Laboratórios de Engenharia de Software

Análise de Tecnologias

Universidade do Porto

Faculdade de Engenharia

FEUP

André Moniz{ei99041@fe.up.pt}

José Fonseca {ei99032@fe.up.pt}

Mário Pereira {ei99047@fe.up.pt}

Miguel Sarmento {ei96049@fe.up.pt}

21 de Outubro de 2002

Conteúdo

1 Introdução

A análise das tecnologias usadas num sistema é um ponto de enorme importância quando se elabora um projecto. É de extrema relevânica estudar e analisar um número variado de hipóteses tecnológicas de forma a que a escolha das tecnologias a usar seja um decisão ponderada e estudada. Depois de escolhidas as tecnologias, um estudo aprofundado foi efectuado de maneira a não termos só uma visão geral do material sobre o qual estamos a trabalhar, mas sim um conhecimento mais técnico e meticuloso.

2 Plataforma .NET

2.1 Introdução

Nos últimos quatro anos, a Microsoft investigou, concebeu, desenvolveu e testou uma nova plataforma de computação que inclui ferramentas de programação, frameworks para construção de aplicações e Web Services, e uma infra-estrutura comum de compilação, execução, segurança e distribuição. Esta plataforma, denominada .NET, foi concebida com base em standards Web e na ideia da interoperabilidade. Em conjunto com a iniciativa Shared Source, esta plataforma fornece uma base excepcional tanto para o desenvolvimento de software comercial como para a investigação.

2.2 Arquitectura .NET

Na arquitetura .NET, todas as aplicações são desenvolvidas sobre componentes modulares denominados "assemblies", que são semi-compilados de maneira a ser chamado por "p-code". No momento da execução dos Módulos (assemblies), a compilação é completada por um software chamado (CLR) Common Language Runtime.

O melhor de tudo isso é que este CLR não é executado apenas sob Windows. Então podemos ter versões para Linux, Solaris etc.

Outra característica impressionante da arquitetura .NET é que uma aplicação pode ser desenvolvida com linguagens que tenham as sintaxes de VB, Delphi, PERL, Visual C++, COBOL entre outras. Todas estas linguagens utilizam a mesma biblioteca chamada ".NET Base Class Library"onde temos recursos para aceder a base de dados, criação de interfaces gráficas e até mesmo criação de páginas dinâmicas para Internet.

Para desenvolver esta nova arquitetura a linguagem C# (pronuncia-se C sharp) usada na plataforma .NET, que substitui o Java da Microsoft, foi contratado nada mais nada menos que Anders Hejlsberg que estava na Borland, onde criou o Turbo Pascal e o Delphi.

Uma diferença radical em relação ao Java é que, as aplicações .NET comunicam pela Internet usando o protocolo HTTP enquanto o Java utiliza o RMI. Comparativamente podemos considerar o XML como não compilado. O RMI usa formato binário.

Finalmente, o Java trabalha mantendo a conexão entre servidor e cliente (stateful) enquanto as aplicações .NET trabalham principalmente com o tipo internitente de conexões que caracterizam o protocolo HTTP da web (stateless).

Apesar de se ter comentando que a linguagem e arquitetura .NET são clones do Java, elas são bem diferentes: o Java utiliza applets que são processados na máquina do cliente, enquanto as aplicações .NET criam interfaces em HTML e eventualmente faz alguma consistência de entradas no cliente.

Uma outra inovação da Microsoft foi colocar disponível para download gratuito pela Internet, o kit básico que permite desenvolver aplicações .NET (inclusive o ASP.NET) com os compiladores para C#, Visual Basic e Visual C++.

2.3 .NET Framework

A .NET Framework é uma nova plataforma que simplifica o desenvolvimento de aplicações no ambiente distribuído da internet. A .NET Framework foi desenhada para cumprir os seguintes objectivos:

- Disponibilizar um ambiente de programção orientada a objectos consistente, quer o código do objecto esteja guardado e executado localmente(mas distribuído pela internet) ou executado noutra máquina (remotamente)
- Disponibilizar um ambiente de execução de código que miniza a distribuição de software e o conflito de versões
- Disponibilizar um ambiente de execução de código que garanta uma execução segura de código, incluindo código criado por terceiros
- Disponibilizar um ambiente de execução de código que elimine os problemas de performance de ambientes interpretados
- Fazer com que o programador experiencie extensamente vários tipos de aplicações, como aplicações Windows e aplicações Web

A Framework .NET tem duas componentes principais: a *common language runtime* e a livraria de classes Framework .NET. A *common language runtime* é a base da Framework .NET. Pode-se considerar o *runtime* como um agente que controla o código durante o tempo de execução, fornecendo serviços *core*, tais como gestão de memória, threads e acessos remotos, ao mesmo tempo que reforça a a precisão do código, assegurando segurança e robustês. Aliás, o conceito de controlo de código é um principio fundamental do *runtime*.
A livraria de classes é uma colecção orientada a objectos de tipos reutilizáveis que podem ser usados no desenvolvimentos de aplicações desde simples aplicações de linha de comandos até aplicações baseadas nas mais recentes inovações fornecidas pela *ASP.NET*, tais como *Web Forms* e *XML Web Services*.

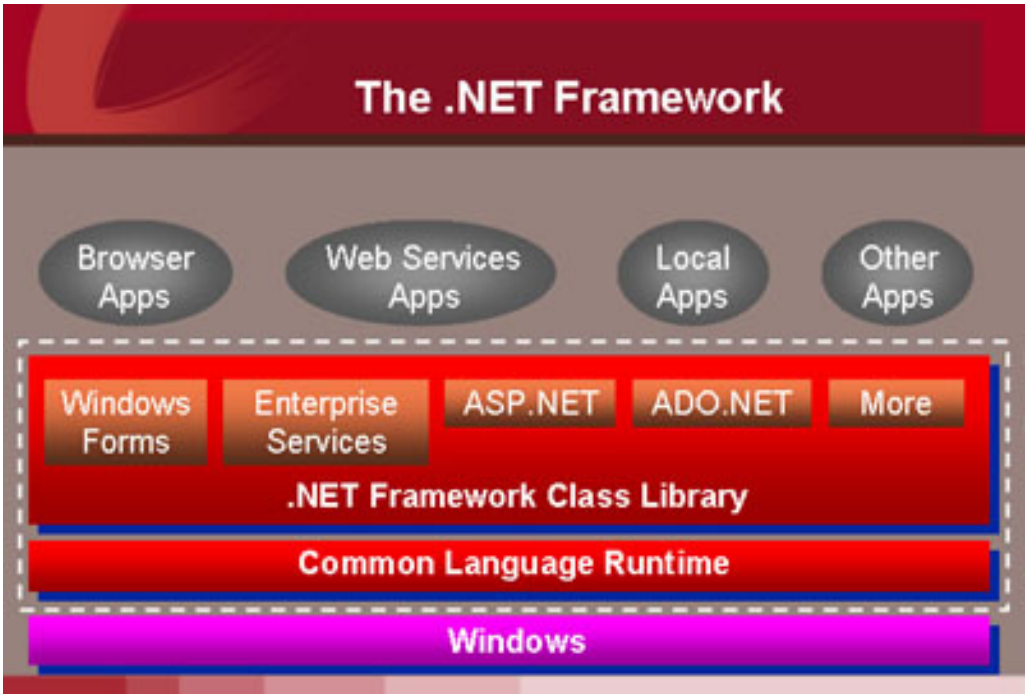

Figura 1: .NET Framework

2.4 Web Services

O conceito de Web Services tem um papel central na plataforma .NET. Estes serviços formam a base de uma visão de computação distribuída, descentralizada, situada na Web e em que diferentes dispositivos e aplicações colaboram na implementação de funcionalidades. Um webservice só se preocupa em fazer pedidos e satisfazer pedidos. O aspecto da comunicação, através de XML, deixa de ser uma preocupação para o programador.

2.5 Common Language Runtime

O Common Language Runtime (CLR) é o núcleo da plataforma .NET. A CLR oferece inúmeros benefícios aos programadores, como gerador de excepções, segurança, depuração e indicação de versão - e esses benefícios estão disponíveis para qualquer linguagem construída para a CLR. Isso significa que a CLR pode funcionar com uma grande variedade de linguagens e que pode também oferecer um kit de ferramentas único para todas elas.

3 Interface web

Uma interface tem que se reger por certos principios básicos, que ao serem cumpridos, permitem ao utilizador uma utilização rápida, com uma menor taxa de erros e com tempo de aprendizagem reduzido. uma interface tem que utilizar termos e conceitos com os quais os utilizadores estejam habituados, para que estes se possam adaptar rápidamente ao sistema e para que este não seja estranho aos seus hábitos de trabalho. o interface tem que ser consistente (menus, design grafico, comandos), para que em diferentes operações o utilizador não estranhe ou interprete mal o sistema, e para que todas as operações se processem da mesma maneira. as respostas do sistema devem ser similares, para que o utilizador nao seja surpreendido e tenha alguma reacção anomala. O sistema deve evitar ao maximo que o utilizador cometa erros, utilizando mecanismos tais como a confirmação de opções.

É necessario também que seja fornecida ajuda ao utilizador, para que interprete e utilize devidamente os campos e as opções a preencher. As mensagens de erro devem ser claras e esclarecer naquilo que o utilizador errou e como deve proceder. A interacção com o sistema deve estar adequada ao actor que esta a utilizar. Vendo estes principios pela prespectiva de um interface web, há aguns reparos a fazer.

no que diz respeito à prevenção de erros pode-se fazer uma solução baseada em confirmação de opções bastante insistente, avisando o utilizador do que uma eventual combinação de acções pode ter reflexos irreversíveis para o sistema. No que diz respeito à ajuda, esta pode ser feita ou através de tooltips (uma pequena caixa que aparece ao passar com o rato por cima de um determinado item, com um texto explicativo do que este item faz), ou de ajuda on-line através do browser. as mensagens de erro podem ser disponiblizadas em integração com a ajuda, o que facilita uma proxima utilização do serviço pelo utilizador.

4 Interface para outras aplicações

Apesar de parecer exagero, hoje em dia, qualquer engenheiro de software ao pensar numa aplicação, deve pensar em fazê-la como *Web Service*, mas porquê?

É evidente que nenhuma empresa existe isolada. Todas têm obrigatoriamente relações comerciais e este relacionamento é o motivo da existência das mesmas. Então, devemos considerar que é mais do que desejável que se criem mecanismos para que isto aconteça da melhor forma possível, com velocidade e segurança. Boa parte desta tarefa pode ser feita através de *Web Services*.

Vamos pensar talvez num caso simples: gestão de *stocks*. Se utilizarmos *Web Services* para este trabalho, podemos obter desde um simples "controlo de *sotcks*"para o frigorífico lá de casa, até grandes armazens revendedores. Pode controlar-se o *sotck* dos alimentos dum frigorífico, geridos automaticamente através do código de barras (no caso de alimentos sem código de barras, é necessário que se digite o código), e o *Web Service* instalado no frigorífico, equipado com Windows CE, leitor de códigos de barras e com uma ligação à *Internet*, solicita automaticamente ao supermercado os alimentos que estão com o *sotck* baixo. Esta tecnologia podia ser facilmente extendida e melhorada por forma a ser aplicada em cafés, bares, restaurantes, hotéis, etc...

Quando pensamos em comunicar com qualquer tipo de empresa, temos necessidade de pensar numa forma de trabalhar com um grande número de plataformas. Para cada plataforma era normal desenvolver-se um protocolo de comunicação (geralmente de naturaza binária). Como resultado, aplicações que trabalham entre plataformas, normalmente compartilham apenas dados. Com o reconhecimento dessas limitações, houve um "empurrão"para formação de um formato padrão para troca de dados. Esse "empurrão"impulsiona-nos para uma visão que rapidamente nos envolve num novo paradigma da computação: a "seamless". Uma rede integrada de serviços além das barreiras tradicionais de hardware e software. No coração dessa visão está o conceito de operabilidade conjunta, ou seja, a capacidade de sistemas diferentes comunicarem, sem estarem ligados entre si.

A única maneira para tratar o número enorme de entidades heterogêneas na *Internet* é usar o denominador comum mais baixo. Ou seja, quando informações são transferidos de um site para outro, o processo necessita utilizar algum padrão que todos na *Internet*, suportem. O protocolo mais comum de transferência na *WEB* é o *HTTP*. O padrão entre plataformas para codificarmos uma informação e transferi-la por *HTTP* é o *XML*. A *Microsoft* uniu

estas ideias e desenvolveu o conceito de WebService - uma maneira de objetos num servidor aceitarem pedidos dos clientes usando o HTTP e o XML.

Um *Web Service* é uma aplicação lógica, programável, acessível, que usa os protocolos padrão da *Internet*, para que se torne possível a comunicação transparente de máquina-para-máquina e aplicação-para-aplicação.

Para desenvolvermos um *Web Service* utilizamos tecnologias pensadas sob esta visão, como o *SOAP (Simple Object Access Protocol)*, *WSDL (Web Service Description Language)*, e *HTTP (HyperText Transfer Protocol)*, usadas para passar mensagens através das máquinas. Estas mensagens podem variar muito na complexidade, desde a chamada de um método, à submissão de uma ordem de compra.

4.1 SOAP

O *SOAP* fornece-nos um mecanismo simples e leve para trocar informação estruturada entre máquinas num ambiente distribuído e descentralizado via *XML*. O *SOAP* por si só não define nunhuma semântica de aplicação, tal como um modelo de programação ou uma semântica específica de implementação, nem define um mecanismo simples de expressar a semântica de uma aplicação fornecendo um modelo de *package* modular e mecanismos de codificação para codificar dados entre modulos. Estas características permitem que o *SOAP* seja utilizado numa grande variedade de sistemas desde sistemas de mensagens a *RPC*.

O *SOAP* consiste em três partes:

- O envelope *SOAP* define uma framework para exprimir o que está na mensagem; quem deve tratar dela e se é opcional ou obrigatória
- As regras de codificação so *SOAP* definem um mecanismo de serialização que pode ser usado para trocar instancias de tipos de dados definidos em aplicações
- A representação *RPC SOAP* define uma convenção que pode ser usada na representação de ”chamamentos”e ”respostas”remotos a procedimentos

Apesar de estas partes estarem descritas juntas como parte do *SOAP*, elas são funcionalmente ortogonais. Em particular, o envelope e as regras de codificação são definidas em *namespaces* diferentes de forma a promoverem simplicidade através da modularidade.

Além destas três partes, esta especificação define dois protocolos de ligação que descrevem como uma mensagem *SOAP* pode ser transportada em mensagens *HTTP*, quer com ou sem a extenção *Framework HTTP*.

4.2 WSDL

Enquanto os protocolos de comunicação e os formatos da mensagem são standartizados na comunidade web, torna-se cada vez mais possível e importante poder descrever as comunicações de uma forma estruturada. O *WSDL*[1] dirige-se a esta necessidade definindo uma gramática de XML para descrever *Web Services* como colecções de comunicações *endpoints* capazes de trocar mensagens. As definições do serviço de *WSDL* fornecem a documentação para sistemas distribuídos e servem como uma receita para automatizar os detalhes envolvidos nas comunicação das aplicações.

Um documento *WSDL* define serviços como coleções de *endpoints* de rede, ou portos. A gramática *WSDL* usa os seguintes elementos em conjunto para descrever *endpoints*

- Tipos - um *container* para definições do tipo de dados usando algum sistema do tipo (tal como XSD).
- Mensagem - uma definição abstracta dos dados que estão a ser comunicados.
- Operação - uma descrição abstrata de uma acção suportada pelo serviço.
- Tipo de porto - um conjunto abstrato das operações suportadas por um ou mais *endpoints*.
- Ligação - uma especificação concreta do formato do protocolo e de dados para um tipo de porto particular.
- Porto - um único *endpoint* definido como uma combinação de uma ligação e de um endereço de rede.
- Serviço - uma coleção de *endpoint* relacionados.

É importante observar que o *WSDL* não introduz uma nova linguagem. O *WSDL* reconhece a necessidade para sistemas ricos para descrever formatos da mensagem, e suporta a especificação dos schemas de XML (XSD) como seu sistema canônico. No entanto, já que é insensato esperar que uma única gramática do sistema seja usada para descrever todos os formatos de uma mensagem, o *WSDL* permite utilizar outras linguagens de definição de tipos.

Adicionalmente, o *WSDL* define um mecanismo comum de ligação. Isto é usado para ligar um protocolo específico ou formato de dados ou estrutura de dados a uma mensagem abstracta, operação ou *endpoint*. Permite também a reutilização de definições abstractas.

[1] Web Services Description Language

Além da *core service definition framework*, esta especificação introduz extencões especificas de ligação para os seguintes protocolos e formatos de mensagens:

- SOAP 1.1
- HTTP GET / POST
- MIME

5 Arquitectura de camadas

5.1 Camada de interface

Esta camada estabelece a ligação, atráves de um cliente *Web*, entre o uitilizador e o sistema. Este interface é desenvolvido utilizando as linguagens de construção de interfaces *Web*. Entre essas linguagens destacam-se o *HTML*, o *CSS* e o *JavaScript*.

O *HTML (HyperText Markup Language)* é uma linguagem universal destinada à elaboração de páginas com hiper-texto, como o nome indica. O conceito de hiper-texto é bastante simples: Certos itens de um documento contêm uma ligação a outra zona do mesmo documento ou, como é mais vulgar, a outros documentos.
A principal aplicação do *HTML* é a criação de interfaces na Web, e convém esclarecer que não se trata de uma linguagem de programação. De facto, o *HTML* é antes uma espécie de linguagem de formatação, um ficheiro de texto que é formatado através de uma série de comandos.

Cascading Style Sheets (CSS) ou Folhas de Estilo, é uma nova tecnologia (apareceu em 1996), padronizada pelo *World Wide Web Consortium* (a entidade que define os padrões da web), mas não faz parte do *HTML* padrão, é sim um conjunto de novas tags que ajudam a controlar e uniformizar/padronizar o aspecto (fontes, cores, tamanhos, etc) das páginas *HTML*.

O *Javascript* é uma linguagem de script que incorporado nos tag's *HTML*, permite incrementar a apresentação e interactividade das páginas *Web*. Este *Javascript* é então uma extensão do código *HTML* das páginas *Web*. Os scripts, que se inserem nos tag's *HTML*, podem ser comparados aos macros de uma formatação de texto. Estes scripts vão ser gerados e executados pelo próprio browser sem fazer apelo aos recursos de um servidor. Estas instruções serão assim executadas directamente e sobretudo sem atrasos.
Javascript foi desenvolvido inicialmente pela *Netscape* e na altura intitulava-se *LiveScript*. Adoptado no fim do ano de 1995, pela firma *Sun* (que também desenvolveu o *Java*), ele tomou assim o seu nome actual *Javascript*.
Javascript não é próprio do browser *Netscape*. A *Microsoft* também adoptou o *Javascript* desde do seu *Internet Explorer 3*. E melhorou significativamente

no *Explorer 4*.
As versões do *Javascript* sucederam-se com as diferentes versões do *Netscape*: *Javascript* para *Netscape 2*, *Javascript 1.1* para *Netscape 3* e *Javascript 1.2* para *Netscape 4*. Apesar de existir alguns problemas de compatibilidade, segundo o browser utilizado nas página que contém codificação *Javascript*.
O futuro do *Javascript* está entre as mãos dos dois grandes browsers da *Web* e em parte ligada a guerra entre a *Microsoft* e a *Netscape*. Podemos mesmo assim prever um futuro promissor a esta linguagem sobretudo pela sua independência em relação ao servidor.

5.2 Camada de Controlo

O interface, se existisse isolado, seria extremamente estático (pois é desenvolvido recorrendo apenas às "linguagens"referidas no tópico anterior), e "estático", na Internet de hoje em dia é uma palavra quase proibida. Como no interface por si só, não é possivel comunicar com o servidor aplicacional é necessário recorrer a uma nova camada, a camada de controlo.

Assim sendo, a camada de controlo (desenvolvida usando a tecnologia *ASP.NET*) é responsável por comunicar com o servidor aplicacional e gerar o interface de uma forma dinâmica.

ASP.NET O *ASP*[2]*.NET* é um conjunto de tecnologias disponíveis na *Microsoft .NET Framework* para construir aplicações *WEB* e *XML Web-Services*. As páginas desnvolvidas em *ASP.NET* são executadas no servidor e enviam para o cliente um resultado em *HTML*, *XML*, etc. Desta forma é possível a separação entre a lógica de negócio e o interface para o utilizador.

O *ASP.NET* é muito mais que uma versão mais recente do *ASP*; é sim uma plataforma de desenvolvimento que fornece os serviços necessários para o desenvolvimento de aplicações em larga escala. Apesar de ser, em termos de sintaxe de linguagem, extremamente compatível com o ASP tradicional, fornece ao programador um novo modelo de programação e as infra-estruturas necessárias para o desenvolvimento de aplicações mais estáveis e seguras.

Como o ASP.NET foi desenvolvido para a .NET Framework, é possivel desenvolver as apliacações numa série de linguagens de programação (Visual Basic .NET, C#, JScript .NET, etc).

[2]Active Server Pages

5.3 Camada do servidor aplicacional

Esta é a camada que implementa toda a lógica de negócio, isto é, comunica com a camada de persistência para recolher os dados, faz o tratamento dos mesmos de acordo com a lógica de negócio especificada e comunica o resultado das operações à camada de controlo para que seja visualizado no interface.

Como estamos a utilizar a plataforma *.NET*, o desenvolvimento (ou implementação da lógica de negócio) pode ser feito com recurso a diferentes linguagens de programação. Entre elas queremos destacar o C# e o Visual Basic .NET.

O C# (pernunciando-se "C sharp") é a evolução do C e C++, já que estas linguagens, apesar de serem bastante flexíveis, obrigavam a um grande esforço de produção, já que o desenvolvimento de software nestas linguagens era bastante demorado, graças ao controlo que o programador possui nestas linguagens. o C# é uma linguagem orientada a objectos, que permite que o utilizador possa construir várias aplicações para a plataforma .NET rápidamente, desde o desenvolvimento de aplicações para negócios até aplicações a nível de sistema. ao utilizar o C#, os componenetes construidos por esta linguagem podem ser convertidos para *Web Services*, para que possam ser utilizados através da Internet, por qualquer linguagem em qualquer sistema operativo. Para além disso, há também a capacidade de poder fazer com que os *Web Services* sejam vistos como objectos em C# pelo programador, facilitando a integraçõa destes no sistema e facilitando o trabalho do porgramador.

Mais do que isso o C# foi desenhado a pensar nos programadores de C++, na adaptação destes à nova linguagem, sem sintam que perderam o poder e o controlo que são a marca registada do C e do C++. Devido a essa heança, o C# tem um grande grau de fidelidade para com o C e o C++. Alguns dos problemas do C++ são colmatados com algumas inovações do C# tais como o *Garbage collection* (que faz com que o programador deixe de se preocupar com a gestão de memória)e a inicialização automática das variáveis.

Para além disso e no que diz respeito aos ficheiros em XML, o C# permite que esses ficheiros - normalmente pequenos - sejam mapeados directamente na estrutura de dados, em vez de construir uma classe para estes ficheiros, sendo esta uma maneira mais eficiente de tratar pequenos pedaços de informação.

O C# permite que meta-dados sejam aplicados como objectos ou seja, o arquitecto de um projecto pode definir certos atributos especificos ao dominio, e aplica-los a classes, interfaces, etc., para além de poder examinar estes atributos em cada elemento. isto facilita a identificação das classes ou interfaces perante as os correspondentes objectos, ou simplesmente criar relatórios baseados nos atributos dos objectos. A ligação estreita entre os meta-dados e o código ajuda a fortalecer a ligação entre o que o programador quer do programa e a implementação deste.

Com o C# todo o objecto é automáticamente do tipo COM, não sendo necessário implementar nos objectos este protocolo de segurança, para além de ter a hipótese de poder implemntar quaisquer APIs e dentro de espaços especificos, podem utilizar funcionalidades do C ou do C++, como gestão de memória ou apontadores o que representa a recuperação de código já construido nestas linguagens, em vez de o descartar completamente. Em ambos os casos - quer no suporte ao COM quer ás APIs- o objectivo é disponibilzar o poder do C/C++ sem sair do ambiente C#.

No que diz respeito ao Visual Basic .NET (VB) o que podemos dizer é que para além de o programador experiente em versões antigas de Visual Basic poder continuar a usar os seus conhecimentos nesta plataforma, existem todas as vantagens que o C# fornece, tais como a possibilidade de utilizar meta-dados, a conversão de módulos em XML, a compatibilidade com o modelo COM (e eleminando os conflitos de registo destes modelos e a sobreescrita de DLLs), tem ainda outras inovações como por exemplo um grupo de ferramentas de desenvolvimento de aplicações rápido (RAD), que incluem a possibilidade de fazer um design em drag-and-drop e outros controlos visuais.

O XCOPY do VB.NET permite que a colocação das aplicações (feitas para Windows) nos clientes possa ser feita copiando os ficheiros directamente para

o directório da aplicação desejada. Para além disso, o *download* automático dessas aplicações para Windows faz com que a colocação de aplicações construidas para Windows seja tão fácil como abrir uma página Web.

5.4 Camada de persistência

Esta camada faz a ligação do servidor aplicacional com a base de dados (implementada em *Oracle*). A plataforma .NET usa a tecnologia *ADO.NET* para comunicar com a base de dados.

O *ADO.NET* é uma evolução do *Microsoft*® *ActiveX*® *Data Objects (ADO)* que fornece uma interoperabilidade entre as várias plataformas do *.NET* e um acesso aos dados escalonado. Utilizando *XML*, o *ADO.NET* assegura que haja uma eficiente tranferencia de dados para qualquer plataforma ou aplicação.

Com o *Visual Studio .NET* trabalhamos com objectos em vez de tabelas e colunas. O *ADO.NET* utiliza uma programação por tipos, permitindo que de uma maneira fiável, se possa escrever codigo que aceda a uma base de dados. Para além disso, ao trabalhar com tipos torna-se mais fácil escrever código. Podemos *navegar* através da base de dados utilizando a tecnologia *IntelliSense*® para visualizar tabelas disponiveis. Adicionalmente, os tipos ajudam a melhorar o tempo de execução pois a aplicação não necessita de procurar colecção de objectos *ADO* de cada vez que é necessário aceder aos dados. O centro de qualquer software que utilize o ADO.NET é o *data set*. O data set é uma cópia residente em memória dos conteúdos da base de dados. um *data set* contem um dado numero de tabelas a que cada uma corresponde à tabela ou vista existente na base de dados. O *data set* constroi uma vista dos dados da base de dados existente. Esta arquitectura desconectada permite um maior escalonamento pois só usa o servidor da base de dados quando lê ou escreve na base de dados original.

Quando corre, os dados são ser transferidos da base de dados para uma camada intermédia e de seguida para interface do utilizador. para fazer a tranferencia, o *ADO.NET* utiliza o formato *XML*. para transmitir dados de uma camada para outra, uma solução que o *ADO.NET* utiliza é exprimir a camada residente em memória como *XML* e então envia o *XML* para o outro componente.

6 Adequação aos requisitos funcionais e não funcionais

No desenvolvimento de aplicações em larga escala, a escolha da tecnologia é de extrema importância. Se houver algum erro nessa escolha pode comprometer o desenvolvimento de todo o projecto. Tendo isso em conta, é de muito importante efectuar um estudo/análise de tecnologia para a decisão seja pensada e ponderada.

Depois de analisada toda a tecnologia que vamos usar para desenvolver o sistema, chegamos à conclusão que o *.NET* se adequa perfeitamente aos nossos requisitos (quer sejam funcionais ou não).

Para o desenvolvimento de sistemas modulares (como o siLEIC que foi separado em vários módulos distintos, a serem desenvolvidos por equipas diferentes usando diferentes linguagens de programação, mas que têm de interagir entre si através de Web-Services) a plataforma .NET adequa-se perfeitamente visto que têm um suporte (que integra/encapsula todos os sistemas de comunicação entre módulos) muito bom para desenvolvimento de aplicações-web/Web-Services.

Recorrendo ao uso de web-services, o acesso à base de dados é também uma tarefa muito simples de efectuar e uniformizar, já que nos permite criar um web-service que executa *query's SQL* e retorna dados na forma de *data-sets*. Isto traz grande vantagens ao desenvolvimento de todo o projecto pois além de nos permitir um acesso uniforme à base de dados, permite também fazer acessos remotos.

Apesar de se tratar da mesma aplicação (em que normalmente o acesso à base de dados é tratado como sendo um acesso local), se, desde o ínicio do desnvolvimento se tratar o acesso à base de dados como sendo um acesso remoto, permite-nos, a qualquer momento transferir a base de dados para outro servidor. É bastante normal, por questões de segurança e fiabilidade, quando uma base de dados atinge dimensões muito grandes, proceder-se à sua transferência para um servidor (mais potente) separado do servidor da aplicação.

No que diz respeito ao uso das linguagens de programação, uma das maiores apostas da Microsoft foi desenvolver uma plataforma que possa tirar partido das potecialidades das diversas linguagens de programação que existem. Como o sistema de informação é desenvolvido em vários módulos distintos, cada

módulo (e até mesmo as várias partes de cada módulo) pode ser desenvolvido na linguagem que melhor se adeque.