tabela 4.2 estes elementos são *"attr1"* e *"attr2"* que irão ter uma correspondência com a base de dados. Os valores que estão associados aos elementos são para serem inseridos / actualizados na base de dados, caso a operação associada à mensagem seja de inserção ou actualização, respectivamente. Caso a operação seja de leitura, os valores associados aos elementos são obtidos da base de dados.

O parâmetro *result* é constituído pelos atributos *"success"* e *"message"*. O atributo *"success"* indica se a operação realizada sobre a base de dados foi executada com

sucesso (*"true"*), ou não (*"false"*). O atributo *"message"* indica a mensagem que se pretende mostrar ao utilizador após a execução de uma acção.

4.4.2. *Serviço de autenticação LDAP*

O serviço de autenticação *LDAP* tem por objectivo suportar a autenticação com base no nome de utilizador e palavra passe, a gestão de sessão e o acesso a uma memória de sessão com possibilidade de leitura e escrita de pares nome valor.

Estas funcionalidades são associadas aos seguintes assuntos da mensagem:

- *authentication*;
- *doLogout*;
- *getFromSession*;
- *addToSession*.

As funcionalidades associadas a cada um destes assuntos são, pela ordem apresentada (ver tabela Tabela 4.3), autenticação de um utilizador, terminar a sessão do utilizador, obtenção dos dados da sessão do utilizador com base numa chave e armazenamento de dados na sessão do utilizador. Estas funcionalidades são disponibilizadas pelo componente *AuthenticationManager*.

Para a interacção com as funcionalidades indicadas, foi definido o seguinte modelo de mensagens:

Assunto	Mensagem
authentication	{ "subject" : "authentication", "content" : { "username" : "user1", "password" : "pass1" }, "result" : { "success" : "true" | "false", "message" : "Message to show" } }
doLogout	{ "subject" : "doLogout", "result" : { "success" : "true" | "false", "message" : "Message to show" } }

getFromSession	{
	“subject” : “getFromSession”,
	“content” : {
	“attr1” : “value1”,
	“attr2” : “value2”, ….
	},
	“result” : {
	“success” : “true” | “false”,
	“message” : “Message to show”
	}
	}

addToSession	{
	“subject” : “addToSession”,
	“content” : {
	“attr1” : “value1”,
	“attr2” : “value2”, ….
	},
	“result” : {
	“success” : “true” | “false”,
	“message” : “Message to show”
	}
	}

Tabela 4.3 – Modelo de mensagens do serviço de autenticação *LDAP*

Para além do parâmetro *subject* que foi mencionado anteriormente, as mensagens são constituídas pelos parâmetros *content* e *result*. O parâmetro *content* é constituído pelos elementos que se pretende transmitir (ou obter) a cada funcionalidade/serviço de autenticação. Para o assunto *authentication* são transmitidos o *username* e a *password* do utilizador. Para o serviço *getFromSession* são obtidos os elementos, por exemplo *“attr1”* e *“attr2”*. Para o serviço *addFromSession* são transmitidos os elementos, por exemplo *“attr1”* e *“attr2”*, que se pretende armazenar na sessão do utilizador.

O parâmetro *result* está associado a todas as funcionalidades apresentadas anteriormente. O campo *result* é constituído pelos atributos *“success”* e *“message”*. O atributo *“success”* indica se a funcionalidade realizada foi executada com sucesso (*“true”*), ou não (*“false”*). O atributo *“message”* indica a mensagem que se pretende mostrar ao utilizador após a execução da funcionalidade.

4.4.3. *Organização de componentes*

A plataforma servidora é constituída pela *Plataforma de Acesso a Dados*, pelo *Gestor de Acessos* e pelo *Subsistema de Autenticação e Gestão de Sessões*. Na Figura 4.6 é apresentada a organização dos componentes da plataforma servidora.

O componente *AccessManager* (*Gestor de Acessos*) é o responsável por registar todas as acções associadas a cada caso de utilização e quais os perfis que estão associados a cada caso de utilização. Este componente verifica se a acção solicitada pelo utilizador se encontra registada e se é válida considerando o perfil do utilizador.

O componente *AuthenticationManager* está associado ao serviço de autenticação, explicado anteriormente, e disponibiliza as funcionalidades que permitem autenticar um utilizador, gerir a sessão do utilizador, obter dos dados da sessão de utilizador com base numa chave e armazenar dados na sessão do utilizador.

A *Plataforma de Acesso a Dados* é responsável por interpretar o conteúdo das mensagens enviadas pelo cliente e traduzir para *SQL*, bem como por traduzir o resultado de uma *query SQL* para o conteúdo de uma mensagem. Para isso, foram implementados os componentes *DataProvider*, *Dictionary* e *InterpreterContent*.

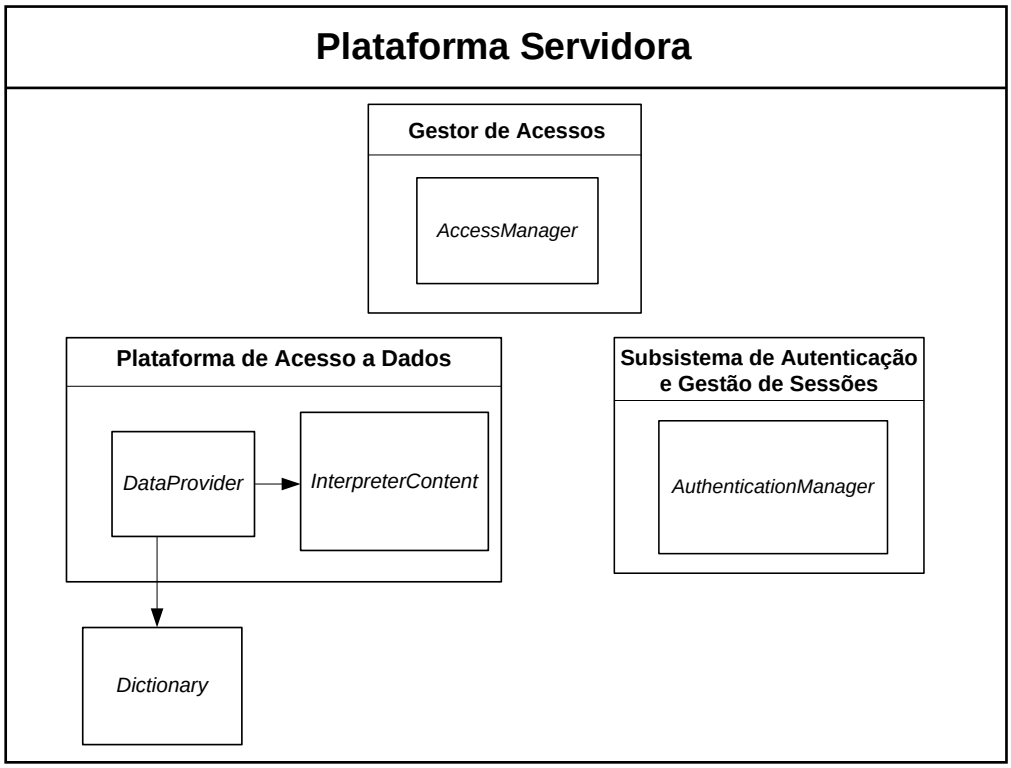


Figura 4.6 – Organização dos componentes da Plataforma Servidora

Na *Plataforma de Acesso a Dados*, o *DataProvider* é responsável por registar as acções possíveis sobre a plataforma (baseado por exemplo em casos de utilização) e as operações associadas, sejam elas de leitura, escrita, actualização e remoção, onde cada acção corresponde ao assunto da mensagem. Este componente permite associar para a mesma mensagem diferentes operações o que representa uma grande versatilidade uma vez que não limita o número de operações associadas a uma mensagem, tornando a plataforma mais flexível na sua utilização.

Para relacionar os atributos da mensagem, que estão associados a requisitos, com a especificação da base de dados foi necessário implementar o componente *Dictionary*.

O *Dictionary* permite registar a associação entre cada atributo da mensagem com o respectivo atributo da base de dados. Através desta associação é possível obter para cada atributo da base de dados o respectivo atributo na mensagem. Com este atributo e recorrendo ao componente *InterpreterContent* é possível obter o valor correspondente ao atributo na mensagem. Isto é conseguido devido à existência de vários métodos que independentemente do tipo do atributo (simples ou *array*) permitem obter o valor respectivo.

Para alem da criação dinâmica de *querys SQL* dotou-se a *Plataforma de Acesso a Dados* com a capacidade de executar procedimentos armazenados. Esta característica torna a plataforma versátil porque possibilita que possa ser utilizada por uma aplicação que assente a sua lógica de acesso a dados em procedimentos armazenados. Desta forma, e partindo do pressuposto que uma organização tem bem definidos os requisitos, é possível:

- obter as entidades de informação que darão origem às mensagens correspondentes e que serão interpretadas pela plataforma de automatização de apresentação, e
- obter os casos de utilização e registá-los com o(s) respectivo(s) procedimento(s) armazenado(s) na *Plataforma de Acesso a Dados*.

Após estas tarefas e realizando as configurações necessárias um novo *website* da organização com novo *layout* e novas funcionalidades foi realizado reaproveitando toda a lógica de acesso a dados, minimizando desta forma o tempo e os riscos de desenvolvimento.

Para além da capacidade da plataforma em executar procedimentos armazenados e para tornar a plataforma ainda mais flexível dotou-se a plataforma com a capacidade de executar *SQL* directamente. Ou seja, para uma determinada mensagem é possível associar uma *query SQL* ou blocos de *querys SQL* (*script*). O modo como é isto é conseguido é explicado na secção *5.2.2.1 Registo das acções*.

4.5. Segurança

Nesta secção serão apresentados aspectos relacionados com segurança que foram tidos em conta no desenvolvimento da plataforma, nomeadamente, o controlo de acesso à aplicação, a necessidade de ligações seguras, a definição de perfis de utilizador e a prevenção de ataques específicos sobre a aplicação.

4.5.1. *Controlo de acesso à aplicação*

Normalmente, é necessário controlar o acesso às aplicações, excepto para *websites* que são projectados para serem abertos ao público em geral. A plataforma foi pensada de forma a permitir estas duas situações. Quando as aplicações *Web* são seguras (isto é, quando as funcionalidades dentro da aplicação são destinadas a ser utilizadas apenas por utilizadores autorizados), os utilizadores devem ser autenticados antes de acederem a essas funcionalidades. Isso geralmente é realizado através de um formulário de autenticação que é exigido ao utilizador antes de aceder a áreas protegidas.

4.5.1.1. Ligações seguras para autenticação do utilizador

Para garantir a segurança da aplicação e privacidade dos utilizadores, as páginas através das quais o utilizador fornece informações pessoais devem ser apresentadas através de uma ligação segura. Isso normalmente significa usar *SSL* (*Secure Sockets Layer*) para a ligação segura a um servidor (geralmente identificadas pela utilização de *HTTPS* no *URL*). Por exemplo, quer a página de autenticação, quer as páginas onde o utilizador fornece informações sensíveis (por exemplo, o número do cartão do crédito, *password*, etc.), devem fazer uso de uma ligação segura para a transmissão desses dados.

As aplicações *Web* são responsáveis não só por autenticar os utilizadores, mas também por manter as informações de autenticação através dos pedidos. Manter essa

informação através de pedidos é normalmente realizado através do uso de sessões. Normalmente, as informações que pertencem a uma sessão (por exemplo, identificação do utilizador) são guardadas no servidor, e são acessíveis a uma aplicação através de uma chave de identificação de sessão que está associada a um utilizador específico. A chave é normalmente fornecida através de um *cookie*, cujo valor é um identificador de sessão único. Os *cookies* não devem conter nem o nome de utilizador nem a *password*.

Quando o utilizador está autenticado, a informação de autenticação pode ser adicionada à sessão. Se a sessão actual não existe para o navegador *Web* que a solicita, uma nova sessão é criada e o seu identificador é transmitido para o navegador *Web*.

A aplicação faz uso da sessão para manter as informações persistentes sobre a identidade do utilizador autenticado e sobre a duração da sessão do utilizador. A aplicação pode tomar decisões sobre quando invalidar a sessão, ou seja, a sessão pode ser considerada inválida após um determinado período de inactividade (tempo durante o qual o utilizador não tenha feito qualquer pedido), ou mesmo após um período fixo de tempo.

4.5.1.2. Definição de perfis de utilizador

Ao projectar uma aplicação, é importante lembrar que ela pode ser utilizada de diferentes formas de acordo com o perfil de utilizador. O utilizador pode ser um aluno, um funcionário, um docente, ou um administrador. Existem vários perfis que um utilizador pode ter quando acede a uma aplicação *Web*. Cada perfil diferente pode ser implementado através de listas separadas de controlo de acessos, ou seja, onde se especifica o que cada perfil tem autorização para fazer. Isto é importante para criar a aplicação porque permite disponibilizar as funcionalidades de acordo com o perfil do utilizador.

Antes do acesso a cada funcionalidade é necessário verificar as credenciais de autenticação do utilizador, de modo a garantir que o utilizador pode aceder a essa mesma funcionalidade. Esta verificação é necessária para evitar os casos em que um utilizador tenta através de um *URL* aceder a uma funcionalidade (página) à qual não está autorizado.

4.5.1.3. Boas práticas para o controlo de acesso à aplicação

De acordo com [1], existe um conjunto de orientações que devem ser seguidas para se controlar o acesso à aplicação, nomeadamente:

1 - Definir previamente os tipos de utilizadores (por exemplo, público em geral, autorizado, administrador), que poderão aceder à aplicação. A análise baseada em casos de utilização é um bom método para o desenvolvimento de cenários que descrevem como a aplicação será usada.

2 - Organizar a arquitectura da aplicação em áreas funcionais específicas. Identificar as classes de utilizadores que devem ter acesso a essas áreas. Garantir que o acesso a áreas funcionais restritas é limitada apenas aos utilizadores autorizados através da verificação da autorização em cada área funcional da aplicação.

3 - A autenticação deve ser realizada através de ligações seguras. Isso inclui não apenas o *link* activado quando um formulário de autenticação é apresentado, mas também a página com o formulário.

4 - As funções de administração para a aplicação devem ser consideradas no início do processo de desenvolvimento. Estas funções precisam ser protegidas contra utilizadores não autorizados. Três passos devem ser tomados para garantir as funções de administração na aplicação:

   a. hospedar essas funções num servidor *Web* separado que não é acessível fora da *firewall* da organização (caso exista);

   b. tornar o servidor acessível através de uma ligação segura *SSL*, mesmo dentro da *firewall*;

   c. permitir que só os utilizadores explicitamente autenticados possam aceder a estas funções.

4.5.2. *Injecção de SQL*

A injecção de *SQL* (*SQL injection*) é um método de ataque a aplicações *Web* através da inserção de instruções *SQL* no qual os atacantes são capazes de executar código *SQL* malicioso, explorando falhas na implementação da aplicação *Web*.

Um atacante usando injecção de *SQL* pode comprometer a segurança dos prestadores de serviços, atacando directamente uma aplicação *Web*. A injecção de *SQL* é um dos ataques de injecção de código mais utilizados [29].

As *querys SQL* que dependem de dados introduzidos pelo utilizador são passíveis de ataques de injecção de *SQL*.

Considerando a *query*:

SELECT * FROM alunos

WHERE nome = '$name' AND password = '$password'

A tabela alunos guarda toda a informação dos alunos. As variáveis de *input*, *$nome* e *$password*, possuem os valores introduzidos pelo utilizador. Estes valores são inseridos nos campos correspondentes da *query* anterior. Um atacante pode explorar isso colocando no campo da *password* o valor **'' OR '='**. A *query SQL* que será enviada para a base de dados é a seguinte:

SELECT * FROM alunos

WHERE nome = 'Alice' AND password = '' OR '='

No que diz respeito às prioridades dos operadores booleanos *AND* e *OR*, a expressão é equivalente à seguinte com parênteses:

SELECT * FROM alunos

***WHERE* (nome = 'Alice' AND password = '') OR '='**

Neste caso, a cláusula *WHERE* será sempre avaliada como verdadeira. Assim, o acesso seria concedido ao atacante.

Outra situação tem a ver com as *querys* que permitem pesquisas, como por exemplo:

SELECT * FROM alunos

WHERE aluno like '%$procurar%'

Isso pode ser mal utilizado por utilizadores mal-intencionados através da introdução de um critério de pesquisa que termina a instrução *SQL* e executa outra. Por exemplo, se a *query* for executada com o critério de pesquisa seguinte:

'; drop table alunos --

Inicialmente a tabela alunos é pesquisada sem nenhum nome indicado. Depois, a segunda instrução é executada provocando o apagamento da tabela alunos.

Os símbolos -- introduzem um comentário na *query SQL*, e neste caso são usados para ignorar a restante parte da *query*.

Para evitar estas situações foram tidos em conta alguns aspectos na implementação que se encontram descritos na secção *5.2.2.2 Injecção de SQL*.

5. Implementação e Configuração

Este capítulo descreve os aspectos de implementação que foram utilizados ao longo do desenvolvimento da plataforma. No final do capítulo, são apresentados os passos de configurações necessários para o funcionamento da plataforma.

A plataforma cliente foi desenvolvida em *Javascript* e a plataforma servidora foi desenvolvida em *PHP*.

Toda a construção da plataforma foi feita com base em objectos, encapsulando assim as funcionalidades. Esta abordagem permite um mecanismo homogéneo de comunicação com base em objectos e um maior encapsulamento das funcionalidades, aumentando a coesão do trabalho desenvolvido.

A comunicação entre a plataforma cliente e a plataforma servidora é realizada através de *JSON*.

Na plataforma cliente, quando se pretende enviar informação para a plataforma servidora é realizada a codificação de uma variável *Javascript* para o formato *JSON* e quando se pretende receber informação proveniente da plataforma servidora é realizada a descodificação do formato *JSON* em variável *Javascript*. Estas acções de codificação e descodificação são realizadas pelo objecto *JSON* obtido através da referência [30]. Foram efectuadas alterações neste objecto de modo a que suportasse a codificação/descodificação de alguns caracteres (por exemplo, à, á, ã, ç, í, ó, ú, entre outros).

Do lado do servidor, a codificação para *JSON* e a sua descodificação é realizada pelo objecto *Services_JSON* obtido através da referência [31].

5.1. Plataforma Cliente

A plataforma cliente foi desenvolvida em *Javascript* permitindo desta forma criar todos os elementos *HTML* dinamicamente, dar funcionalidades e manipular a estrutura e estilo com base na árvore *DOM* e estilos *CSS*.

5.1.1. *Plataforma de Automatização da Apresentação*

A *Plataforma de Automatização da Apresentação* é constituída por vários componentes que são explicados de seguida. A Figura 5.1 apresenta o detalhe dos vários componentes que fazem parte da plataforma de automatização de apresentação e como estes se relacionam entre si.

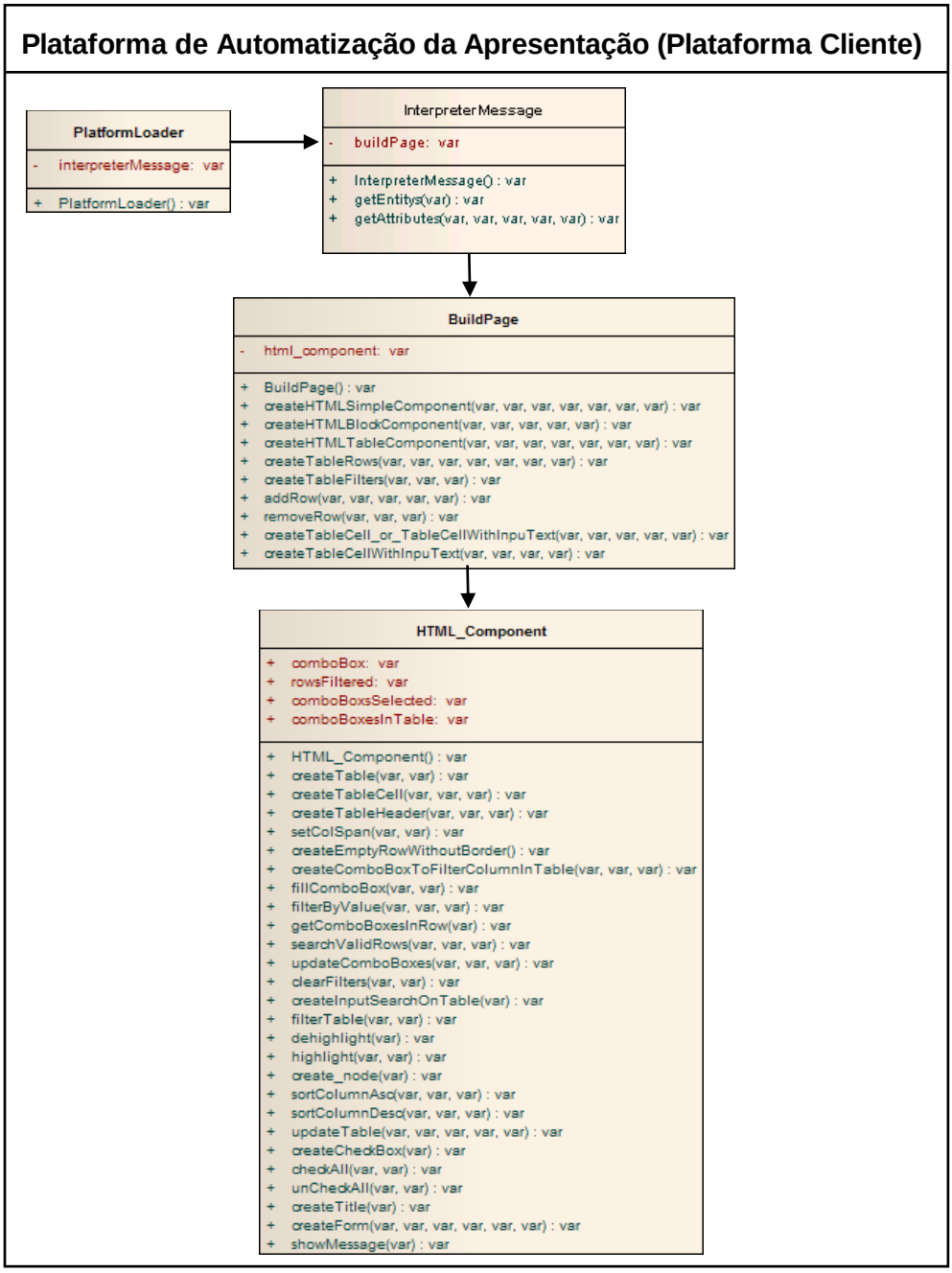


Figura 5.1 – Classes associadas aos componentes da *Plataforma de Automatização da Apresentação*

O componente *HTML_Component* que contêm métodos de criação de componentes base definidos a partir dos requisitos elaborados iterativamente durante o desenvolvimento. Alguns dos componentes que é possível criar são: tabelas, *combo boxes*, caixas de texto, entre outros.

Os componentes que foram necessários implementar foram criados paralelamente à plataforma de modo a facilitar e a minimizar o risco de desenvolvimento. Assim, e só após testes sobre o componente e as suas funcionalidades é que este foi integrado na plataforma.

Assim, o componente *HTML_Component* para além de ter sido utilizado na plataforma pode ser utilizado na criação de outras aplicações *Web* uma vez que é bastante genérico (Figura 5.1). Na secção seguinte são descritos os componentes e funcionalidades que foram implementadas.

Sendo uma plataforma orientada a mensagens que estão associadas a requisitos é através do componente *InterpreterMessage* que é efectuada a interpretação da mensagem e com base no formato definido são construídos os componentes *HTML*.

O componente *BuildPage* será o responsável por construir a estrutura da página tendo em conta a especificação associada a cada atributo da mensagem, sendo este que irá construir todos os componentes *HTML* necessários recorrendo ao componente *HTML_Component*. Caso seja pretendido disponibilizar novas estruturas para apresentar os conteúdos isso terá que ser implementado no componente *BuildPage*.

Para que a utilização da plataforma seja facilitada, foi criado o componente *PlatformLoader* que efectua o carregamento da plataforma através da inclusão de um único ficheiro *Javascript* na página. No carregamento da plataforma este ficheiro é interpretado e é feito o carregamento de todos os ficheiros necessários. Cada componente referido anteriormente corresponde a um ficheiro *Javascript*.

5.1.1.1. Componentes Gráficos e Funcionalidades Desenvolvidas

De seguida, são apresentados os componentes gráficos desenvolvidos e as respectivas funcionalidades que tornam as aplicações *Web* construídas pela plataforma mais dinâmicas, fáceis de usar e ricas em termos de utilização.

Os componentes gráficos, e respectivas funcionalidades, que foram desenvolvidos são:

- ***Combo Box***;
- **Tabela**:
    - Ordenável;
    - Dinâmica, ou seja, com a possibilidade de adicionar/remover linhas quando a aplicação se encontra no modo de edição;
    - Com pesquisa por conteúdo na tabela;
    - Com possibilidade de selecção múltipla de linhas;
    - Com pesquisa baseada em filtros (*combo boxes*) existentes no cabeçalho da tabela. Os filtros são preenchidos com os valores existentes na coluna da tabela onde pertence o filtro o que permite se ajustarem às selecções que forem realizadas. Esta funcionalidade torna a utilização dos filtros mais intuitiva uma vez que em cada filtro apenas são apresentadas ao utilizador as opções que fazem sentido. Uma vez que esta funcionalidade foi implementada de forma genérica pode ser utilizada por tabelas que possuam filtros de pesquisa nas suas colunas. Os métodos que permitem realizar esta lógica foram implementados no objecto *HTML_Component* e são os seguintes: *filterByValue*, *searchValidRows* e *updateComboBoxes* (Figura 5.1). O método *filterByValue* é o que é invocado quando uma *combo box* é seleccionada. O método *searchValidRows* é responsável por ocultar as linhas que não respeitam os filtros de pesquisa. O método *updateComboBoxes* é o responsável por actualizar cada *combo box* com os valores existentes na coluna a que pertence.
- **Componente Hierárquico** - este componente possibilita uma representação hierárquica de um conjunto de componentes.

Para além das funcionalidades associadas aos componentes foi implementado um mecanismo que permite traduzir os campos da mensagem para o idioma pretendido. Esta funcionalidade mostra-se principalmente relevante, por exemplo, na utilização de tabelas, em que é preciso preencher o cabeçalho de cada coluna com o texto associado a um determinado idioma.

Para isso, implementou-se um dicionário em que para cada assunto e para um determinado idioma permite registar a correspondência de cada atributo com o significado pretendido. Desta forma, apenas é necessário fornecer um ficheiro de

dicionário diferente para o idioma que se pretende e poder-se-á utilizar toda a restante base da aplicação.

5.1.2. *Subsistema de Coordenação*

Nesta secção será explicado como é realizada a gestão do fluxo de páginas e o controlo de acesso às funcionalidades das aplicações *Web*.

5.1.2.1. Gestão do fluxo de páginas

Para se realizar o fluxo entre as páginas é necessário definir um conjunto de informação, nomeadamente:

- O caso de utilização principal, que pode contemplar vários casos de utilização;
- O caso de utilização de origem;
- O caso de utilização de destino;
- O nome do evento que permite transitar do caso de utilização de origem para o caso de utilização de destino;
- Os parâmetros necessários para se transitar de um caso de utilização para outro.

Para a gestão do fluxo de páginas das aplicações *Web* adoptou-se uma abordagem que permitisse realizar essa gestão de forma centralizada, possibilitando desta forma um maior controlo e conhecimento do fluxo entre as várias páginas (estando cada página associada a um caso de utilização). Para facilitar esta tarefa, o fluxo entre as páginas é definido através de um ficheiro de configuração da seguinte forma:

```
"ConsultarPauta", "ConsultarPauta", "ev1", "ResumoFichaAluno",
[ "attr1" : "attr2" ]

"ConsultarCurso", "ConsultarCurso", "ev1", "ConsultarUC",
[ "attr1" : "attr2" ]
```

Exemplo 5.1 – Ficheiro de configuração para definir o fluxo de páginas

De seguida, é explicado o que é necessário para definir o ficheiro de configuração:

- 1ª posição - Caso de utilização principal, que pode contemplar vários casos de utilização;
- 2ª posição - Caso de utilização de origem;
- 3ª posição – Nome do evento que permite transitar do caso de utilização de origem para o caso de utilização de destino;
- 4ª posição - Caso de utilização de destino;
- 5ª posição - Lista de atributos que associa os atributos do caso de utilização de origem com os atributos do caso de utilização de destino. É através desta lista que é possível transitar de um caso de utilização para outro.

Para tornar a plataforma robusta implementou-se um mecanismo que evita ao utilizador definir as combinações possíveis de navegação entre as várias páginas da aplicação. Desta forma, o utilizador só tem que definir o caminho natural associado a um caso de utilização. Toda as outras hipóteses de navegação possíveis são realizadas automaticamente pela plataforma, permitindo depois construir a aplicação com um *menu* de navegação que permite ao utilizador navegar para as páginas directamente, sem o recurso à funcionalidade *back* do navegador *Web*.

É o componente *FlowManager* o responsável por registar os estados de transição e por criar os respectivos estados de transição (ou de navegação) possíveis, através dos métodos *addStateOfTransition* e *createReversePath*, respectivamente.

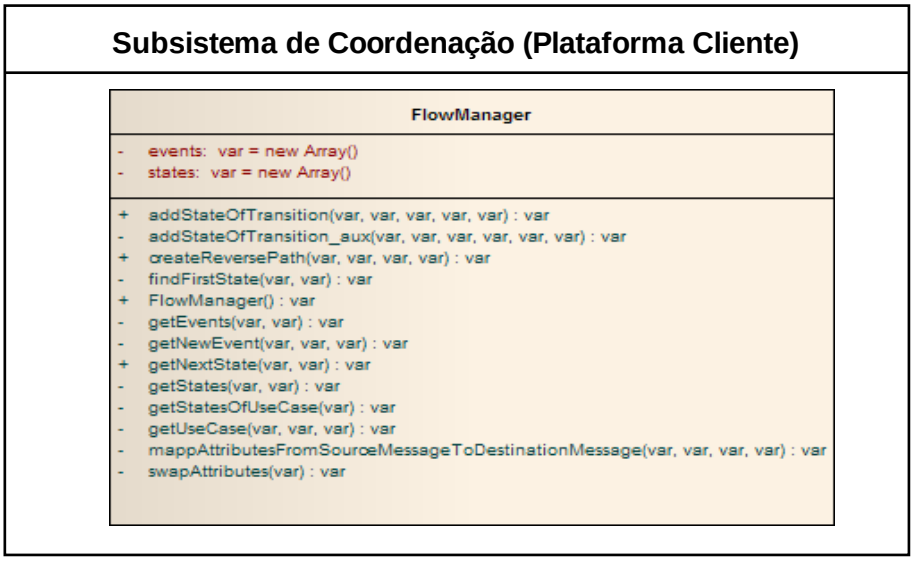


Figura 5.2 – Componente *FlowManager*

Como exemplo, o caso de utilização *"ConsultarAlunosPrescritos"* está associado a dois níveis de transição (*"ConsultarAlunosPrescritos"* → *"ResumoFichaAluno"* e *"ResumoFichaAluno"* → *"InscricoesAluno"*), os quais são definidos da seguinte forma:

```
"ConsultarAlunosPrescritos", "ConsultarAlunosPrescritos", "ev1",
"ResumoFichaAluno", ["Alunos.Numero" : "Numero"]

"ConsultarAlunosPrescritos", "ResumoFichaAluno", "ev2",
"InscricoesAluno", ["Numero" : "Numero"]
```

Exemplo 5.2 – Ficheiro de configuração para definir os estados de transição

Esta configuração dá origem aos estados de transição que se encontram na Tabela 5.1 (apresentada a seguir). Tendo em conta que o nome dos casos de utilização do Exemplo 5.2 são muitos extensos, foram obtidas abreviaturas de modo a serem colocadas na tabela seguinte, para que esta fique mais perceptível. De notar, que a plataforma não abrevia os nomes.

Neste exemplo são utilizadas as seguintes abreviaturas:

- ConsultarAlunosPrescritos = CAP
- ResumoFichaAluno = RFA
- InscricoesAluno = IA

Caso de Utilização	Caso de Utilização de origem	Evento	Caso de Utilização de destino	Atributos necessários para a transição de estado
CAP	CAP	ev1	RFA	["Alunos.Numero":"Numero"]
CAP	RFA	(1)	CAP	
CAP	RFA	ev2	IA	["Numero":"Numero"]
CAP	IA	(2)	RFA	["Numero":"Numero"]
CAP	IA	(3)	CAP	

Legenda:

As linhas apresentadas a azul correspondem à informação que se encontra no ficheiro de configuração que permite definir os estados de transição.

As linhas a branco correspondem aos estados de transição (caminhos inversos) gerados automaticamente pela plataforma.

Nome dos eventos gerados pela plataforma:

(1) "ResumoFichaAluno_ConsultarAlunosPrescritos_1"

(2) "InscricoesAluno_ ResumoFichaAluno_1"

(3) "InscricoesAluno_ ConsultarAlunosPrescritos _1"

Tabela 5.1 – Estados de transição associados ao caso de utilização *"ConsultarAlunosPrescritos"*

Esta tabela apresenta as várias hipóteses de navegação na aplicação *Web*, construída pela plataforma, tendo em consideração os casos de utilização apresentados.

Tendo em conta que existem transições de estados que necessitam de atributos foi necessário nos estados de transição gerados automaticamente pela plataforma efectuar a troca desses atributos, para que a relação entre os casos de utilização de origem/destino e os respectivos atributos fique correcta. Para isso, foi implementado no componente *FlowManager* o método *swapAttributes* responsável por realizar a troca de atributos.

Com a Tabela 5.1 são construídos os seguintes *menus* de navegação:

- página *"Inscrições Aluno"*:

```
Consultar Alunos Prescritos -> Resumo Ficha Aluno -> Inscricoes Aluno
```

- página *"Resumo Ficha Aluno"*:

```
Consultar Alunos Prescritos -> Resumo Ficha Aluno
```

5.1.2.2. Controlo de Acessos

O controlo de acessos é realizado recorrendo a um ficheiro de configuração que especifica as funcionalidades a disponibilizar a cada perfil de utilizador (Exemplo 5.3). Com este ficheiro é possível criar o *menu* principal e associar as funcionalidades a cada opção do *menu*.

O atributo *dynamicMenu* define para cada perfil as funcionalidades que se pretende disponibilizar.

O atributo *optionsDynamicMenu* define para cada perfil e para cada funcionalidade especificada no atributo *dynamicMenu* as respectivas sub-funcionalidades (sub-opções).

```
var menu = {
 "dynamicMenu" : {
   "Aluno"    : [
                    "Dados Pessoais",
                    "Informação Académica",
                    "Inscrições",
                    "Conta Corrente"
   ],
   "Docente" : [
                    "Consultar Pauta",
                    "Lançar Notas"
   ],
   "Colaborador" : [
                    "Consultar Cursos",
                    "Consultar UCs",
                    "Ficha Aluno",
                    "Inscrever",
                    "Listagens"
   ]
 },

 "optionsDynamicMenu" : {
   "Aluno" : {
      "Dados Pessoais" : [],
      "Informação Académica" : [
                        "Consultar percurso académico",
                        "Consultar plano de estudos",
                        "Consultar plano de curso"
      ],
```

```
      "Inscrições" : [
                        "Inscrição Online",
                        "Inscrição a Exames"
      ],
      "Conta Corrente" : [
                        "Situação Actual",
                        "Extracto de Conta",
                        "Dívidas",
                        "Referências Multibanco"
      ]
   },
   "Docente" : {
      "Consultar Pauta" : [],
      "Lançar Notas" : []
   },
   "Colaborador" : {
      "Consultar Cursos" : [],
      "Consultar UCs" : [],
      "Ficha Aluno" : [],
      "Inscrever" : [],
      "Listagens" : []
   }
 }
};
```

Exemplo 5.3 – Ficheiro de configuração que especifica as funcionalidades para cada perfil

A Figura 5.3 representa o *menu* resultante do exemplo anterior para o perfil Aluno.

Figura 5.3 – Exemplo da estrutura do *menu* para o perfil Aluno

5.2. Plataforma Servidora

Como foi apresentado anteriormente, a plataforma servidora é constituída por vários componentes, nomeadamente, um *Gestor de Acessos*, um *Subsistema de Autenticação* e por uma *Plataforma de Acesso a Dados*. Todos estes componentes foram desenvolvidos em *PHP*.

O acesso à base de dados foi realizado através da *API ODBC* existente nesta linguagem. Desta forma, possibilita-se que o mesmo programa possa utilizar simultaneamente o *MySQL*, o *Access*, o *SQL* Server, o *Oracle*, ou outro sistema de gestão de base de dados, sem a necessidade de mudanças na sua camada de dados. Uma vez que o objectivo do trabalho é realizar um Portal Académico baseado no que existe actualmente no ISEL e estando este assente sobre uma base de dados *Oracle* criou-se um *driver ODBC Oracle* sobre a base de dados de réplica do ISEL.

Não foi utilizada qualquer *API* de persistência de dados, apenas foi usada a *API ODBC* do *PHP* para acesso à base de dados do ISEL pois o foco do projecto não estava nessa parte, sendo todo o estado mantido em memória.

A Figura 5.4 apresenta as classes associadas aos componentes que fazem parte da plataforma servidora, nomeadamente, *DataProvider*, *Dictionary* e *InterpreterContent*.

De seguida, são explicados os detalhes de implementação de cada um destes componentes. O *Gestor de Acessos* (*AccessManager*) será explicado mais à frente e o *Subsistema de Autenticação* (constituído pelo componente *AuthenticationManager*) explicar-se-á na secção *5.2.4 Serviços desenvolvidos*.

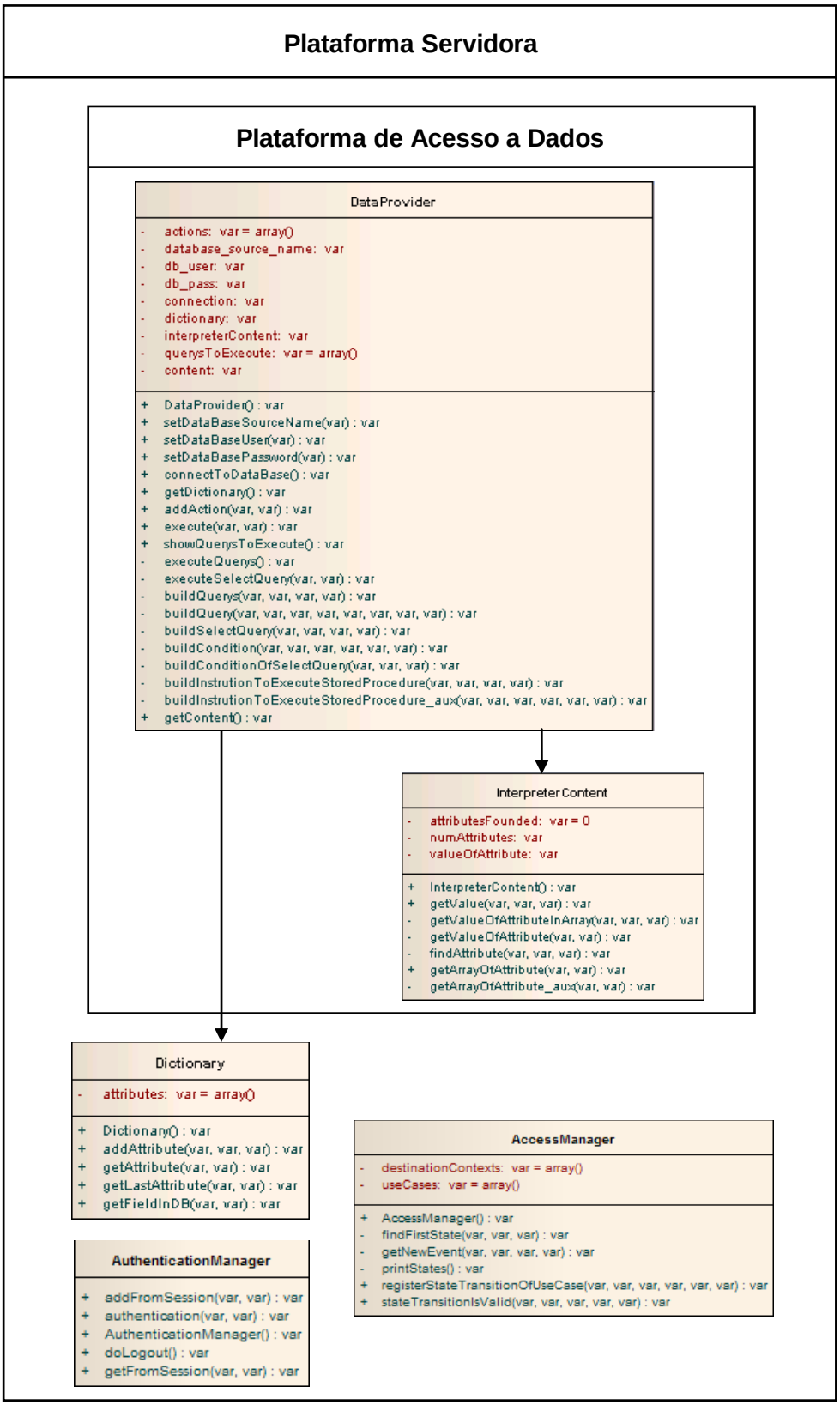


Figura 5.4 – Classes associadas aos componentes da Plataforma Servidora

5.2.1. *Dicionário*

O dicionário está relacionado com o modelo de informação (ver secção *4.1 Modelo de Informação*), o que permite associar os requisitos com a especificação da base de dados. Tendo em conta que os requisitos dão origem a entidades de informação, que por sua vez, dão origem a mensagens, é através do dicionário que é possível associar cada atributo da mensagem com o respectivo atributo da base de dados. O dicionário é representado pelo objecto *Dictionary*.

O dicionário encontra-se estruturado da seguinte forma:

- 1º posição - corresponde ao caso de utilização;
- 2ª posição – especificação do campo na base de dados;
- 3ª posição – atributo na mensagem.

```
EditarUC, CSE.T_TBDISCIP.CD_DISCIP, Codigo
EditarUC, CSE.T_TBDISCIP.DS_DISCIP, Design
EditarUC, CSE.NM_DOC_INT, Respons
EditarUC, CSE.OBJECTIVO.NUMORD, Objectivos.NumOrd
EditarUC, CSE.OBJECTIVO.TEXTO, Objectivos.Texto
EditarUC, CSE.RESULTAPREND.NUMORD, ResultAprend.NumOrd
EditarUC, CSE.RESULTAPREND.TEXTO, ResultAprend.Texto
EditarUC, CSE.RESULTAPREND.CD_DISCIP, Codigo
```

Exemplo 5.4 – Dicionário com campos da base de dados e o respectivo atributo na mensagem

O dicionário disponibiliza um método para obter o atributo da mensagem associado a um campo da base de dados e um método para obter o campo na base de dados associado a um atributo da mensagem.

O dicionário em conjunto com o objecto *InterpreterContent* permite obter o valor correspondente a qualquer atributo da mensagem.

5.2.2. *Plataforma de Acesso a Dados*

A *Plataforma de Acesso a Dados* é constituída pelos componentes *DataProvider* e *InterpreterContent*.

O componente *DataProvider* é um *service provider* que disponibiliza serviços de acesso a dados de carácter geral. Para além disso, também fornece serviços genéricos de acesso a dados, de acordo com a respectiva configuração, fazendo a adaptação entre o Modelo de Informação e o Modelo de Dados de forma automática. O componente *DataProvider* é o responsável por registar as acções possíveis sobre a plataforma (baseado por exemplo em casos de utilização) e as operações associadas, sejam elas de leitura, escrita, actualização e remoção, onde cada acção corresponde ao assunto da mensagem.

O componente *InterpreterContent* é o responsável por interpretar a mensagem e obter os valores associados a cada atributo da mensagem, para deste modo ser possível criar dinamicamente as *querys SQL*.

5.2.2.1. Registo das acções

A plataforma servidora permite registar acções de três formas possíveis, que serão explicadas de seguida. Estas acções estão associadas a casos de utilização. O objecto *DataProvider* é responsável pelo registo das acções.

(I) Registo para construção dinâmica de querys

Uma vez que se pretende construir *querys* dinamicamente foi definida a estrutura que se encontra no Exemplo 5.5 que especifica os elementos necessários para a construção da *query*. O atributo *attributeInMsg* é o que permite indicar se a *query* que se pretende construir está associada a um atributo na mensagem que representa um *array*. Se sim, significa que a *query* tem que ser criada o número de vezes correspondente à dimensão desse *array*, com a particularidade que cada *query* terá que ter valores correspondentes a cada elemento do *array*.

```
$querys = array(
          "update" => array (
                        array(
                           "table" => "CSE.T_TBDISCIP",
                           "columns" => array("DS_DISCIP"),
                           "condition" => array("CD_DISCIP"),
                           "attributeInMsg" => ""
                        ),
                        array(
                           "table" => "CSE.RESULTAPREND",
                           "columns" => array("NUMORD","TEXTO"),
                           "condition"=>array("CD_DISCIP","NUMORD"),
                           "attributeInMsg" => "ResultAprend"
                        )
                     )
          );
```

Exemplo 5.5 – Estrutura para a criação de *querys*

A estrutura é depois associada ao caso de utilização corresponde através do método *addAction* do objecto *DataProvider* (Figura 5.4). A estrutura permite definir múltiplas acções (leitura, escrita, actualização e remoção) e associá-las ao caso de utilização pretendido.

```
$dataProvider->addAction ( "EditarUC", $querys );
```

Exemplo 5.6 – Registo das acções associadas a um caso de utilização

(II) Registo directo de querys SQL ou de blocos de querys SQL

Outra forma possível de utilizar a plataforma é de executar *SQL* directamente. Ou seja, para uma determinada mensagem é possível associar uma *query SQL* ou blocos de *querys SQL* (*script*). Esta forma permite reutilizar as *querys* de uma organização, aproveitando toda a lógica de negócio e deste modo construir uma nova aplicação *Web* com um baixo custo de mudança e de forma rápida.

O Exemplo 5.7 apresenta uma *query* do Sistema de Gestão Académica (SGA) do ISEL que permite obter a informação pessoal de aluno. De salientar a existência do campo *putValueHere*. É este campo que permite associar o conteúdo correspondente da mensagem ao atributo da *query*. Para isso, no dicionário tem que existir a associação entre o atributo da *query* e o atributo da mensagem correspondente.

```
$querys = array(
    "select" => array(
                  array(
                    "query" =>
                                "SELECT cd_aluno, cd_curso, nome
                                 FROM   cse.t_alunos
                                 Where
                                   cd_aluno = putValueHere AND
                                   cd_curso = putValueHere
                                 ORDER BY cd_aluno",
                    "resultset"   => ""
                  )
                )
);
```

Exemplo 5.7 – Estrutura para registar *querys SQL* directamente

(III) Registo de procedimentos armazenados

Para além da capacidade de criar *querys* dinamicamente dotou-se a plataforma servidora com a capacidade de executar procedimentos armazenados. Para isso, foi definida a estrutura de dados que se encontra no Exemplo 5.8 que permite registar a informação necessária para se criar procedimentos armazenados.

De seguida, encontra-se a explicação dos atributos da estrutura de dados:

- O atributo *name* especifica o nome do procedimento armazenado.
- O atributo *params* especifica os parâmetros do procedimento armazenado. Se o atributo não tiver valor associado significa que o valor tem que ser obtido do conteúdo da mensagem. Para isso, é necessário que exista no dicionário a associação entre o atributo do procedimento armazenado e o atributo da mensagem. O componente *InterpreterContent* disponibiliza os métodos que permitem obter os valores pretendidos no conteúdo da mensagem (Figura 5.4).
- O atributo *attributeInMsg* permite indicar se o procedimento armazenado que se pretende construir está associado a um atributo na mensagem que representa um *array*. Se sim, significa que o procedimento armazenado tem que ser criado o número de vezes correspondente à dimensão desse *array*, com a particularidade que cada procedimento armazenado terá que ter valores correspondentes a cada elemento do *array*.

O Exemplo 5.8 corresponde ao procedimento armazenado que é utilizado pela aplicação de gestão académica do ISEL para realizar a inscrição de um aluno.

```
$storedProcedure = array(
    "storedProcedure" =>
          array (
                array(
                      "name" => "MANU_CSE.INSERIR_INSCRICAO",
                      "params" => array(
                                        "CD_LECTIVO" => "",
                                        "CD_DURACAO" => "",
                                        "CD_CURSO" => "",
                                        "CD_ALUNO" => "",
                                        "CD_DISCIP" => "",
                                        "CD_GRUPO" => "",
                                        "CD_OPCAO" => "",
                                        "CD_TURMA" => "",
                                        "CD_CUR_DIS" => "",
                                        "CD_PLA_DIS" => "",
                                        "CD_RAM_DIS" => "",
                                        "CD_PE_DIS" => 0,
                                        "IGNOR_NVALD" => 'N',
                                        "MELHORIA" => 'N',
                                        "INS_SEM_FREQ" => 'N',
                                        "FAZ_COMMIT" => 'S'
                                  ),
                      "attributeInMsg" => "UCs"
                )
          )
);
```

Exemplo 5.8 – Estrutura para registar procedimentos armazenados

5.2.2.2. Injecção de SQL

Para evitar estas situações foram tidos em conta os seguintes aspectos na implementação [29]:

- **Verificação dos parâmetros** - Uma possibilidade de impedir a injecção de *SQL* é verificar a sintaxe dos parâmetros de entrada, ou seja, verificar se eles estão no formato esperado pelo serviço.
- **Tratamento de excepções** – quando se implementa aplicações *Web* baseadas em bases de dados deve ser realizado um mecanismo de tratamento de excepções adequado. Erros de base de dados que são apresentados ao utilizador em vez de serem apanhados, não só transmitem a impressão de uma aplicação com baixa qualidade, como também fornecem informações úteis para os atacantes.

Ao nível da administração da base de dados pode ser aplicado o princípio do privilégio mínimo. Normalmente, as aplicações *Web* acedem ao sistema de base de dados através de uma conta da base de dados. Dependendo dos privilégios que são concedidos a essa conta, os ataques de injecção de *SQL* variam no que diz respeito aos danos que podem causar. Supondo que a funcionalidade de pesquisa é executada sob uma conta com privilégios de administração da base de dados. Assim, o ataque de apagar uma tabela (por exemplo, '*; drop table alunos*--') terá sucesso. O mesmo não iria acontecer se o princípio do privilégio mínino fosse seguido, ou seja, se fosse atribuído à conta da base de dados somente os privilégios necessários, ou seja, a operação de leitura (*SELECT*) sobre a tabela.

5.2.3. *Gestor de Acessos*

O componente *AccessManager* (*Gestor de Acessos*) é o responsável por registar os casos de utilização e quais os perfis que estão associados. Este componente verifica se a acção solicitada pelo utilizador se encontra registada e se é válida considerando o perfil do utilizador.

A definição dos casos de utilização e dos perfis de utilizador (actores) que lhe estão associados é realizada através de um ficheiro de configuração da seguinte forma:

```
"ConsultarAlunosPrescritos" : [ "Funcionario" ]
"ResumoFichaAluno" : [ "Funcionario", "Docente" ]
"InscricoesAluno" : [ "Funcionario" ]
"ConsultarCurso" : [ "Funcionario", "Docente", "Aluno" ]
```

Exemplo 5.9 – Ficheiro de configuração para definir os casos de utilização e os respectivos perfis

Para se verificar se acção solicitada pelo utilizador (contexto de destino) é válida tendo em conta o contexto de origem, foi implementado no componente *AccessManager* o método *stateTransitionIsValid* (ver Figura 5.4). Desta forma, garante-se que o acesso a cada funcionalidade é verificado, evitando-se assim acessos não autorizados.

5.2.4. *Serviços desenvolvidos*

Para além do serviço de acesso a dados, também foi implementado o serviço de autenticação *LDAP*. Este serviço foi implementado no *Subsistema de Autenticação*, que se encontra ilustrado na Figura 4.1 do capítulo *4 Arquitectura da Plataforma*.

Para a autenticação recorreu-se ao *Active Directory* disponibilizado pelo ISEL, assente no *Windows Server 2003*. O *Active Directory* é um sistema que gere todas as informações que permitem controlar o acesso dos utilizadores à rede e a aplicações. Nele ficam registados os nomes e senhas dos utilizadores, entre outras informações.

É no servidor de autenticação suportado pela ferramenta *Active Directory* que o serviço de autenticação realizado vai validar a informação de utilizador, não sendo aqui, no entanto, que é guardada a informação de sessão, sendo essa informação armazenada no servidor *Web*.

Para a utilização do *Active Directory* foram configurados três grupos "Docentes", "Funcionários" e "Alunos" (que representam os actores associados aos casos de utilização obtidos a partir dos requisitos) tendo sido inseridos alguns *logins* para teste em ambos.

A comunicação com o *Active Directory* foi realizada utilizando o protocolo de directório (Figura 5.5), o que levou a que fosse utilizada a *API LDAP* do *PHP* [32] para aceder aos dados pretendidos.

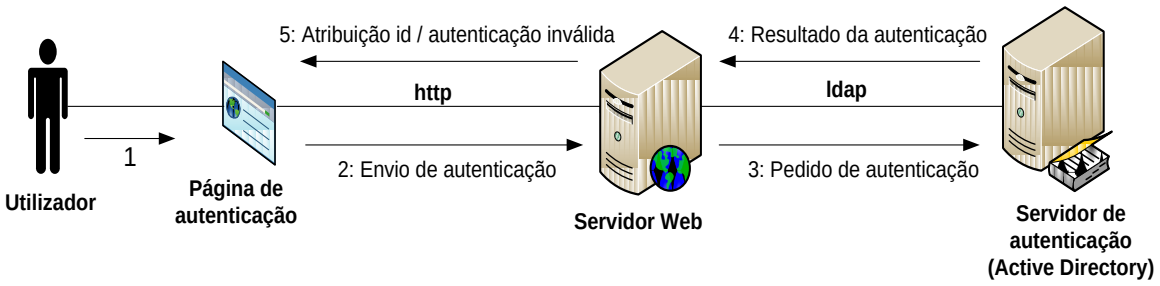


Figura 5.5 – Autenticação com recurso ao servidor *LDAP*

A Figura 5.5 representa a autenticação de um utilizador. No processo de autenticação é criado um contexto de autenticação, onde são indicados os dados do utilizador (nome e palavra chave), é realizado o pedido de autenticação e é atribuído um identificador de sessão ao utilizador autenticado. No caso da autenticação ser válida, para além do identificador de sessão é guardada a informação do perfil do utilizador em sessão.

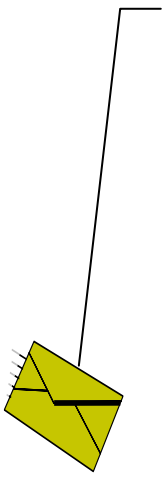
5.3. Configuração da Plataforma

Nesta secção são descritos os passos necessários para configurar a plataforma.

Tendo com conta que o foco da solução é baseado nos requisitos, é necessário inicialmente definir esses mesmos requisitos, especificar as entidades de informação e os casos de utilização, através da definição das acções a realizar pelo utilizador.

De seguida, são descritos os passos necessários para configurar a plataforma:

1. Após a definição dos requisitos e das respectivas entidades de informação, criar as respectivas mensagens respeitando o formato definido na secção *4.2 Mensagens Aplicacionais*. A Figura 5.6 corresponde à mensagem que está associada à funcionalidade *"Consultar Pauta"*. Cada mensagem corresponde a um ficheiro *Javascript*.



Acl_Pauta.js

```
"subject" : "Consultar Pauta",
....
"outer" :  {
        "Pauta" : {
              "UnidadeCurricular" : [ {"type" : "StrEditRequired"} ],
              "AnoLectivo" : [ {"type" : "AnoLectivo"} ],
              "PeriodoLectivo" : [ {"type" : "PeriodoLectivo"} ],
              "ResponsavelUnidadeCurricular" : [ {"type" : "StrEditRequired"} ],
              "Turma" : [ {"type" : "StrEditRequired"} ],
              "Alunos" : [ {"set" : "Aluno"} ]
        }
},
"inner" : {
        "Aluno" : {
              "Numero" : [ … ],
              "Nome" : [ …. ],
              "Classificacao" : [ … ],
              "TipoAvaliacao" : [ … ],
              "SitAvaliacao" : [ … ],
              "Data" : [ … ]
        }
},
"content" : {
     "UnidadeCurricular" : "Engenharia de Software",
     "AnoLectivo" : "2009-2010",
     "PeriodoLectivo" : "2º Semestre",
     "ResponsavelUnidadeCurricular" : "José Alberto",
     "Turma" : "MI12D",
     "Alunos" : [ …. ]
 },
….
```

Figura 5.6 – Parte da mensagem *ACL* associada ao assunto *"ConsultarPauta"*

2. Registo em ficheiro de configuração os endereços responsáveis (*endpoints*) por tratar de cada pedido (assunto). Isto permite que os serviços possam ser executados em servidores diferentes, possibilitando desta forma distribuir os pedidos consoante o assunto.

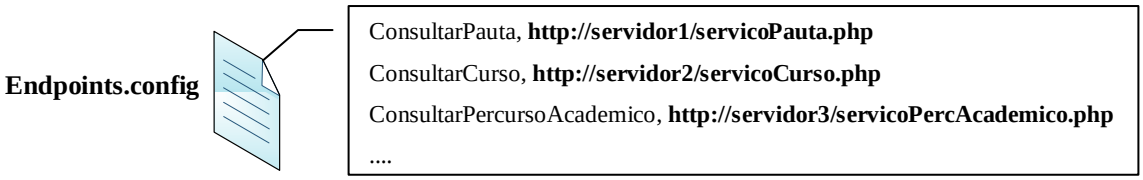


Figura 5.7 – Ficheiro de configuração dos *endpoints* responsáveis por tratar de cada pedido

3. Registo em ficheiro de configuração existente no servidor dos perfis de utilizador (actores) que podem aceder a cada funcionalidade. Na secção *5.1.2.2 Gestor de Acessos* é explicado o conteúdo deste ficheiro.

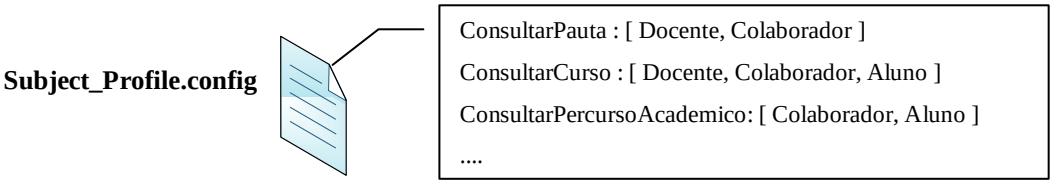


Figura 5.8 – Ficheiro de configuração dos perfis de utilizador associados a cada funcionalidade

4. Registo no dicionário existente no servidor da associação dos atributos do conteúdo da mensagem com a especificação da base de dados, ou seja, é através deste dicionário que é possível determinar qual o campo da base de dados que está associado a cada atributo da mensagem. Na secção *5.2.1 Dicionário* é explicado o conteúdo deste ficheiro.

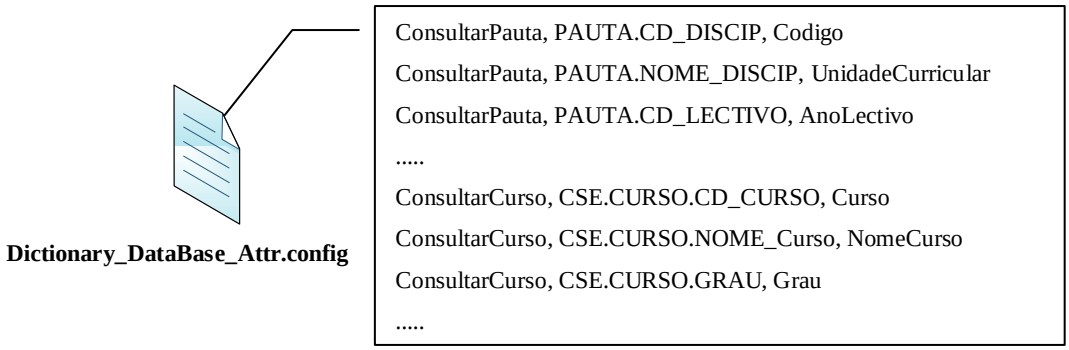


Figura 5.9 – Dicionário de atributos e respectiva especificação na base de dados

5. Registo das acções a realizar sobre a base de dados associadas a cada funcionalidade num ficheiro existente no servidor. Na secção *5.2.2.1 Registo das acções* são apresentadas as várias formas possíveis de registar as acções na plataforma. A figura seguinte apresenta o ficheiro que permite registar as acções associadas a cada funcionalidade.

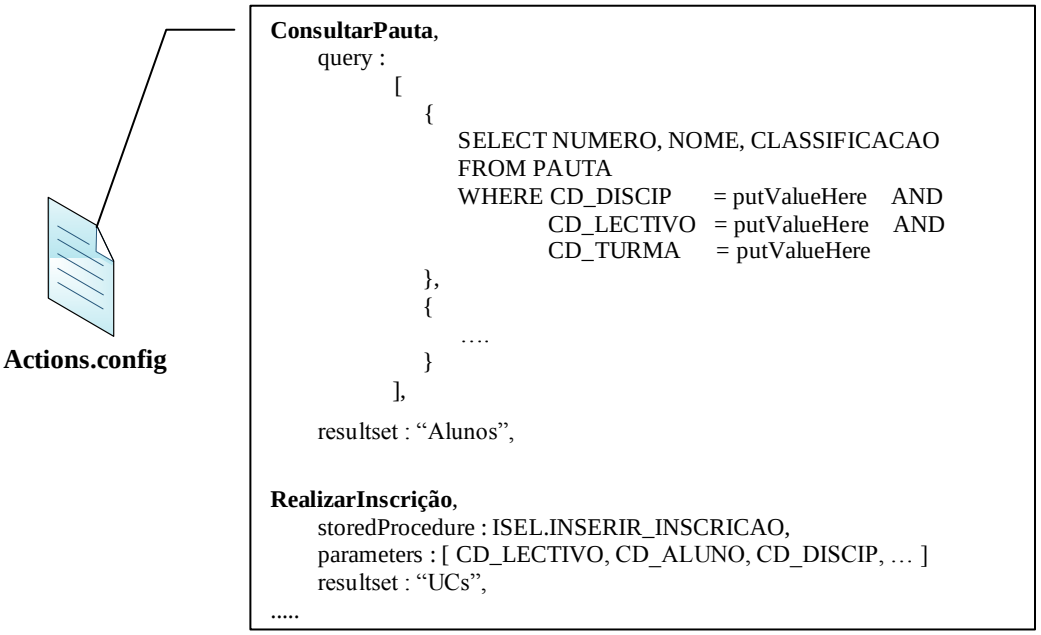


Figura 5.10 – Ficheiro com as acções a associadas a cada funcionalidade

6. Registo num ficheiro existente no servidor das transições de estado associadas a cada funcionalidade e os parâmetros necessários para se passar de um estado (página) para outro(a). A figura seguinte apresenta um exemplo desse ficheiro. Na secção *5.1.2.1 Gestão do fluxo de páginas* é explicado o conteúdo deste ficheiro.

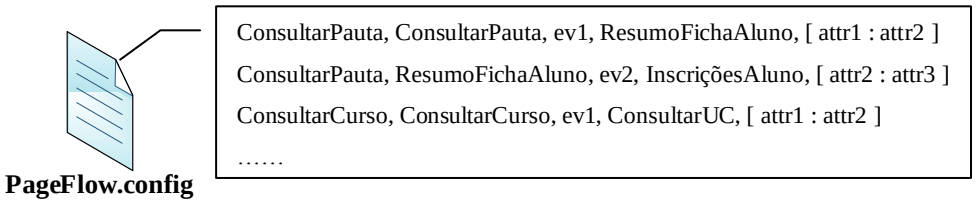
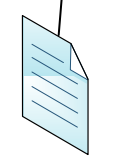


Figura 5.11 – Ficheiro com o fluxo de páginas

7. Especificação das opções a disponibilizar a cada perfil de utilizador (ficheiro *Javascript*). Com base nesta especificação serão construídos dinamicamente os *menus* e as respectivas opções consoante o perfil do utilizador. Na secção *5.1.2.2 Controlo de Acessos* é explicado como isso é conseguido.



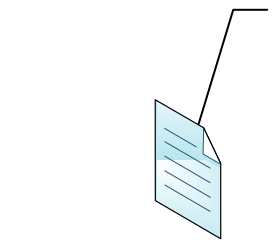

```
var menu = {
   "dynamicMenu" : {
         "Aluno"   : [ "Dados Pessoais", "Informação Académica", "Consultar Curso" ],
         "Docente" : [ "Consultar Pauta", "Consultar Curso" ],
         "Colaborador" : [ "Consultar Pauta", "Consultar Curso", "Criar Curso" ]
   },

   "optionsDynamicMenu" : {
         "Aluno" : {
               "Dados Pessoais" : [],
               "Informação Académica" : [
                                          "Consultar percurso académico",
                                          "Consultar plano de estudos",
                                          "Consultar plano de curso"
               ],
               "Consultar Curso" : []
         },
         "Docente" : {
               "Consultar Pauta" : [],
               "Consultar Curso" : []
         },
         "Colaborador" : {
               "Consultar Pauta" : [],
               "Consultar Curso" : [],
               "Criar Curso" : []
         }
   }
};
```

Figura 5.12 – Ficheiro para criação dinâmica do *menu* consoante o perfil do utilizador

8. Definição do ficheiro de dicionário responsável por associar os atributos da mensagem com os nomes que se pretende apresentar na página. Se no dicionário não for indicado a associação com um atributo da mensagem será apresentado o nome do próprio atributo.

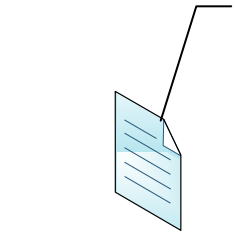
A plataforma suporta vários idiomas, para isso é necessário apenas fornecer um ficheiro de dicionário diferente para o idioma que se pretende e poder-se-á utilizar toda a restante base da aplicação. A Figura 5.13 apresenta o dicionário para a língua portuguesa e a Figura 5.14 apresenta o dicionário para a língua inglesa.



Portuguese_Dictionary.js

```
"ConsultarPauta" : {
      "UnidadeCurricular" : "Unidade Curricular",
      "ResponsavelUnidadeCurricular" : "Responsável da Unidade Curricular",
      ……
},
"ConsultarCurso" : {
       "Design" : "Designação",
       "ECTS" : "Nº de ECTS Total",
        ….
},
…..
```

Figura 5.13 – Dicionário para tradução de termos para a língua portuguesa



English_Dictionary.js

```
"ConsultarPauta" : {
      "ConsultarPauta" : "Consult list of students",
      "UnidadeCurricular" : "Discipline",
      "ResponsavelUnidadeCurricular" : "Coordinator of Discipline",
      ……
},
"ConsultarCurso" : {
       "ConsultarCurso" : "Consult Course",
       "Design" : "Designation",
       "ECTS" : "Total Number of ECTS",
        ….
},
…..
```

Figura 5.14 – Dicionário para tradução de termos para a língua inglesa

6. Protótipo (Portal Académico)

Como prova de conceito foi desenvolvido um Portal Académico que servirá para demonstrar o funcionamento da plataforma e dos conceitos aplicados. Este portal é constituído por um conjunto de funcionalidades tendo em conta o perfil do utilizador. Como foi referido ao longo do trabalho, as funcionalidades estão associadas a casos de utilização que expressam os requisitos funcionais da aplicação.

6.1. Fase de Preparação

Nesta fase foi efectuado o levantamento dos requisitos e dos casos de utilização que estão associados a um Sistema de Gestão Académica (SGA), de modo a contemplar algumas dessas funcionalidades no Portal Académico. A Figura 6.1 apresenta alguns dos casos de utilização que foram considerados para o protótipo e que serão explicados de seguida.

		Actor		
Prioridade	Caso de utilização	Aluno	Docente	Operador
1	Aceder à página de entrada			
1	Consultar percurso académico			
1	Criar Curso			
1	Inscrever a Unidades Curriculares			
1	Consultar pauta			
1	Lançar notas			

Figura 6.1 – Casos de utilização

Na figura apresentada, para além dos casos de utilização é indicada a respectiva prioridade de implementação e a associação dos casos de utilização a perfis de utilizadores (actores).

Para a implementação do protótipo do Portal Académico foi realizado o diagrama de contexto do Sistema de Gestão Académica, o diagrama de casos de utilização, a especificação dos casos de utilização, os diagramas de sequências e uma máquina de estados. Estes itens são explicados nas secções seguintes.

Os requisitos serviram para obter a mensagem que dá origem à página *Web* e os diagramas de sequência em conjunto com os casos de utilização serviram para ilustrar as interacções entre os objectos do Sistema de Gestão Académica o que permitiu obter as transições entre as várias páginas (fluxo de páginas). Com estas transições, obtém-se o ficheiro com as transições de estado associadas a cada funcionalidade (fluxo de páginas), apresentado na Figura 5.11 do capítulo anterior.

6.1.1. *Estudo das aplicações actuais de gestão académica do ISEL*

Antes da implementação do Portal Académico efectuou-se um estudo da aplicação de *back office* de gestão académica do ISEL (CSE) e da aplicação de *front office* (Portal Académico). Estas apresentam algumas limitações e alguns problemas de funcionamento que se relatam a seguir.

Aplicação de *back office* (CSE):

- Listagens incorrectas face ao pretendido. Por exemplo, perante um filtro onde se especifica que os valores a obter tem que ser menores ou iguais a 59 obtêm-se valores superiores a 59, como se observa na figura seguinte.

Total de Disciplinas Matriculadas diferente de 0 / Total de ECTS aprovados menor ou igual que 59 / Cód. curso: 9126 - Engenharia Química e B

Numero	Nome	Ano Lect. Ingresso	Dt. Reing.	Total ECTS Aprovados
100000	Aluno A	2007081		59
100001	Aluno B	2007081		62
100002	Aluno C	2007081		79
100003	Aluno D	2007081		73
100004	Aluno E	2007081		90
100005	Aluno F	2007081		59
100006	Aluno G	2007081		0
100007	Aluno H	2007081		35,5
100008	Aluno I	2008091		35
100009	Aluno J	2007081		47,5
100010	Aluno L	2007081		36,5
100011	Aluno M	2007081		84,5

Figura 6.2 – Listagem obtida da aplicação de *back office* do ISEL (CSE)

- Funcionalidades utilizadas frequentemente exigem muitas interacções do utilizador, por exemplo, a funcionalidade que permite ao colaborador dos serviços académicos inscrever um aluno exige pelo menos sete interacções.

Aplicação de *front office* (Porta Académico):

- Filtros de pesquisa com mau funcionamento. Na figura seguinte verifica-se que os filtros permitem ao utilizador seleccionar opções que não irão retornar qualquer informação. Desta forma, a aplicação não disponibiliza ao utilizador as opções que fazem sentido tendo em conta cada contexto, tornando a sua utilização confusa.

Figura 6.3 – Aplicação de *front office* do ISEL (Portal Académico)

Um problema que é transversal às duas aplicações é o de não permitir listar todo o percurso do aluno na instituição caso ele tenha frequentado vários cursos.

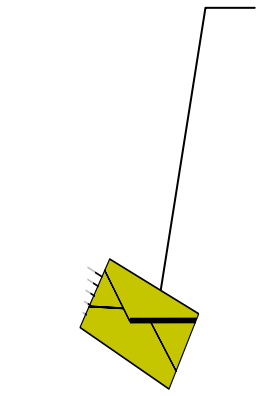
Na aplicação actual de *front office* (Portal Académico) do ISEL para o aluno puder visualizar o percurso académico dos cursos que frequentou tem que aceder ao sistema com *logins* diferenciados para cada curso que frequentou. Isto torna confusa a utilização deste portal por parte dos alunos uma vez não permite uma visualização integrada, obrigando ao conhecimento dos *logins* de acesso para cada curso.

Na aplicação de *back office* de gestão académica do ISEL esta limitação levanta muitos problemas na realização de algumas tarefas, nomeadamente, na criação de certidões para os alunos, no processo de reconhecimento de competências, entre outras.

Devido a estas limitações, surgiu a ideia de implementar esta funcionalidade para permitir uma visão integrada de todo o percurso do aluno. Para isso, foi analisado o modelo de dados associado do ISEL e foi criado o serviço que retorna todo o percurso do aluno. Este serviço está associado ao caso de utilização *"Consultar Percurso Académico"*, explicado na secção *6.1.4 Especificação de Casos* de Utilização.

Outra situação que foi detectada tem a ver com a inexistência de uma funcionalidade que permite definir o plano de estudos de um aluno por parte de um docente, sendo este um processo que é totalmente realizado no ISEL com base em papel.

Tendo em conta isso, foi criado o caso de utilização *"Criar Plano de Estudos"*, explicado na secção *6.1.4 Especificação de Casos de Utilização*. Através da análise de requisitos obtêm-se a seguinte mensagem:

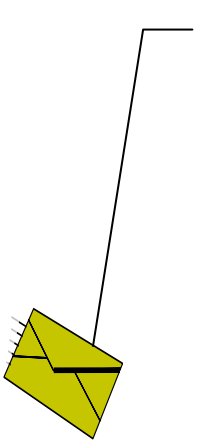


Acl_PlanoEstudos.js

```
{
  ….
  "subject" : "Criar Plano de Estudos",
  ….
  "content" : {
      "NumAluno" : 36000,
      "Nome" : "Ana",
      "Curso" : "Mestrado em Engenharia Informática",
      "UCsRealizadas" : [
            {
                "Design" : "Análise Matemática II",
                "Sigla" : "AM II",
                "ECTS" : 6.0,
                "Instituicao" : "ISEC"
            },
            ….
      ],
      "UCsARealizar" : [
            {
                "Design" : "Engenharia de Software",
                "Sigla" : "ES",
                "ECTS" : 6.0
            },
            ….
      ]
  },
  "tablesWithMultipleChoice" : [ "UCsARealizar" ]
  ….
};
```

Figura 6.4 – Parte da mensagem *ACL* associada ao assunto *"Criar Plano de Estudos"*

Devido ao facto do ISEL não disponibilizar aos docentes uma aplicação que permita efectuar o lançamento de notas *online* foi criado o caso de utilização *"Lançar Notas"*, explicado na secção *6.1.4 Especificação de Casos* de Utilização. Através da análise de requisitos obtêm-se a seguinte mensagem:



Acl_LancarNotas.js

```
{
  ….
  "subject" : "Lançar Notas"
  ….
  "content" : {
       "AnoLectivo" : "2009-2010",
       "PeriodoLectivo" : "2º Semestre",
       "UnidadeCurricular" : "Fundamentos de Inteligência Artificial",
       "ResponsavelUnidadeCurricular" : "José Eduardo",
       "Turma" : "LR41D",
       "Alunos" : [
              {
                   "Numero" : 25906,
                   "Nome" : "Ricardo Augusto",
                   "Classificacao" : 12,
                   "TipoAvaliacao" : "Época Normal",
                   "SitAvaliacao" : "Aprovado",
                   "Data" : "Sem Data"
              },
              {
                   ……
              }
       ],
  },
  …
};
```

Figura 6.5 – Parte da mensagem *ACL* associada ao assunto *"Lançar Notas"*

A Figura 6.15, que se encontra na secção *6.2 Implementação*, apresenta a página resultante da mensagem associada ao caso de utilização *"Lançar Notas"*.

6.1.2. *Diagrama de contexto*

O diagrama de contexto representa todo o sistema como um único processo e é composto por fluxos de dados que mostram as relações entre o sistema e as entidades (actores). O Actor é algo que interage com o sistema, mas sobre o qual não se tem controlo e está fora da influência do sistema. Os actores têm um papel externo e são quem inicia os casos de utilização.

O diagrama de contexto é a representação geral do sistema, destacando [26]:

• Entidades – agentes externos ao sistema que interagem com ele, gerando estímulos e recebendo respostas.

• Fluxos – identificação dos tipos de estímulos e respostas entre as entidades do sistema.

A Figura 6.6 ilustra o diagrama de contexto do Sistema de Gestão de Académica que se propôs realizar.

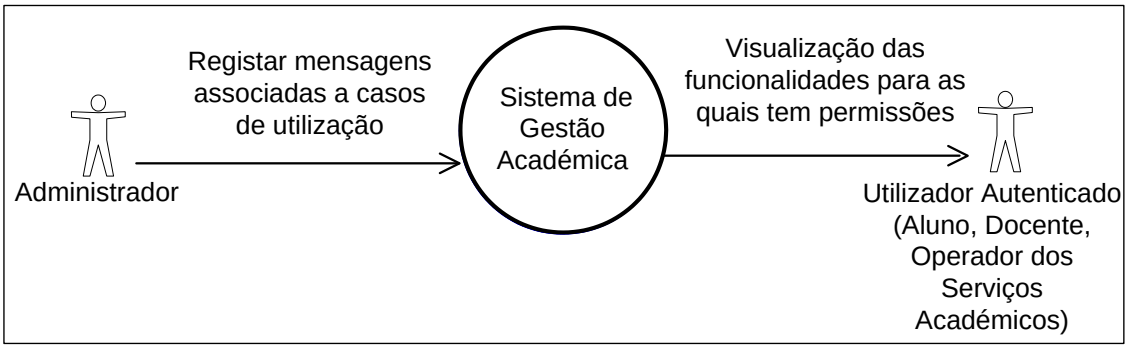


Figura 6.6 – Diagrama de contexto

6.1.3. *Diagrama de Casos de Utilização*

O Diagrama de Casos de Utilização especifica o comportamento e os aspectos envolventes do sistema em termos de casos de utilização e de actores. Este diagrama fornece um modo de descrever a visão externa do sistema e suas interacções com o mundo exterior, representando uma visão de alto nível da funcionalidade do sistema.

Tendo em conta que se pretende criar um sistema de gestão académica (Portal Académico) foram identificados os actores e algumas das acções que efectuam no sistema e procedeu-se à exposição dos casos de utilização recorrendo ao Diagrama de Casos de Utilização. De seguida, são apresentados os Diagramas de Casos de Utilização

para cada um dos actores identificados, nomeadamente, aluno (Figura 6.7), operador dos serviços académicos (Figura 6.8) e docente (Figura 6.9).

O Diagrama de Casos de Utilização para o perfil aluno apresenta as acções que este utilizador pode realizar no Sistema de Gestão Académica.

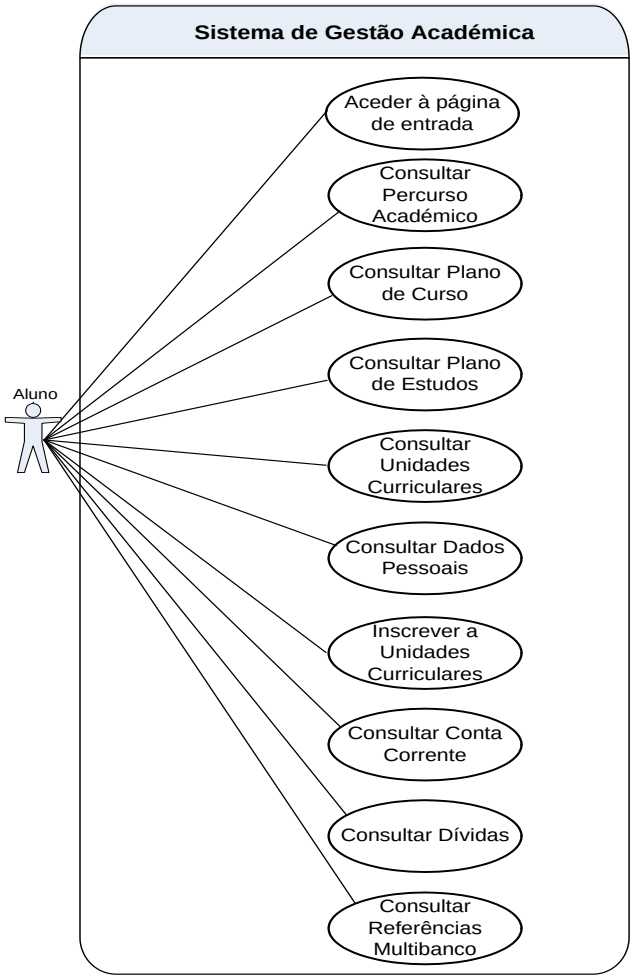


Figura 6.7 – Diagrama de Casos de Utilização para o perfil Aluno

O Diagrama de Casos de Utilização para o perfil operador dos serviços académicos apresenta as acções que este utilizador pode realizar no Sistema de Gestão Académica.

Figura 6.8 – Diagrama de Casos de Utilização para o perfil Operador (Serviços Académicos)

O Diagrama de Casos de Utilização para o perfil docente apresenta as acções que este utilizador pode realizar no Sistema de Gestão Académica.

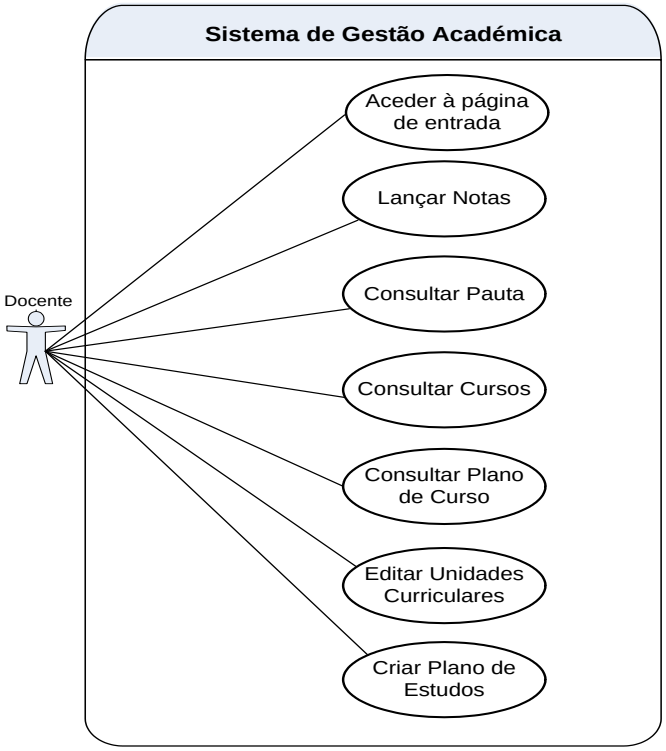


Figura 6.9 – Diagrama de Casos de Utilização para o perfil Docente

6.1.4. *Especificação de Casos de Utilização*

Um caso de utilização descreve o que faz um sistema (ou parte deste), mas não como é que tal é realizado. O foco é portanto na visão externa do sistema, ou seja, na visão que os utilizadores têm dele.

A especificação de casos de utilização detalha a operação dos casos de utilização anteriormente identificados.

De seguida, são especificados alguns dos casos de utilização envolvidos no modo de funcionamento do Sistema de Gestão Académica, nomeadamente, *"Aceder à página de entrada"* (Exemplo 6.1), *"Consultar Percurso Académico"* (Exemplo 6.2), *"Criar Curso"* (Exemplo 6.3), *"Inscrever a Unidades Curriculares"* (Exemplo 6.4), *"Consultar Pauta"* (Exemplo 6.5), *"Lançar Notas"* (Exemplo 6.6) e *"Criar Plano de Estudos"* (Exemplo 6.7).

De seguida, são indicadas as operações associadas ao caso de utilização *"Aceder à página de entrada"*.

Nome: Aceder à página de entrada **Descrição:** Permite a um utilizador aceder à página de entrada do Sistema de Gestão Académica. **Actores:** Aluno, Colaborador dos Serviços Académicos e Docente
Cenário principal: 1. O utilizador acede à página de *login* do Sistema de Gestão Académica. 2. O utilizador introduz o nome e a palavra passe. 3. O sistema verifica os dados. Dados válidos. Acesso à página de entrada do Sistema de Gestão Académica associada ao perfil do actor e o caso de utilização termina. **Cenário alternativo 1:** 1. No passo 3 os dados são inválidos. 2. O sistema mostra uma mensagem a informar o utilizador desse facto e o caso de utilização termina.

Exemplo 6.1 – Caso de utilização *"Aceder à página de entrada"*

No exemplo apresentado a seguir, são indicadas as operações associadas ao caso de utilização *"Consultar Percurso Académico"*.

Nome: Consultar Percurso Académico

Descrição: Permite a um utilizador consultar o percurso académico de um aluno.

Actores: Aluno e Colaborador dos Serviços Académicos

Pré condições: O utilizador tem de estar autenticado perante o sistema.

Cenário principal:

1. O utilizador encontra-se na página de entrada do Sistema de Gestão Académica.
2. O utilizador selecciona a opção "Consultar Percurso Académico".
3. Se o utilizador é um aluno:
   3.1. O sistema mostra o percurso académico do aluno e o caso de utilização termina.
4. Se o utilizador é um colaborador dos Serviços Académicos:
   4.1. Incluir o caso de utilização "Procurar Aluno".
   4.2. Se aluno existe é apresentado o percurso académico do aluno e o caso de utilização termina.
   4.3. Se o aluno não existe o sistema mostra uma mensagem a informar o utilizador desse facto e volta ao passo 4.1.

Exemplo 6.2 – Caso de utilização *"Consultar Percurso Académico"*

As operações associadas ao caso de utilização *"Criar Curso"* encontram-se especificadas no exemplo seguinte.

Nome: Criar Curso

Descrição: Neste caso de utilização é permitido criar um curso.

Actores: Colaborador dos Serviços Académicos

Pré condições: O utilizador tem de estar autenticado perante o sistema.

Cenário principal:

1. O utilizador encontra-se na página de entrada do Sistema de Gestão Académica.
2. Incluir o caso de utilização "Consultar Cursos".
3. Seleccionar a opção "Criar Curso".
4. Inserir os dados do curso.
5. O utilizador prime o botão "Guardar".
6. Se os dados são válidos, estes são guardados e o caso utilização termina.

Cenário alternativo 1

1. Idem até ao passo 5 do cenário principal.
2. O sistema mostra uma mensagem a indicar falta de preenchimento de campos obrigatórios
3. O caso de utilização termina.

Cenário alternativo 2

1. Idem até ao passo 5 do cenário principal.
2. O sistema mostra uma mensagem a indicar que o código do curso já existe.
3. O caso de utilização termina.

Exemplo 6.3 – Caso de utilização *"Criar Curso"*

De seguida, são indicadas as operações associadas ao caso de utilização *"Inscrever a Unidades Curriculares"*.

Nome: Inscrever a Unidades Curriculares

Descrição: Neste caso de utilização é permitido inscrever o aluno às unidades curriculares pretendidas.

Actores: Aluno e Colaborador dos Serviços Académicos

Pré condições: O utilizador tem de estar autenticado perante o sistema.

Cenário principal:

1. O utilizador encontra-se na página de entrada do Sistema de Gestão Académica.
2. Incluir o caso de utilização "Consultar Unidades Curriculares para Inscrição".
   2.1. Se o utilizador é um aluno passar para o passo 3.
   2.2. Se o utilizador é um colaborador dos Serviços Académicos incluir caso de utilização "Procurar Aluno" e passar para o passo 3.
3. Seleccionar as unidades curriculares pretendidas.
4. Seleccionar as turmas pretendidas para cada unidade curricular.
5. O utilizador prime o botão "Inscrever".
6. Se existem vagas nas turmas seleccionadas a inscrição é guardada e o caso de utilização termina.

Cenário alternativo 1

1. Idem até ao passo 5 do cenário principal.
2. O sistema mostra uma mensagem a indicar que existem turmas seleccionadas sem vagas disponíveis.
3. O caso de utilização termina.

Exemplo 6.4 – Caso de utilização *"Inscrever a Unidades Curriculares"*

No exemplo apresentado a seguir, são indicadas as operações associadas ao caso de utilização *"Consultar Pauta"*.

Nome: Consultar Pauta

Descrição: Consultar os alunos inscritos às unidades curriculares leccionadas pelo docente.

Actores: Colaborador dos Serviços Académicos e Docente

Pré condições: O utilizador tem de estar autenticado perante o sistema.

Cenário principal:

1. O utilizador encontra-se na página de entrada do Sistema de Gestão Académica.
2. Seleccionar a opção "Consultar Pauta".
   2.1. Se o utilizador é um docente passar para o passo 3.
   2.2. Se o utilizador é um colaborador dos Serviços Académicos incluir caso de utilização "Procurar Docente" e passar para o passo 3.
3. O sistema mostra as unidades curriculares leccionadas pelo docente.
4. O utilizador selecciona uma unidade curricular.
5. O sistema mostra os alunos inscritos à unidade curricular seleccionada.
6. O caso de utilização termina.

Exemplo 6.5 – Caso de utilização *"Consultar Pauta"*

As operações associadas ao caso de utilização *"Lançar Notas"* encontram-se especificadas no exemplo seguinte.

Nome: Lançar Notas

Descrição: Neste caso de utilização é permitido lançar notas para os alunos.

Actores: Docente

Pré condições: O utilizador tem de estar autenticado perante o sistema.

Cenário principal:

1. O utilizador encontra-se na página de entrada do Sistema de Gestão Académica.
2. Incluir o caso de utilização "Consultar Pauta".
3. O utilizador seleccionar a opção "Editar Nota(s)".
4. Introduz a(s) nota(s) pretendida(s).
5. Prime o botão "Guardar".
6. Se os dados são válidos, estes são guardados.
7. Prime o botão "Lançar Notas" e o caso de utilização termina.

Cenário alternativo 1

1. Idem até ao passo 5 do cenário principal.
2. O sistema mostra uma mensagem a indicar falta de preenchimento de campos obrigatórios.
3. O caso de utilização termina.

Cenário alternativo 2

1. Idem até ao passo 5 do cenário principal.
2. O sistema detecta dados inválidos.
3. O sistema mostra uma mensagem a informar o utilizador desse facto e o caso de utilização termina.

Exemplo 6.6 – Caso de utilização *"Lançar Notas"*

De seguida, são indicadas as operações associadas ao caso de utilização *"Criar Plano de Estudos"*.

Nome: Criar Plano de Estudos **Descrição:** Neste caso de utilização é criado o plano de estudos de um aluno quando este entra num curso possuindo um grau académico. **Actor:** Supervisor (Docente) **Pré condições:** O utilizador tem de estar autenticado perante o sistema. As unidades curriculares realizadas pelo aluno têm que estar introduzidas no sistema.
Cenário principal: 1. O utilizador encontra-se na página de entrada do Sistema de Gestão Académica. 2. Incluir o caso de utilização "Consultar Unidades Curriculares realizadas pelo Aluno". 3. Incluir o caso de utilização "Consultar Plano de Curso". 4. Seleccionar as unidades curriculares do curso do aluno que este terá que realizar para concluir o curso. 5. O utilizador prime o botão "Guardar" e o caso de utilização termina.

Exemplo 6.7 – Caso de utilização *"Criar Plano de Estudos"*

6.1.5. *Diagramas de Sequência*

Os diagramas de sequência ilustram interacções entre objectos num determinado período de tempo, realizadas por um determinado actor. Os diagramas de sequência são representados através de duas dimensões: a dimensão horizontal, que representa o conjunto de objectos intervenientes; e a dimensão vertical que representa o tempo. Em particular, os objectos são representados pelas suas "linhas de vida" e interagem por troca de mensagens ao longo de um determinado período de tempo.

Os diagramas de sequência são a realização dos casos de utilização e serviram para ilustrar as interacções entre os objectos do Sistema de Gestão Académica, de modo a realizar um determinado objectivo de um actor. Estes diagramas também ajudaram na implementação dos vários casos de utilização, permitindo definir na implementação de cada caso de utilização do SGA o modo de funcionamento estipulado nestes diagramas.

De seguida, são ilustrados os diagramas de sequência associados aos casos de utilização explicados na secção anterior, nomeadamente, *"Aceder à página de entrada"*, *"Criar Curso"*, *"Inscrever a Unidades Curriculares"*, *"Consultar Pauta"* e *"Lançar Notas"*.

O diagrama de sequência da Figura 6.10 ilustra as interacções entre os objectos do SGA envolvidos no caso de utilização *"Aceder à página de entrada"* (ver Exemplo 6.1). No caso do actor ser um aluno ou docente a página de entrada é referente à aplicação de *FrontOffice*. No caso do actor ser o colaborador dos serviços académicos a página de entrada é referente à aplicação de *BackOffice*.

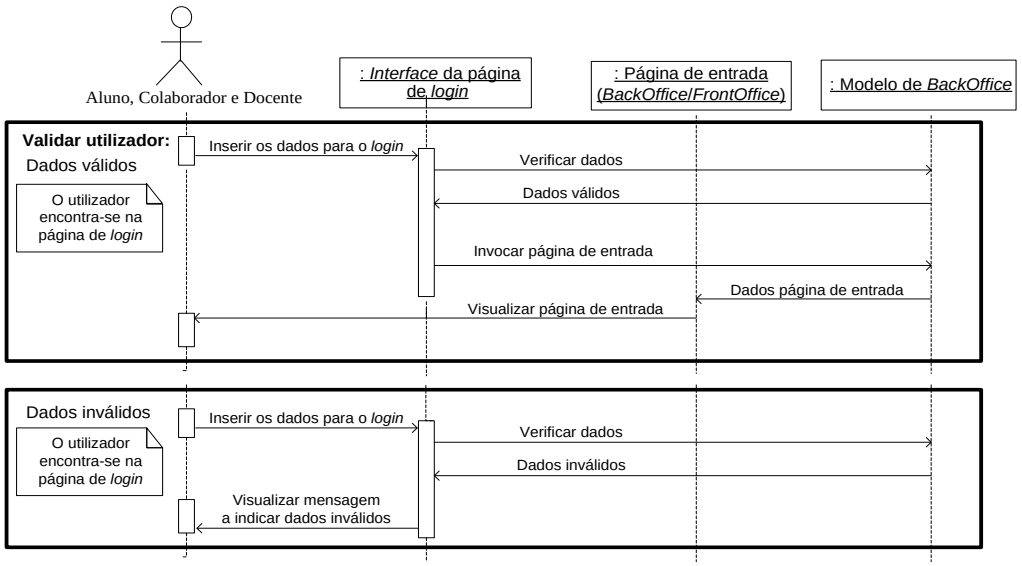


Figura 6.10 – Diagrama de sequência do caso de utilização *"Aceder à página de entrada"*

O diagrama de sequência da Figura 6.11 ilustra as interacções entre os objectos do SGA envolvidos no caso de utilização *"Criar Curso"* (ver Exemplo 6.3). O objecto *Curso (BackOffice)*, que se encontra ilustrado na figura seguinte, representa a página que é constituída pela informação que permite criar um curso.

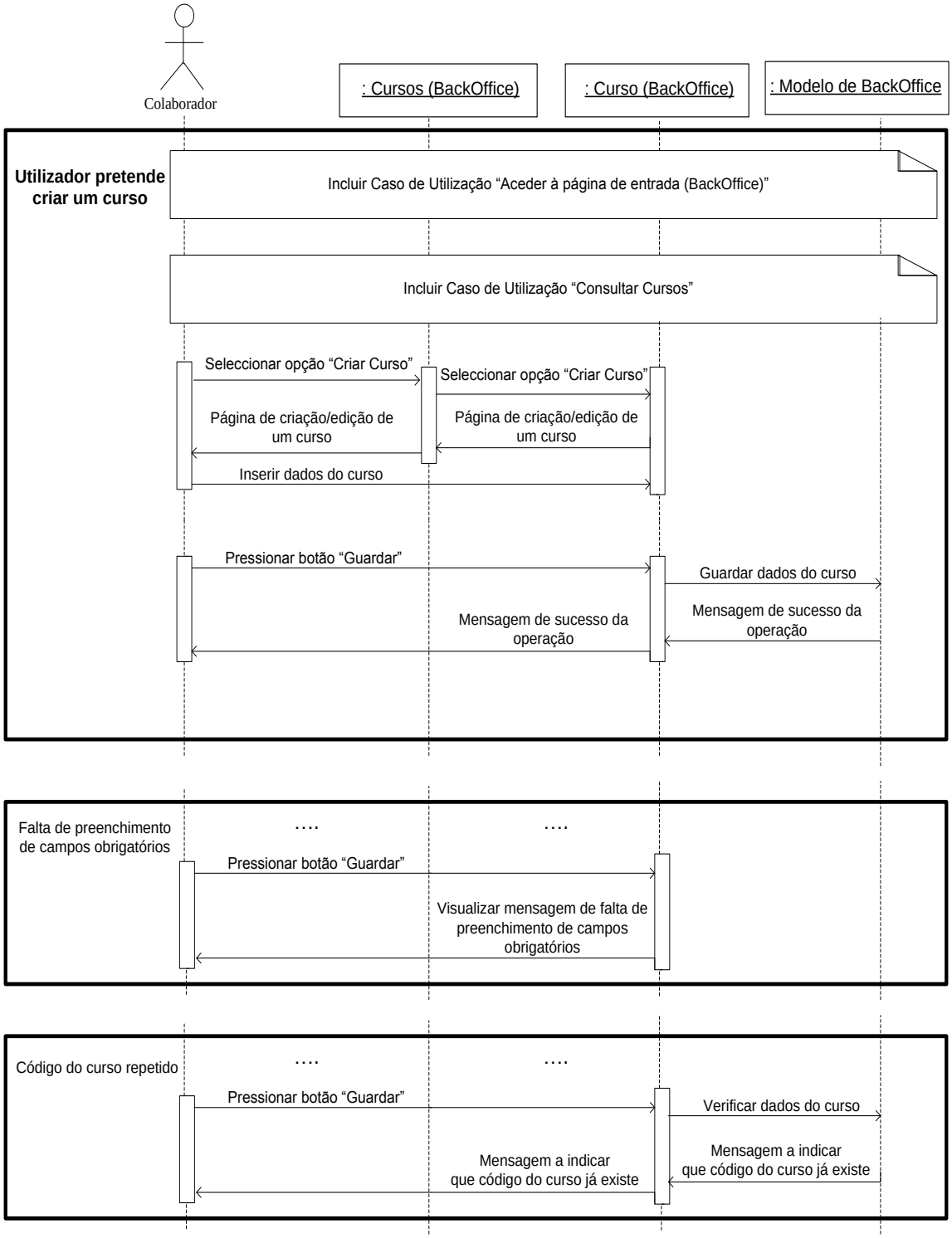


Figura 6.11 – Diagrama de sequência do caso de utilização *"Criar Curso"*

O diagrama de sequência da Figura 6.12 ilustra as interacções entre os objectos do SGA envolvidos no caso de utilização *"Inscrever a Unidades Curriculares"* (ver Exemplo 6.4). O objecto *Inscrição (FrontOffice)*, que se encontra ilustrado na figura seguinte, representa a página que permite realizar uma inscrição, através da selecção das unidades curriculares e das turmas pretendidas.

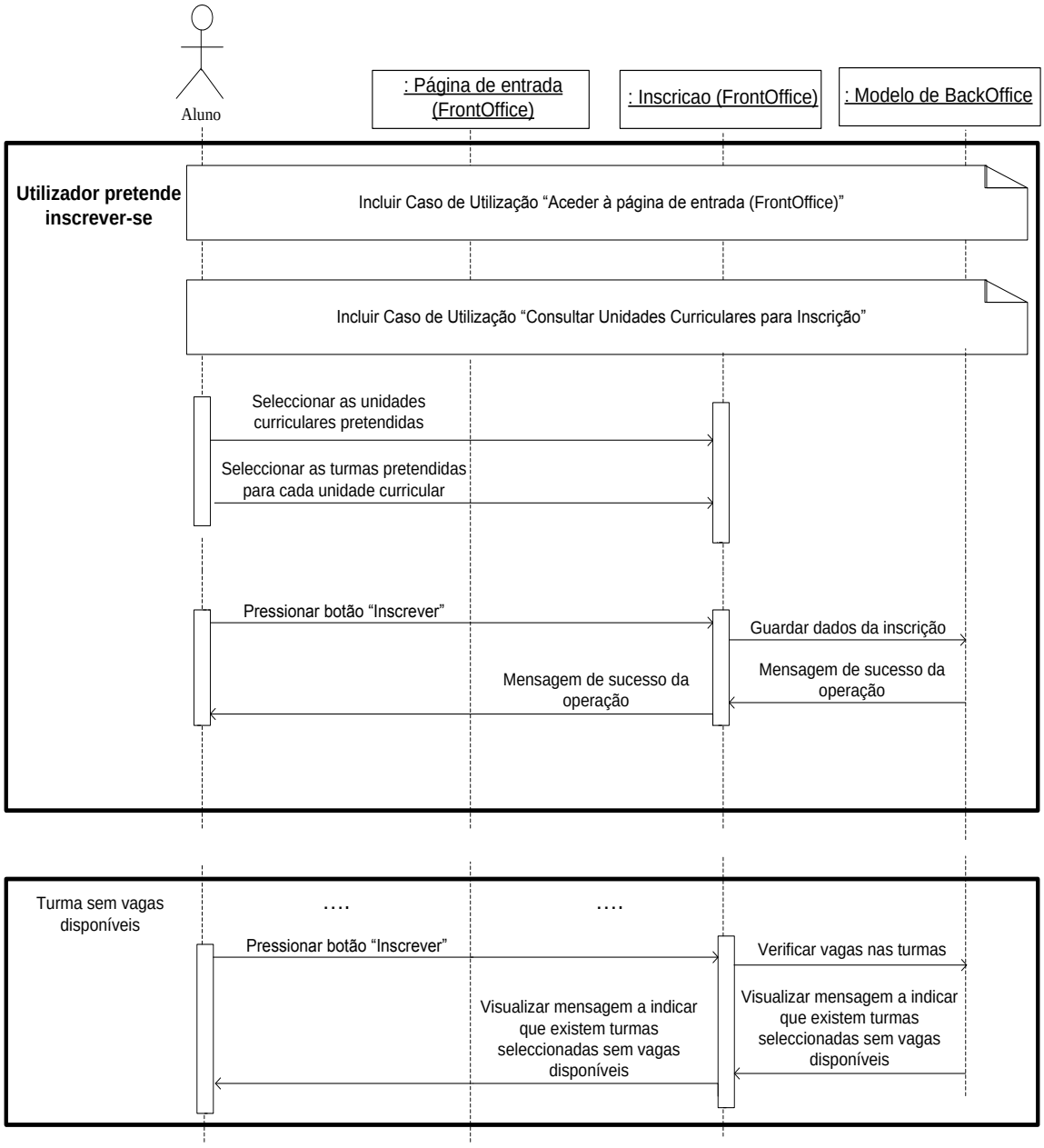


Figura 6.12 – Diagrama de sequência do caso de utilização *"Inscrever a Unidades Curriculares"*

O diagrama de sequência da Figura 6.13 ilustra as interacções entre os objectos do SGA envolvidos no caso de utilização *"Lançar Notas"* (ver Exemplo 6.6). O objecto *Pauta (FrontOffice)*, que se encontra ilustrado na figura seguinte, representa a página que permite visualizar a pauta de alunos inscritos a uma determinada unidade curricular. Através desta página é possível editar a nota de cada aluno.

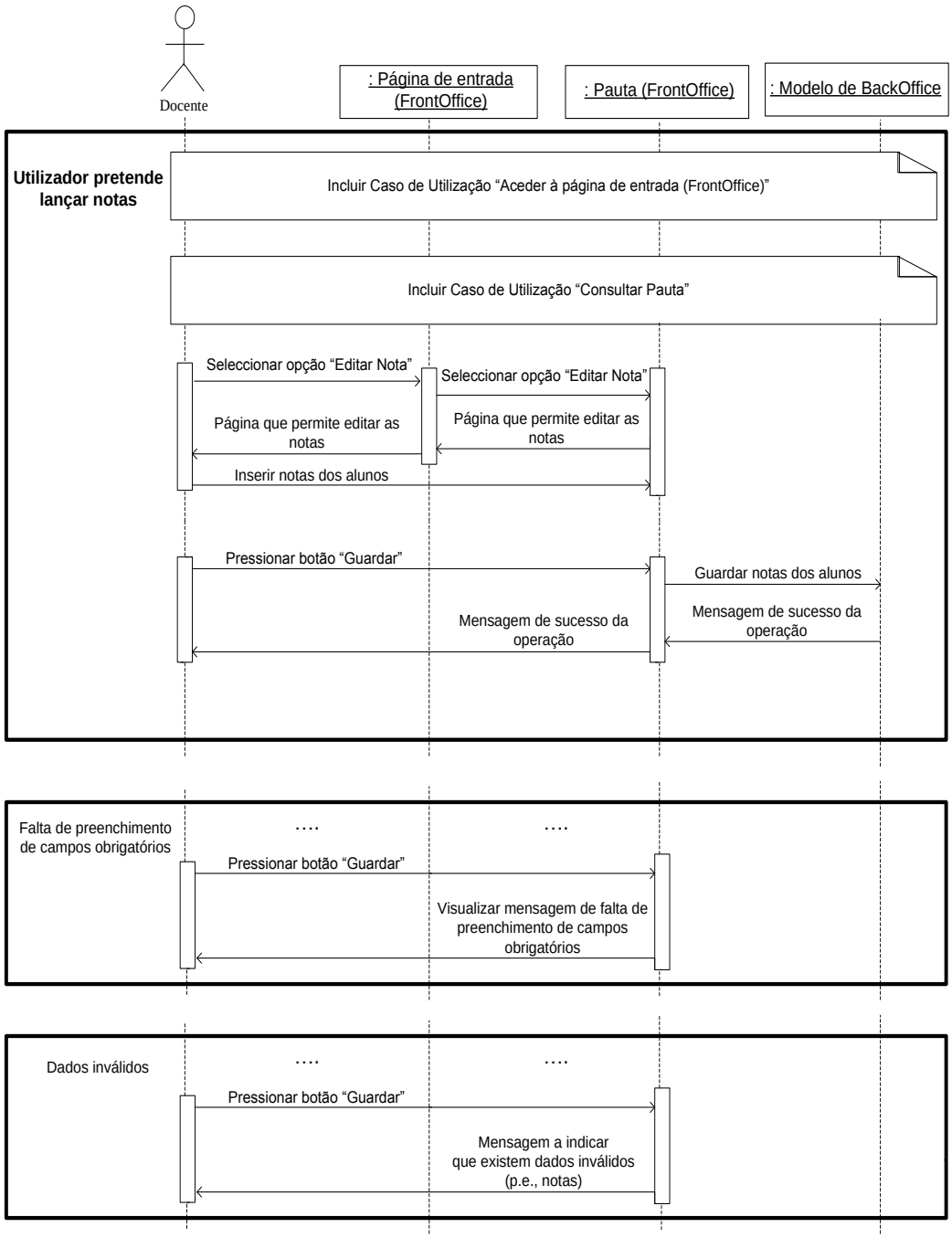


Figura 6.13 – Diagrama de sequência do caso de utilização *"Lançar Notas"*

6.1.6. *Modelação da dinâmica - Máquina de estados*

Uma máquina de estados permite modelar o comportamento de um determinado objecto, subsistema ou sistema global.

Estas máquinas representam os possíveis estados de um objecto, as correspondentes transições entre estados, os eventos que fazem desencadear as transições, e as operações (acções e actividades) que são executadas dentro de um estado ou durante uma transição [26]. Os objectos evoluem ao longo do tempo através de um conjunto de estados como resposta a eventos e à passagem de tempo.

A plataforma tem a capacidade de suportar objectos com estados. Por exemplo, o caso de utilização *"Lançar Notas"*, explicado anteriormente, tem associado o objecto pauta que tem três estados possíveis, nomeadamente, *"Pendente de Confirmação"*, *"Aceite"* e *"Rejeitada"*.

A Figura 6.14 ilustra a máquina de estados da pauta com os estados referidos.

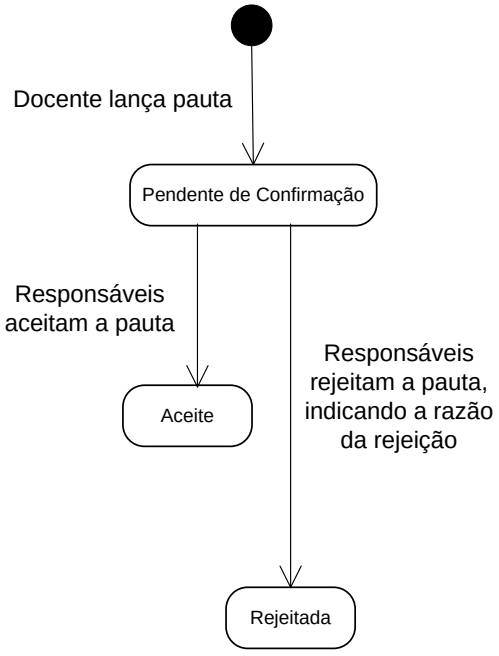


Figura 6.14 – Máquina de estados da pauta (Pendente de Confirmação, Aceite e Rejeitada)

6.2. Implementação

Para além da implementação dos casos de utilização mencionados anteriormente, também se teve em consideração os problemas referidos das aplicações do ISEL de modo a que estes fossem resolvidos com recurso à nova plataforma desenvolvida.

A implementação do Portal Académico consiste em seguir os passos descritos na secção *5.3 Configuração da Plataforma*.

Para resolver o problema das listagens incorrectas é necessário implementar os serviços que respondem ao que é pretendido e seguir os passos referidos anteriormente.

O problema dos filtros de pesquisa é resolvido automaticamente pela plataforma, como foi descrito na secção *5.1.1.1 Componentes Gráficos e Funcionalidades Desenvolvidas*.

Como o âmbito do trabalho não reside na definição da apresentação gráfica das aplicações *Web*, foi implementada uma *interface* de utilização simples mas funcional.

Foi realizada uma página base para a aplicação e os ficheiros de estilo necessários, originando a apresentação gráfica apresentada na Figura 6.15. Esta figura está associada ao caso de utilização *"Lançar Notas"*.

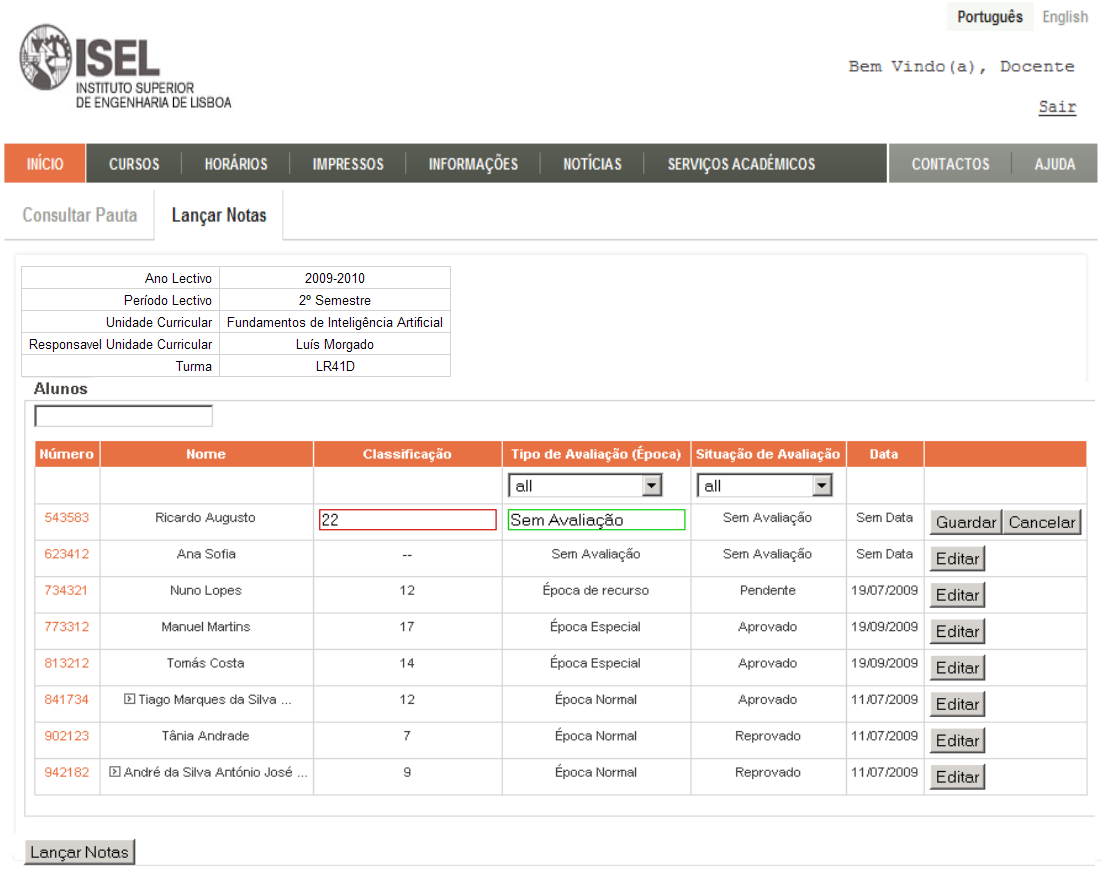


Figura 6.15 – Exemplo do Portal Académico (*"Lançar Notas"*)

As opções que deverão estar disponíveis a cada utilizador podem ser facilmente alteradas através do ficheiro de configuração explicado na secção *5.1.2.2 Controlo de Acessos*. Nesta secção é apresentado um *menu* para o perfil de Aluno que agora se apresenta integrado com o Portal Académico.

Figura 6.16 – *Menu* integrado no Portal Académico

Os requisitos de usabilidade são automaticamente respondidos pela utilização da plataforma, que devido à sua natureza implica directamente uma uniformização das aplicações. Esta uniformização concede à aplicação um aspecto estruturado e organizado, como se observa nas figuras anteriores.

7. Processo de Desenvolvimento

Este capítulo descreve a abordagem do problema, as diferentes fases do projecto, a metodologia de desenvolvimento adoptada e as aplicações utilizadas.

7.1. Abordagem do problema

Existe uma limitação natural da capacidade humana de lidar com a complexidade: não conseguimos estar em dois locais ao mesmo tempo, nem pensar em dois problemas simultaneamente. Por isso, o problema foi dividido em sub-problemas mais elementares e assim sucessivamente, até que a sua resolução fosse mais simples, ou seja, adoptou-se uma decomposição hierárquica.

Também a aplicação de um mecanismo de abstracção favorece a eliminação da complexidade: já que não é possível lidar com toda a realidade dos sistemas complexos, o ser humano opta por "esquecer" os detalhes menos importantes e focar a sua atenção nos mais relevantes, lidando com um modelo simplificado da realidade, mas considerado suficiente para entender e solucionar correctamente o problema em análise.

Para além destas duas ideias, da decomposição hierárquica e da abstracção, foram tidos em conta outros princípios considerados fundamentais para a produção de *software* de qualidade, designadamente:

- O desenvolvimento foi efectuado de forma iterativa, repetindo as mesmas actividades em momentos temporais desfasados, mas detalhando o âmbito das funcionalidades do sistema. Foi efectuada uma gestão integrada dos requisitos, permitindo a verificação da "rastreabilidade"[6] [26] dos mesmos desde o momento da sua identificação até à respectiva implementação, facilitando todo o processo de gestão de alterações.

[6] "**Rastreabilidade** é a capacidade de se seguir as relações entre os diferentes artefactos produzidos no processo de desenvolvimento de *software*, de forma a poder-se averiguar, por exemplo, o impacto que uma determinada alteração de um requisito tem em todos os restantes artefactos (na análise e na implementação)".

- Foi efectuada uma verificação sistemática da qualidade, não apenas no final do desenvolvimento.
- Foi realizado o controlo de alterações, de forma a gerir adequadamente um problema incontornável ("os requisitos de negócio mudam frequentemente") e definindo a forma e o momento em que essas alterações seriam contempladas no sistema.

Cada uma destas práticas tem impacto nas outras. Por exemplo, o desenvolvimento iterativo favorece a implementação de uma política de controlo de alterações, uma vez que ao diminuir o tempo que vai desde a identificação da necessidade até à disponibilização de uma versão funcional (se bem que parcial) da aplicação, as alterações que entretanto ocorram podem ser incorporadas na nova iteração.

7.2. Metodologia

O desenvolvimento do trabalho foi baseado numa metodologia ágil. Este tipo de metodologia tenta minimizar o risco do projecto, ao dividir o desenvolvimento de *software* em períodos de tempo curtos, designados de iterações. Por cada iteração é realizada uma fase de análise, de desenvolvimento, de teste (e refactorização) e uma demonstração do que foi implementado. No final do desenvolvimento, o conjunto de todas as demonstrações resulta no trabalho final.

A metodologia concreta utilizada é uma variante simplificada da metodologia *Scrum*[7], dividindo-se em três fases, como apresentado na Figura 7.1:

1. Preparação;
2. Desenvolvimento Iterativo;
3. Lançamento da Solução.

[7] ***Scrum*** é uma metodologia ágil para desenvolvimento de projectos, geralmente de *software*. Uma metodologia ágil tenta minimizar o risco no desenvolvimento de *software* e assenta numa comunicação frequente com as partes interessadas, com o objectivo de atingir a solução pretendida.

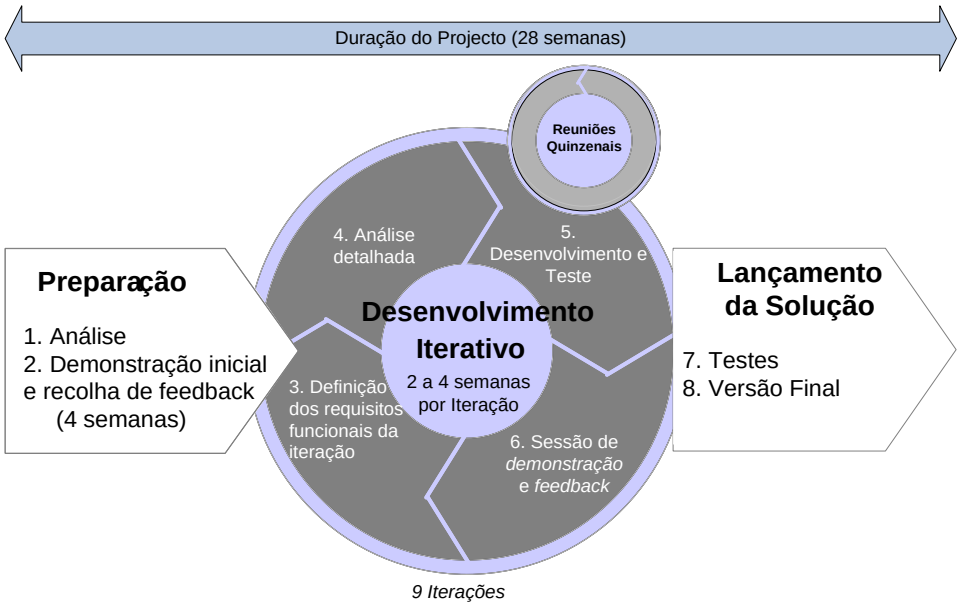


Figura 7.1 – Actividades realizadas com base na metodologia adoptada

A metodologia adoptada promove a interacção e o *feedback* com as partes envolvidas no projecto com a finalidade de se obter uma compreensão comum da solução e quais os requisitos do sistema. Deste modo, o *feedback* apresentado pelos utilizadores permite convergir para a solução funcional e ter uma solução completamente integrada a todo o momento, promovendo um baixo custo de mudança. Durante as sessões de *feedback* com o cliente são avaliadas as prioridades dos requisitos. Novos requisitos com prioridades elevadas são adicionados e a funcionalidade existente pode ser adaptada a fim de aumentar o valor da solução.

Na maioria das vezes, os utilizadores tomam decisões no início do projecto quando ainda não tiveram nenhuma experiência com a solução. A possibilidade dos utilizadores terem contacto com uma *interface* visual do sistema no início do projecto, se bem que ainda de reduzido detalhe, permite que eles possam dar o seu *feedback* para que se oriente / direccione o projecto para os objectivos a atingir.

Para além dos testes efectuados durante o processo de desenvolvimento foram realizados testes finais, para determinar se as funcionalidades que cada utilizador pode efectuar estão de acordo com os requisitos funcionais pretendidos. Os erros de *interface* e de funcionamento foram tidos em consideração, e na existência de erros estes foram corrigidos. Depois destes testes obteve-se a versão final da plataforma.

7.3. Planeamento

O planeamento foi dividido em várias fases, perfazendo um total de 28 semanas úteis para análise, desenvolvimento e testes. A partir da demonstração inicial por cada 2 ou 4 semanas foi disponibilizada uma demonstração do projecto. O planeamento efectuado encontra-se ilustrado na Figura 7.2.

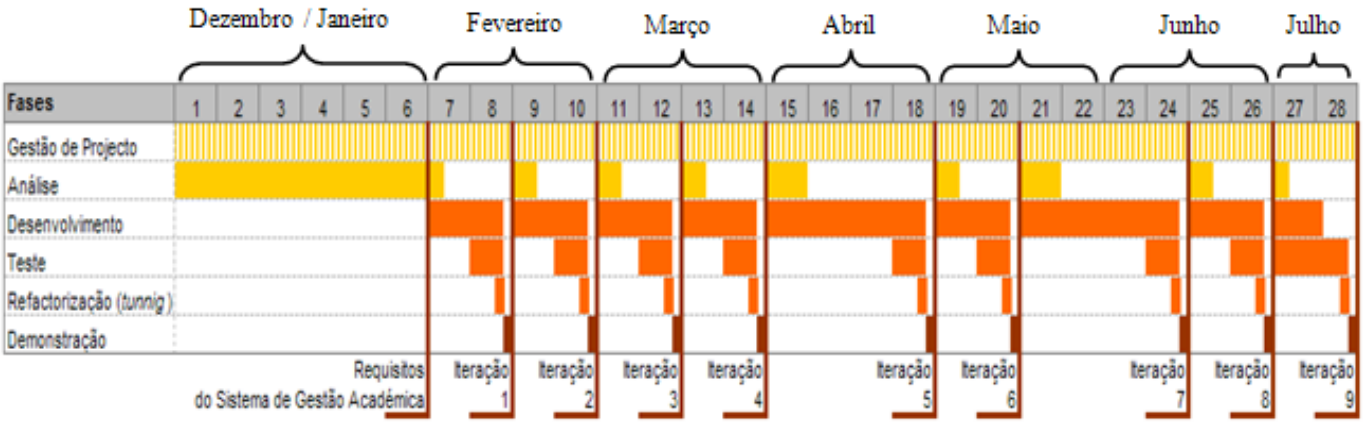


Figura 7.2 – Fases de execução do projecto

Em todas as fases do projecto foi realizada a gestão do projecto para se controlar o ritmo e se o trabalho desenvolvido corresponde ao que foi planeado e às expectativas inicialmente previstas. De seguida, é realizada uma descrição de cada fase onde se menciona quais as tarefas que foram realizadas.

Fase de Análise (seis semanas):

- Estudo do trabalho de projecto "ISELnet – Um portal *Internet* para o ISEL" [22]. Este trabalho aborda a implementação de uma plataforma de integração ao nível da camada de apresentação, seguindo os conceitos *Web* 2.0, assente em tecnologias cliente suportadas em navegadores *Web*. Foi analisado o modo como a arquitectura foi concebida, as tecnologias usadas, o modo de comunicação entre os vários serviços e o modo como o projecto foi gerido. Este estudo permitiu assimilar novos conceitos utilizados na construção de aplicações *Web*.
- Tendo em conta que a prova de conceito consiste na implementação de uma aplicação *front office* relacionada com um Sistema de Gestão Académica foi realizada uma análise de requisitos e definidos os vários casos de utilização que

se pretende contemplar. Tendo em consideração os requisitos foi definido o formato modelo das mensagens *ACL* que irão ser utilizadas entre os vários componentes da plataforma.

Iteração 1 (duas semanas):

- Implementação da *Plataforma de Automatização da Apresentação* (páginas) com base em *Javascript* e *DOM*. Esta plataforma interpreta a mensagem *ACL* e com base nela cria os componentes *HTML* necessários. Tendo em conta os *performatives input* (modo de edição) e *output* (modo de leitura), são criados os componentes correspondentes tendo em conta o conteúdo da mensagem. Nesta primeira iteração pretende-se demonstrar a criação de componentes através de mensagens *ACL*, nomeadamente a apresentação quer em modo leitura quer em modo de edição da estrutura de um curso, de uma unidade curricular e uma pauta de alunos inscritos a uma unidade curricular. Apesar do protótipo requerer alguns melhoramentos, nomeadamente no que diz respeito à validação de tipos existentes num formulário, decidiu-se realizar essa tarefa na iteração 6 de modo a que nas iterações seguintes se implemente a *Plataforma de Acesso a Dados* para que se tenha rapidamente um protótipo que espelhe o funcionamento de toda a plataforma (vertente *Web* e de acesso a dados).

Iteração 2 (duas semanas):

- Iteração idêntica à iteração 1 mas considerando o *feedback* obtido. A opção por dividir a iteração 1 em duas iterações mais curtas (em vez de uma única iteração de um mês) deve-se ao facto de na fase inicial do desenvolvimento ser muito importante obter *feedback* em menor tempo, e desta forma, caso seja necessário efectuar alguma alteração ela será realizada nesta iteração e não numa fase posterior.

Iteração 3 (duas semanas):

- Implementação da *Plataforma de Acesso a Dados*. Esta plataforma tem como objectivo interpretar mensagens *ACL* e construir a *query SQL* necessária para responder à acção descrita na mensagem. Foi construído um dicionário que

permite relacionar atributos da mensagem com campos da base de dados. Foi definida a arquitectura a utilizar para implementar a plataforma.

Iteração 4 (duas semanas):

- Iteração idêntica à iteração 3 mas considerando o *feedback* obtido. Como na iteração 3 foi iniciada uma nova fase de desenvolvimento dividiu-se a iteração 3 em duas de modo a obter *feedback* em menor tempo. Desta forma, possíveis alterações poderão ser realizadas nesta fase em vez de serem adoptadas numa fase posterior. No final desta iteração e com base nas funcionalidades implementadas na iteração 1, nomeadamente, o modo de edição da estrutura de um curso, de uma unidade curricular e de uma pauta de alunos inscritos a uma unidade curricular foi testado se a *Plataforma de Acesso a Dados* consegue interpretar correctamente as mensagens *ACL* associadas a estas funcionalidades e gerar a *query SQL* correspondente.

Iteração 5 (quatro semanas):

- Continuação da implementação da *Plataforma de Acesso a Dados*. O objectivo desta iteração é interpretar o resultado do pedido realizado pelo utilizador, em formato *SQL*, e associar à mensagem *ACL* de acordo com o modelo predefinido da mensagem. Esta mensagem será enviada à plataforma de automatização de páginas que a sabe interpretar e construir os componentes *HTML* necessários.

Iteração 6 (duas semanas):

- Tendo em conta que na iteração 1 não se conseguiu dotar a *Plataforma de Automatização da Apresentação Web* com um mecanismo de validação de tipos foi nesta iteração que foi realizado o componente responsável por essa tarefa (uma semana). Para além desta tarefa, foi implementado o componente que com base em mensagens *ACL* e sempre que se justifique cria os filtros de pesquisa necessários, ou seja, se o conteúdo da mensagem conter listagens/tabelas irão ser criados os filtros de pesquisa que fazem sentido para cada caso. Estas funcionalidades irão tornar a *Plataforma de Automatização da Apresentação* mais robusta e rica, características que também se vão reflectir nas páginas

cliente que forem criadas. Devido à necessidade de implementar um novo componente *HTML* (tabela com opção de selecção de linhas) para responder aos requisitos do caso de utilização que permite ao aluno efectuar uma inscrição decidiu-se inserir esta tarefa na iteração 6 o que originou um atraso de duas semanas fase ao planeamento inicialmente definido. Face a isto, a iteração 7 foi iniciada na segunda quinzena de Junho.

Iteração 7 (quatro semanas):

- Nesta iteração implementou-se o sistema de coordenação de páginas de forma a gerir-se o fluxo entre as mesmas.
- Uma vez que cada caso de utilização é representado por uma (ou mais) entidade de informação e tendo em conta que é a partir desta que é construída a mensagem foi implementado o módulo capaz de construir a mensagem através da descrição da entidade (*EntityInformation2Message.js*). O módulo pode ser visto como um interpretador e um tradutor. Interpretador porque interpreta o conteúdo da entidade de informação e tradutor porque traduz esse conteúdo para o formato correspondente na mensagem *ACL*.

Iteração 8 (duas semanas):

- Uma vez que as páginas de interacção com o utilizador estão associadas a casos de utilização foi implementado o módulo que permite registar para cada actor os vários casos de utilização (*Gestor de Acessos*).
- Foi implementado o componente *AccessControl* que disponibiliza as funcionalidades tendo em conta o perfil de utilizador, ou seja, consoante o perfil assim são mostradas as funcionalidades correspondentes.

Iteração 9 (duas semanas):

- Nesta iteração foi aplicado estilo gráfico aos componentes gerados pela plataforma de apresentação e foram efectuados testes mais exaustivos sobre toda a plataforma de modo a se optimizar e corrigir situações detectadas.
- Implementação do protótipo (Portal Académico).

- Sendo a metodologia adoptada ágil significa que em determinados momentos há a necessidade de aplicar mudanças ao que tinha sido inicialmente previsto, o que poderá originar atrasos em algumas fases. Esta fase serviu também para colmatar os atrasos que eventualmente surgiram ao longo do projecto.

7.4. Aplicações utilizadas no desenvolvimento

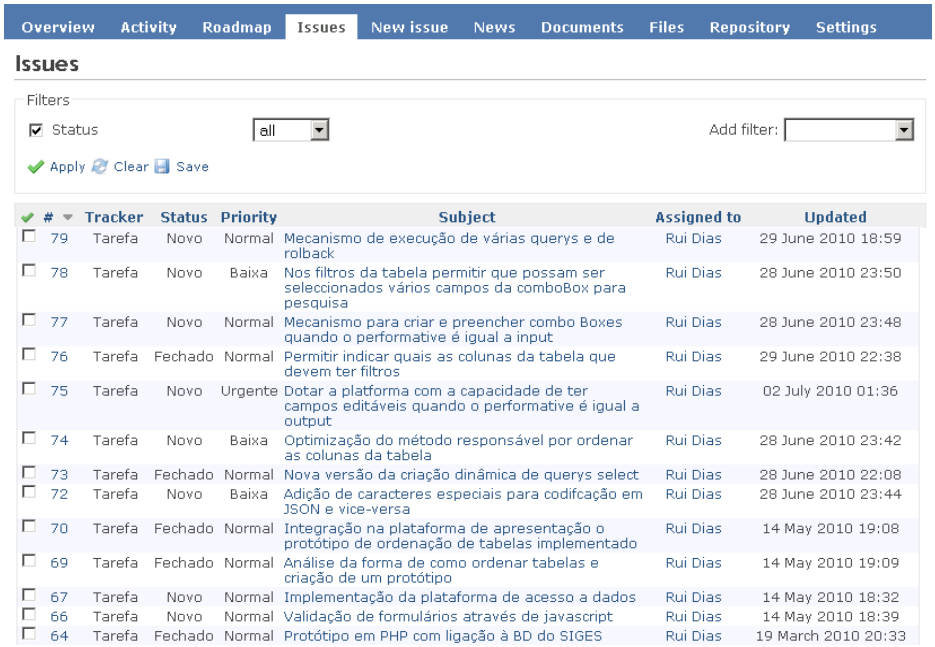
Para a plataforma cliente foi utilizada a aplicação *1st Javascript Editor* [33] porque oferece uma biblioteca completa de *tags HTML*, atributos *HTML*, eventos *HTML*, eventos e funções em *Javascript*, atributos, declarações e operadores, permitindo serem inseridos no código através de um clique. Outra característica que levou a ser escolhida esta aplicação deve-se ao facto de incluir um *debugger Javascript*.

Como a plataforma servidora foi implementada em *PHP* foi utilizada a aplicação *Eclipse for PHP Developers* [34] com a ferramenta de *debugging* denominada *Zend Debugger* [35].

Para a realização dos diagramas de classe foi utilizada a aplicação *Enterprise Architect Version 8* [36] porque permite obter a partir do código fonte as classes e vice-versa.

Como suporte ao processo de desenvolvimento foi adoptada a plataforma de gestão de projecto *Redmine* [37]. Uma plataforma deste tipo permite que vários intervenientes actuem sobre os requisitos do projecto como também no seu planeamento, permitindo a disponibilização de documentação e recepção de *feedback*.

A plataforma *Redmine* foi utilizada principalmente para planeamento, definindo as tarefas necessárias a realizar com base nas funcionalidades e organizando-as temporalmente, definindo prioridades e encadeamentos. É possível através da utilização desta ferramenta fazer uma gestão das funcionalidades a desenvolver, obter *feedback* e gerir a documentação criada.



#	Tracker	Status	Priority	Subject	Assigned to	Updated
79	Tarefa	Novo	Normal	Mecanismo de execução de várias querys e de rolback	Rui Dias	29 June 2010 18:59
78	Tarefa	Novo	Baixa	Nos filtros da tabela permitir que possam ser seleccionados vários campos da comboBox para pesquisa	Rui Dias	28 June 2010 23:50
77	Tarefa	Novo	Normal	Mecanismo para criar e preencher combo Boxes quando o performative é igual a input	Rui Dias	28 June 2010 23:48
76	Tarefa	Fechado	Normal	Permitir indicar quais as colunas da tabela que devem ter filtros	Rui Dias	29 June 2010 22:38
75	Tarefa	Novo	Urgente	Dotar a platforma com a capacidade de ter campos editáveis quando o performative é igual a output	Rui Dias	02 July 2010 01:36
74	Tarefa	Novo	Baixa	Optimização do método responsável por ordenar as colunas da tabela	Rui Dias	28 June 2010 23:42
73	Tarefa	Fechado	Normal	Nova versão da criação dinâmica de querys select	Rui Dias	28 June 2010 22:08
72	Tarefa	Novo	Baixa	Adição de caracteres especiais para codificação em JSON e vice-versa	Rui Dias	28 June 2010 23:44
70	Tarefa	Fechado	Normal	Integração na plataforma de apresentação o protótipo de ordenação de tabelas implementado	Rui Dias	14 May 2010 19:08
69	Tarefa	Fechado	Normal	Análise da forma de como ordenar tabelas e criação de um protótipo	Rui Dias	14 May 2010 19:09
67	Tarefa	Novo	Normal	Implementação da plataforma de acesso a dados	Rui Dias	14 May 2010 18:32
66	Tarefa	Novo	Normal	Validação de formulários através de javascript	Rui Dias	14 May 2010 18:39
64	Tarefa	Fechado	Normal	Protótipo em PHP com ligação à BD do SIGES	Rui Dias	19 March 2010 20:33

Figura 7.3 – Algumas tarefas associadas ao projecto (*Redmine*)

A plataforma permite que sejam adicionados comentários a cada tarefa à medida que esta vai evoluindo (ver figura seguinte), disponibilizando uma importante fonte de conhecimento sobre cada funcionalidade implementada.

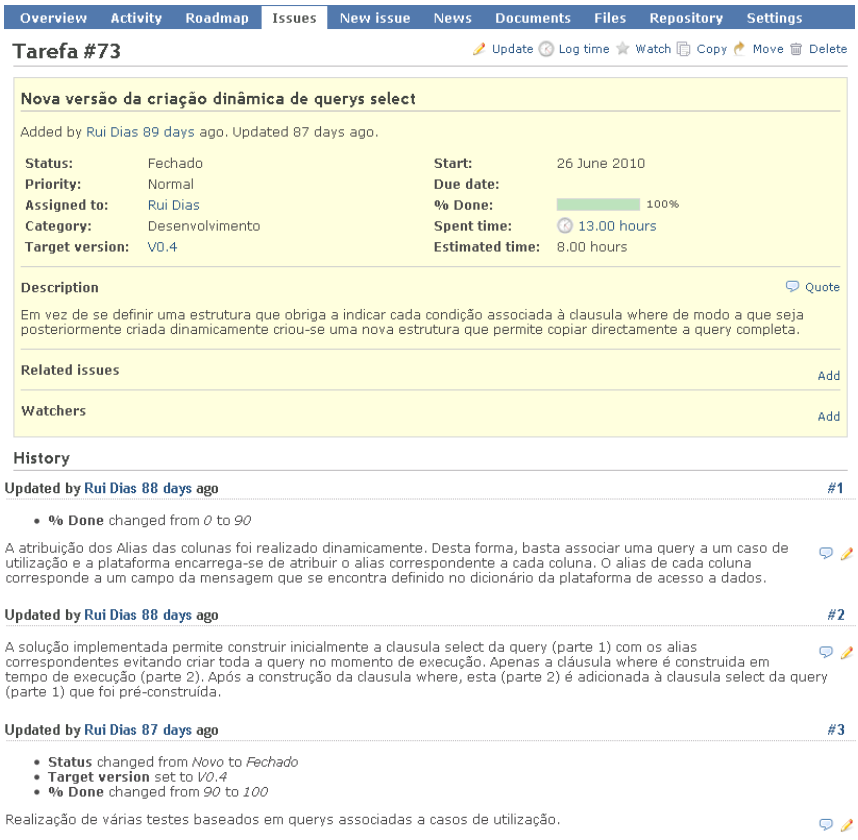


Figura 7.4 – Detalhes associados a uma tarefa (*Redmine*)

Esta plataforma permite que a equipa de desenvolvimento e o cliente possam estar mais sincronizados, havendo um seguimento real do estado do projecto, permitindo também a cooperação mais fácil dentro da equipa. A plataforma também foi utilizada para *upload* de ficheiros e como um servidor de versões. Como ferramenta auxiliar para realizar o *upload* de ficheiros e para realizar o controlo destes ficheiros foi utilizada a aplicação *TortoiseSVN* [38].

Neste projecto, o uso da plataforma *Redmine* consistiu na disponibilização do estado do projecto ao orientador, na organização do trabalho realizado, no trabalho em desenvolvimento e na disponibilização e organização de documentação. Também foram registadas as dificuldades encontradas para cada tarefa e as opções tomadas no desenvolvimento de cada funcionalidade.

Desta forma, toda a informação relativa ao projecto foi centralizada e gerida nesta aplicação de uma forma simples e intuitiva.

8. Conclusões e Trabalho Futuro

Este projecto teve como objectivo desenvolver uma plataforma que automatize quer a apresentação dos conteúdos das mensagens quer o acesso a dados.

A plataforma foi concebida de forma modular, no sentido de garantir uma organização funcional coesa, tornando possível a sua utilização como base para a construção de outras aplicações *Web*.

8.1. Conclusões

A separação da plataforma em duas plataformas, uma responsável pela apresentação de conteúdos (plataforma cliente) e outra pelo acesso a dados (plataforma servidora), permitiu o desenvolvimento independente de cada parte. Esta separação para além de tornar o desenvolvimento independente de cada plataforma possibilitou que fossem realizados testes independentemente, simplificando desta forma o ambiente de desenvolvimento e de operação.

A separação da apresentação dos conteúdos permite que o formato dos dados não esteja vinculado a um único formato de apresentação, o que permite uma maior flexibilidade da aplicação. Com esta separação é possível garantir a divisão de responsabilidades entre aqueles que têm acesso aos conteúdos (e aos processos) e aqueles que apresentam os conteúdos. Assim, é possível atribuir a um programador a responsabilidade de desenvolver os serviços com que a aplicação vai comunicar e disponibilizar os dados para tal, enquanto o *designer* fica com a responsabilidade da criação gráfica da aplicação.

A comunicação entre os vários componentes da plataforma é feita com base em mensagens. A utilização de mensagens possibilita um desacoplamento entre as partes da aplicação, tornando-a mais modular e adaptativa. Os benefícios da utilização de mensagens para a comunicação entre componentes são os de não fazer um componente depender da existência de outro e poder dissociar completamente os componentes da origem dos seus dados. Desta forma, a plataforma não se encontra comprometida com

serviços específicos disponibilizados por plataformas servidoras, sendo esses serviços dependentes apenas da configuração das mensagens *ACL*.

A plataforma tem a capacidade de comunicar com serviços independentes, mesmo que esses serviços estejam localizados em máquinas diferentes. Isto é possível através de um módulo que permite registar o *endpoint* responsável por fornecer determinado serviço. Esta capacidade de comunicar com *endpoints* distintos disponíveis na *Web* e a possibilidade de redireccionamento de mensagens com base em assuntos disponibiliza um ponto de extensão importante levando o conceito de *SOA* à *Internet*, utilizando como base uma *Internet* de serviços.

A utilização da plataforma permite uniformizar as aplicações. Esta uniformização concede à aplicação um aspecto estruturado e organizado.

A existência do dicionário facilita a internacionalização das aplicações sendo mais um ponto forte da plataforma.

A concretização deste projecto passou pela elaboração de um Portal Académico, que serviu para demonstrar o funcionamento da plataforma e dos conceitos aplicados.

A metodologia ágil adoptada permitiu minimizar o risco do projecto, ao dividir o desenvolvimento de *software* em períodos de tempo curtos, designados de iterações. No final de cada iteração foi apresentada uma demonstração do projecto, obtendo-se *feedback* por parte do cliente, facilitando assim a compreensão se o caminho seguido até essa iteração era o pretendido. A utilização de uma metodologia de desenvolvimento suportada numa ferramenta de gestão de projecto foi muito importante para uma gestão eficiente do projecto. Apesar do tempo dispendido em gestão, este revelou-se essencial porque deu origem a uma redução no tempo de desenvolvimento.

8.2. Trabalho Futuro

Após o trabalho realizado identificam-se duas áreas para trabalho futuro, nomeadamente, a implementação de novas funcionalidades gráficas e a criação de um mecanismo de configuração da plataforma baseado numa *interface* gráfica.

Para além das diversas funcionalidades gráficas disponibilizadas podem ser disponibilizadas outras para conferir às aplicações outras funcionalidades, complementando a plataforma lógica desenvolvida. Por exemplo, para além dos filtros de pesquisa disponibilizados pela plataforma nas tabelas em que só é possível escolher

uma opção por filtro (*combo box*), permitir que nos filtros da tabela possam ser seleccionados vários campos da *combo box* para pesquisa, semelhante aos filtros da aplicação *Excel*. Desta forma, possibilita-se que o utilizador seleccione mais do que uma opção por filtro, tornando a utilização da aplicação mais versátil.

Tendo em conta que a configuração da plataforma é baseada em ficheiros, a criação de um mecanismo de configuração da plataforma baseado numa *interface* gráfica (*wizard*) irá tornar esta tarefa mais intuitiva e rápida.

Bibliografia

[1] Leon Shklar and Richard Rosen, "*Web Application Architecture - Principles, protocols and practices*", John Wiley & Sons Ltd, 2003.

[2] Bill Weinman, *"The CGI Book"*, New Riders.

[3] Jason Hunter and William Crawford, *"Java Servlet Programming"*, O'Reilly Media, second edition, April 2001.

[4] John Farrar, *"ColdFusion 9 Developer Tutorial"*, Packt Publishing Ltd, July 2010.

[5] *Llc Books*, *"CFML Programming Language"*, *Llc Books*, September 2010.

[6] *Digitalis*, *"DIF Introduction"*, Digitalis, 5 de Janeiro de 2005.

[7] *OutSystems, "OutSystems Platform Technical Overview", OutSystems, 2001-2007.*

[8] *OutSystems*, *"OutSystems Platform 4.0 - Product Data Sheet"*, *OutSystems*.

[9] Michael Azoff, *"OutSystems Platform 4.0"*, Butler Group, October 2006.

[10] *Echo Web Platform*, http://echo.nextapp.com/site/, 12 de Agosto de 2010.

[11] Marc Loy, Robert Eckstein, Dave Wood, James Elliott, Brian Cole, *"Java Swing"*, O'Reilly Media, second edition, November 2002.

[12] *Eclipse Standard Widget Toolkit (SWT)*, http://www.eclipse.org/swt/, 12 de Agosto de 2010.

[13] *Architectural Overview of Echo Web Platform*, http://echo.nextapp.com/site/node/67, 12 de Agosto de 2010.

[14] *Java Servlet Container*, http://onjava.com/pub/a/onjava/2003/05/14/java_*Web*server.html, 12 de Agosto de 2010.

[15] Andrew Harbourne-Thomas, Bjarki Holm, John Bell, Meeraj Kunnumpurath, Sam Dalton, Simon Brown, Sing Li, Subrahmanyam Allamaraju, Tony Loton, "*Professional Java Servlets 2.3*", Peer Information, 1st edition, January 2002.

[16] *Modules of Echo Web Platform*, http://echo.nextapp.com/site/node/167, 12 de Agosto de 2010.

[17] *Echo Client-Side Application*, http://echo.nextapp.com/site/node/83, 12 de Agosto de 2010.

[18] *Echo Server-Side Application*, http://echo.nextapp.com/site/node/68, 12 de Agosto de 2010.

[19] *W3Schools Browser Statistics*, http://www.w3schools.com/browsers/browsers_stats.asp, 14 de Agosto de 2010.

[20] Tim O'Reilly, "*What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software*", O'Reilly Media, 2007.

[21] John Musser, "*Web 2.0 Principles and Best Practices*", O'Reilly Radar, 2007.

[22] Hugo Sousa Raimundo, "ISELnet – Um portal *Internet* para o ISEL", Setembro de 2009.

[23] *Foundation for Intelligent Physical Agents*, *FIPA ACL Message Structure Specification* (http://www.fipa.org/specs/fipa00061/SC00061G.pdf), 23 de Fevereiro de 2010.

[24] *Java Script Object Notation*, http://www.JSON.org/, 2 de Junho de 2010.

[25] Sven Casteleyn, Florian Daniel, Peter Dolog, Maristella Matera, "*Engineering Web Applications*", Springer, 2009.

[26] Alberto Silva e Carlos Videira, "*UML, Metodologias e Ferramentas CASE*, Centro Atlântico", 1ª edição, Abril de 2001.

[27] Robert Hoekman Jr., *"Designing the Obvious: A Common Sense Approach to Web Application Design"*, New Riders, 2006.

[28] Gerald Carter, *"LDAP System Administration"*, O'Reilly Media, March 2003.

[29] Gerti Kappel, Birgit Proll, Siegfried Reich, Werner Retschitzegger, "*Web Engineering*", John Willey & Sons Ltd, 2006.

[30] *JSON encoder / decoder* (*Javascript*), http://devpro.it/JSON/files/JSON-js.html, 4 de Março de 2010.

[31] *JSON encoder / decoder* (*PHP*), http://mike.teczno.com/JSON/JSON.phps, 2 de Abril de 2010.

[32] *PHP: LDAP – Manual*, http://php.net/manual/en/book.ldap.php, 27 de Agosto de 2010.

[33] *1st Javascript Editor*, http://www.yaldex.com/JSFactory_Pro.htm, 17 de Fevereiro de 2010.

[34] *Eclipse for PHP Developers*, http://www.eclipse.org/downloads/, 19 de Abril de 2010.

[35] *Debugging PHP using Eclipse and PDT*, http://www.eclipse.org/pdt/articles/debugger/os-php-eclipse-pdt-debug-pdf.pdf, 19 de Abril de 2010.

[36] *Enterprise Architect*, http://www.sparxsystems.com/, 1 de Fevereiro de 2010.

[37] *Redmine*, http://www.redmine.org/, 13 de Janeiro de 2010.

[38] *TortoiseSVN*, *http://tortoisesvn.tigris.org/*, 17 de Janeiro de 2010.