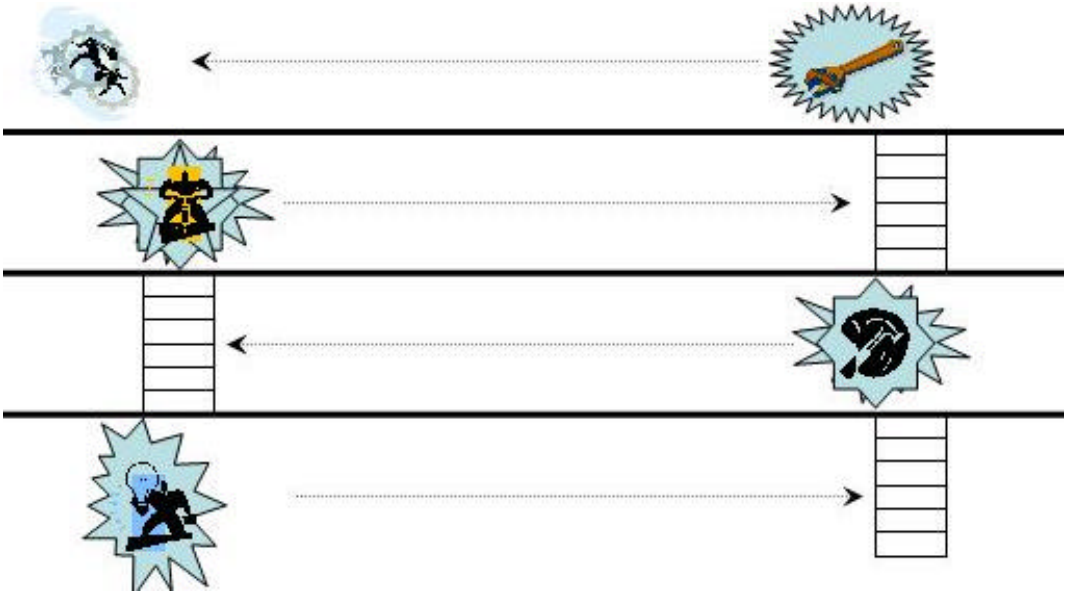
Instituto Superior de Engenharia do Porto

Licenciatura em Engenharia Informática



Microsoft .net Framework SDK

Microsoft Visual Studio .NET

"Suporte do .Net para a construção de aplicações N-Tier"

Romeu Tiago de Campos Faria

Índice

Introdução

Actualmente as necessidades na construção de software evoluíram para um ponto onde é necessário lidar com computação distribuída, processamento distribuído, transacções em bases de dados, estabelecimento de regras de negócio, controlo de acessos e distribuição de informação, e muitas outras situações em que há necessidade de dividir toda a aplicação num conjunto de níveis que facilitem a implementação e aumentem a eficácia dessa aplicação.

Uma forma de fazer frente a todas estas exigências é o recurso ao desenho de aplicações em camadas – tier – e, mediante as necessidades e especificidades das aplicações, implementar em cada uma das camadas uma parte das regras de negócio da mesma aplicação. Com esta divisão em camadas, podemos estabelecer, para cada camada uma função específica que será definida pelo local onde estará a camada (local físico, e local lógico – cliente, servidor), pelas funções que deverá desempenhar e os tipos de dados que pode tratar.

A Microsoft lançou o Visual Studio .Net e a Framework .Net que nos fornece uma vasta gama de serviços e tecnologias que facilitam a construção deste tipo de aplicações. Estas tecnologias vão desde o suporte para ligação a diversos sistemas de gestão de bases de dados, novos tipos de dados e novas formas de troca desses mesmos dados, definição de enterprise templates que permitem que quem desenhe o sistema, defina de imediato o que pode ou não ser construído em cada uma das camadas em construção.

É necessário salientar outro factor de elevada importância. Trata-se de uma grande inovação – o Commom Language Runtime – que permite, tendo em conta toda esta divisão de uma aplicação em camadas, que cada camada seja desenvolvida por programadores diferentes e na linguagem em que cada um se sinta mais à vontade.

Tendo em conta os factores apresentados anteriormente, é de grande interesse mostrar o suporte que o Visual Studio .Net, em conjunto com a Framework .Net, fornece para a construção de aplicações N-Tier, tendo em conta as especificidades das aplicações em questão, bem como as tecnologias disponibilizadas pela ferramenta da Microsoft.

Objectivo

O objectivo deste trabalho é mostrar o suporte fornecido pela Framework .Net e pelo Visual Studio .Net na construção de aplicações que fazem uso do padrão N-Tier (divisão por camadas), focando tecnologias emergentes que permitem melhorar de forma eficiente e simples a construção de aplicações.

No decorrer deste trabalho vou pois demonstrar algumas características das aplicações n-tier, vantagens e desvantagens, e mostrar algumas aplicações de técnicas revolucionárias que suportam este tipo de aplicações, algumas características importantes do .Net, e dar exemplos de aplicação, tudo isto tendo em conta sempre a tecnologia e o modelo de camadas da Microsoft.

Para que se possa comprovar a eficácia da utilização do .Net, vou demonstrar com relativa simplicidade a forma de construir uma aplicação com quatro camadas que explicarei mais adiante.

A denominação .Net será usada frequentemente querendo significar o conjunto da Framework .Net e do Visual Studio .Net-

Requisitos

Para a correcta execução e visualização dos exemplos elaborados, é necessário que se respeitem os seguintes requisitos:

- ✓ Windows XP (Desenvolvido em Windows XP Professional em português).
- ✓ Framework .Net SDK.
- ✓ Microsoft Internet Explorer 5.5 ou posterior.
- ✓ Internet Information Services (IIS).
- ✓ Microsoft Visual Studio .Net Enterprise Arquitect (esta versão é a que me parece mais indicada para a construção da estrutura de uma aplicação).

A visualização das soluções criadas para exemplo, deve ser feita abrindo com o Visual Studio .Net o ficheiro que corresponde à solução que se encontra no directório do exemplo referente. Por exemplo, para abrir o exemplo *licExemplo01*, selecciona-se o menu File – Open Solution – *<licExemplo01>* e selecciona-se o ficheiro *licExemplo01.sln*.

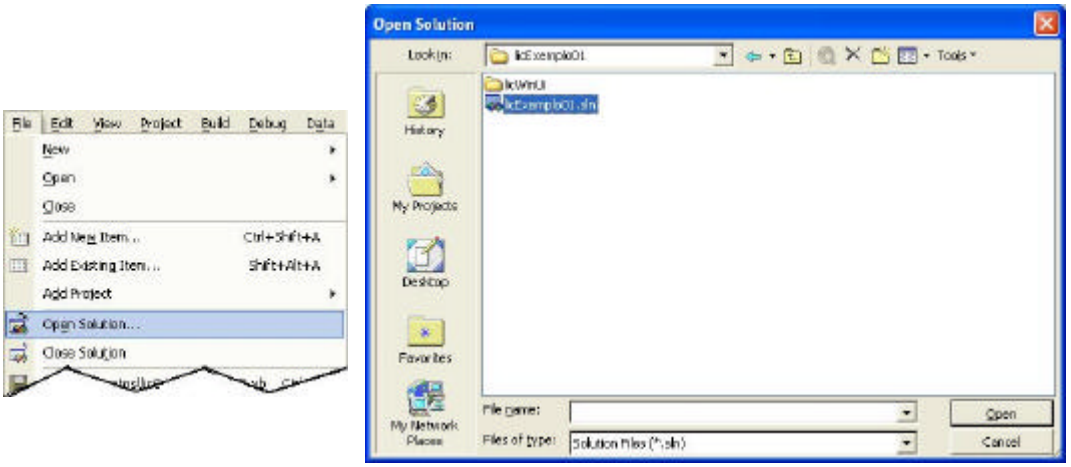


Figura I : Abrir uma solução

Os exemplos disponibilizados assentam sobre uma base de dados elaborada em Microsoft Access, pelo que para a correcta execução, deve-se alterar a connection sting do objecto Connection do ADO.Net, indicando o directório no qual ela se encontra colocada. Para tal, basta procurar no projecto DataAccess os objectos do tipo Component que são identificáveis pela denominação *licNomeClasse_CMP.vb* que se encontram no Solution Explorer. Depois de encontrar o componente, no objecto Connection, basta alterar a connection string, colocando a path correcta referente à localização do ficheiro da base de dados.

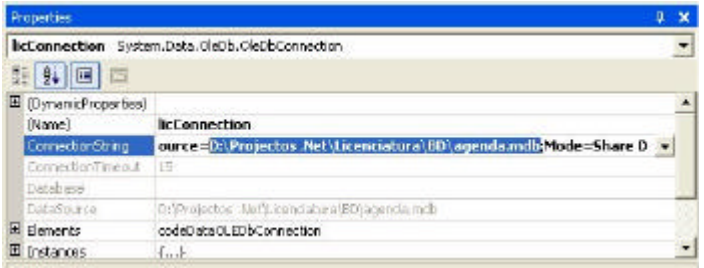


Figura II : Como alterar a Connection String

Pressupostos

Para o melhor entendimento do trabalho, deve possuir conhecimento de:

- ✓ Conhecimento básico da Framework .Net
- ✓ Conhecimento básico do Visual Studio .Net
- ✓ Conhecimento básico de Sistemas de gestão de bases de dados

N-Tier

Conceito

O termo N-Tier significa número de camadas, em que N especifica o número de camadas. Cada uma das camadas representa um nível que se relaciona directamente apenas com os níveis/camadas que lhe são adjacentes, e que tem tarefas distintas e pré-definidas, e que se destinam a servir as outras camadas.

As aplicações n-tier, integradas em sistemas n-tier, permitem a utilização dinâmica dos recursos de hardware e software computacional. Uma aplicação desenvolvida em camadas (tiers) permite que cada uma das camadas seja desenvolvida, controlada e gerida de forma independente.

História

A arquitectura N-Tier nasce do paradigma cliente/servidor. Nas passadas duas décadas, este modelo tem sido usado na construção de grande parte das aplicações existentes. Podemos definir este modelo (cliente/servidor) como uma aplicação N-Tier com duas camadas. Neste caso o cliente comunica directamente com o servidor (em alguns casos sistemas gestores de bases de dados). Toda a lógica de negócio está presente no cliente ou então definida em stored procedures na base de dados.

A primeira geração de aplicações deste género tinha como componente principal os "clientes pesados" (fat client). É nestes que se realiza a maior parte do processamento. Nesta altura (anos oitenta e início dos anos noventa), os GUI (Graphical User Interface) eram o ambiente dominante. Com o aumento da complexidade da aplicação, os clientes iam ficando cada vez mais "pesados", tal era a necessidade de processamento. O custo de construção e manutenção de redes (LAN) baseadas neste tipo de arquitectura (fat client) era elevadíssimo já que em termos de comunicação era necessário enviar toda a informação para o cliente e este depois faria o tratamento. No caso dos postos de trabalho o custo ainda seria superior, pois as tarefas realizadas pelo cliente exigiam máquinas de elevada capacidade de processamento e portanto de custo superior.

Numa segunda fase, tentou-se construir um modelo alternativo em que o processamento passaria para o servidor, chamado "servidor pesado" (fat server). Nestes servidores eram colocados procedimentos com regras de negócio, e os clientes apenas faziam pedidos.

Apesar de mais eficiente, e capaz de atingir performances elevadas, este modelo sobrecarrega o servidor com todo o processamento, o que pode causar estados de espera por parte dos clientes.

Outro problema desta técnica era que as stored procedures (que implementam regras de negócio) estão inseridas na base de dados, causando sobrecarga de processamento no

servidor. Outro problema era que quando se pretendia mudar regras de negócio, por exemplo a pedido de um cliente, podia acontecer o caso de ter de mudar a base de dados, o que nem sempre é fácil pois as aplicações clientes não seriam todas iguais e uma alteração poderia causar perdas.

No início dos anos noventa, surge uma nova geração de implementações N-Tier. A arquitectura em três camadas (3-tier). Nesta arquitectura aproveita-se a tecnologia do lado do cliente para fazer uma camada de apresentação (Interface – "thin client"). Uma segunda camada passa a residir num servidor de aplicações que implementa as regras de negócio, e a terceira camada, trata do acesso a dados. Nesta altura nasce o conceito de middle-tier.

Neste ponto, alterações na camada de apresentação e na camada de regras de negócio passam a ser independentes, e as aplicações passam a permitir uma evolução mais fácil.

"Thin is in" – foi este o mote para o desenvolvimento de aplicações em camadas com reduzida carga de processamento, e baixa necessidade em termos volume de comunicação.

Temos que ponderar qual a melhor solução no desenvolvimento de uma aplicação.

O uso generalizado da técnica N-Tier está a ser acompanhado por constantes mudanças a nível de necessidades de negócio e adaptação às novas tecnologias, e é normalmente possível que uma aplicação seja estruturada em várias segmentações com duas, três ou mais camadas, conforme as necessidades e especificidade do cliente.

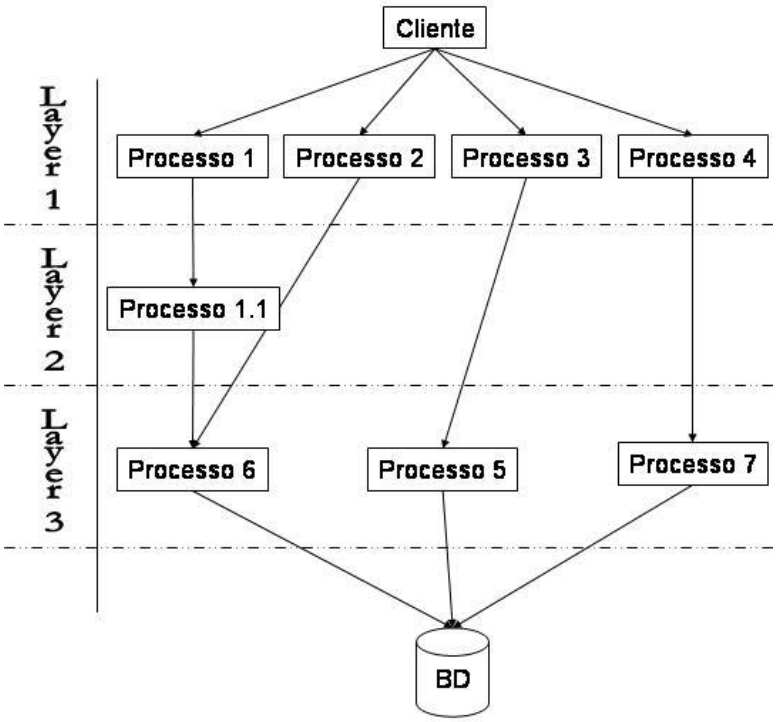


Figura III: Estrutura de uma aplicação

A tecnologia

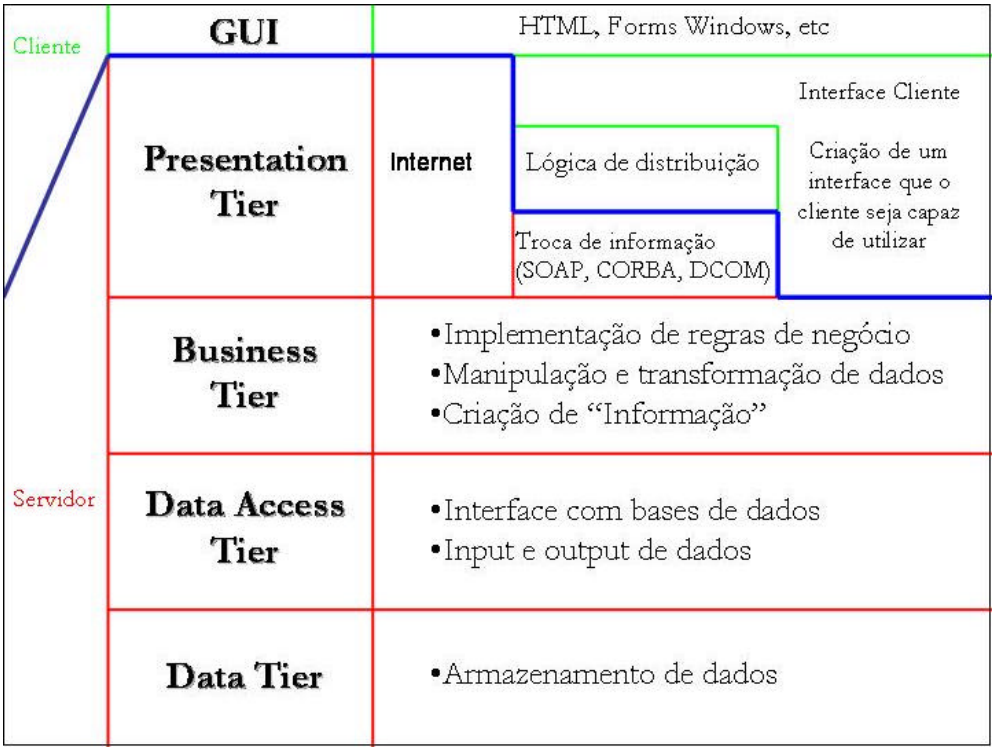


Figura IV : Arquitectura N-Tier Simplificada

Na prática

Para o fim a que se destina este trabalho, vou definir aplicação N-Tier, como um sistema que tem requisitos de performance, escalonabilidade e confiança elevados. A aplicação deve ser dividida em camadas lógicas com regras de negócio definidas pelas organizações.

O conjunto das camadas pode estar localizado numa única máquina, mas tem a grande vantagem de poder ser distribuído por várias máquinas ou mesmo redes diferentes.

A performance, a escalonabilidade e a confiança são actualmente conceitos muito importantes principalmente no domínio do comércio electrónico. Deseja-se que uma aplicação seja capaz de efectuar todas as funções para que foi criada, evitando problemas e falhas de sistema, mas tendo em conta o grau de importância e exigência. É importante que se saiba sacrificar alguns destes factores em função de outros mais importantes. Por exemplo, é normal sacrificar a performance quando o sistema é altamente escalonável e requer níveis de segurança elevados. É impossível encontrar soluções óptimas, pelo que todos estes factores devem ser estudados e tidos em conta na construção destas aplicações.

Porquê dividir uma aplicação?

Primeiro porque podemos querer restringir acesso a certas partes da aplicação, e porque podemos ter necessidade de aumentar a eficiência, colocando a carga de processamento em várias máquinas, para que não estejamos apenas a sacrificar os recursos de uma delas.

Um outro motivo para dividir uma aplicação é o acesso a informação. Se uma aplicação não usa o conceito de camadas, todo o código estará concentrado num mesmo local e será executado na mesma máquina. Este facto permite que o cliente, através de processos de reverse-engeneering possa aceder ao código fonte e saber como e onde ir buscar informação a bases de dados ou mesmo a ficheiros proprietários.

Uma solução simples, seria colocar toda esta informação e detalhes da aplicação num local de acesso restrito, aumentando assim o nível de segurança. Deste modo o cliente apenas efectua pedidos e recebe respostas, e a aplicação, em termos de regras de negócio e dados de acesso a bases de dados não estarão facilmente ao seu alcance.

A arquitectura baseada em duas camadas (2-tier), define que uma camada Gui/Windows trata toda a parte da apresentação, enquanto que o servidor acede aos dados da base de dados. O problema é que, com a diversidade de aplicações cliente, e consequentemente das tecnologias que usam, torna-se necessário desenvolver inúmeras aplicações, uma para cada tipo de aplicação cliente. Isto significa também que cada pequena alteração iria resultar na necessidade de reescrever a totalidade da aplicação, o que poderia também afectar o funcionamento dos restantes clientes.

Com a divisão em três ou mais camadas, é possível mudar uma parte da aplicação sem afectar as restantes, e uma vez que as regras de negócio encontram-se numa camada isolada, a sua alteração não iria afectar a totalidade da aplicação (tendo em conta que esta será possivelmente a camada que será objecto de maiores alterações no processo de vida de uma aplicação).

Quais os sistemas que mais beneficiam com o N-Tier?

A generalidade dos sistemas que possam ser integrados dentro do paradigma cliente/servidor. A divisão em camadas permite acesso seguro, consistente e fiável a dados críticos.

Qual o significado que poderemos atribuir ao “N” em N-Tier?

A utilização dinâmica de recursos computacionais, o aumento dos níveis de segurança, a possibilidade de desenvolvimento de camadas por diferentes equipas de programadores, os níveis de performance elevados devido à divisão da carga de processamento, a maior facilidade na gestão do código e a base tecnológica para permitir suporte multi-plataforma, dão um interesse extra aos sistemas N-Tier.

Tendo em conta o grande número de vantagens que o uso desta tecnologia permite, podemos definir N como o número de:

- ✓ Camadas
- ✓ Clientes
- ✓ Objectos
- ✓ Servidores / Serviços
- ✓ Configurações
- ✓ Transacções
- ✓ Capacidades
- ✓ Benefícios
- ✓ Vantagens
- ✓ Oportunidades

As arquitecturas standard

Uma camada

É um pouco complicado tentar construir uma aplicação com alguma complexidade numa única camada. O simples facto de existir uma interface gráfica faz com que se torne complicado não fazer qualquer separação. Uma aplicação de dimensão considerável torna-se praticamente impossível de gerir com uma única layer, e uma tarefa tão simples como encontrar, e alterar uma parte de código que faz uma determinada operação pode tornar-se uma dor de cabeça, já que toda a aplicação se encontra concentrada na mesma camada.

Ainda assim, em pequenas aplicações podemos usar a arquitectura com esta configuração, chamada 1-tier.

Neste tipo de aplicações, o tráfego gerado para a rede é mínimo ou inexistente pois todos os recursos se encontram centralizados na mesma máquina.

Para melhor entender o funcionamento, podemos observar o seguinte caso que clarifica o funcionamento descrito anteriormente:

- A aplicação é instalada num computador. Este computador contém a base de dados com os stored procedures implementados pela aplicação e a própria aplicação. Toda a aplicação executa no mesmo espaço de memória e não há separação lógica entre aplicação e base de dados.

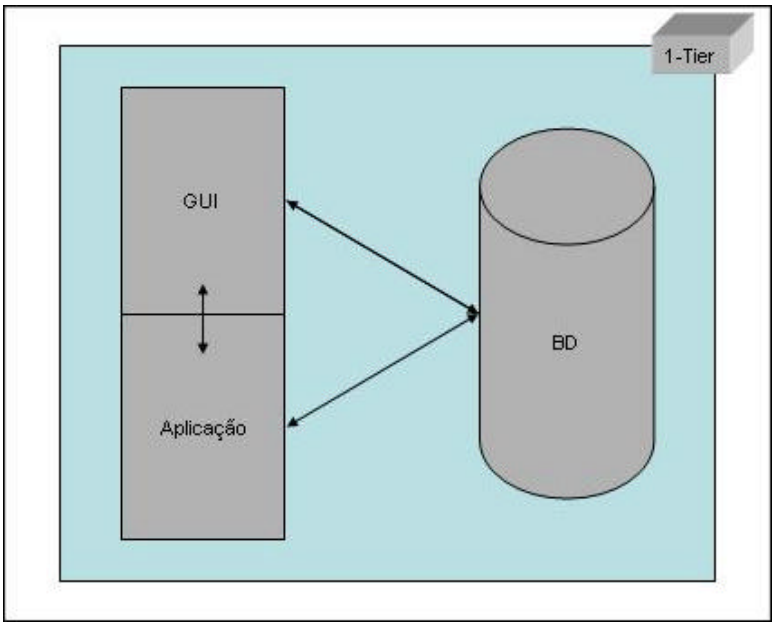


Figura V : A estrutura 1-Tier

Duas camadas

A evolução dos sistemas computacionais e das necessidades das aplicações levam-nos à divisão de uma aplicação em duas camadas, com a separação do programa e da base de dados. Neste ponto, é de grande interesse inserir stored procedures nas bases de dados, de modo a registar algumas das regras de negócio, tirando um pouco da carga de processamento da aplicação, passando-a portanto para o servidor.

Ainda que passem a existir duas camadas, a interface e as regras de negócio encontram-se juntas. No entanto a separação da base de dados permite avanços importantes no que diz respeito a capacidade de acesso (número de utilizadores ligados, que na estrutura anterior não permitia).

No entanto, este aumento da capacidade de serviço, causa o aumento do tráfego nas redes que servem de canal de comunicação à aplicação, uma vez que na maior parte dos casos é na aplicação/interface que grande parte do tratamento tem lugar, e são enviados blocos com toda a informação relativa ao pedido.

Há apenas alguns anos, esta era a estrutura com maior índice de utilização, porque apesar da sobrecarga de tráfego, trata-se de uma arquitectura fiável e relativamente robusta, e em termos de custos numa forma mais básica podemos apenas pensar numa máquina com a base de dados e depois clientes que se ligam a ela, portanto a gestão era simples.

Para melhor entender o funcionamento, podemos observar os seguintes casos, que clarificam o funcionamento descrito anteriormente:

- Primeiro caso: a aplicação foi instalada num único computador. Este computador contém a base de dados, juntamente com os stored procedures implementados pela aplicação e a própria aplicação. Toda a aplicação executa no mesmo espaço de memória, mas o funcionamento continua a ser regido pela lógica da divisão das camadas.
- Segundo caso: um servidor contém a base de dados, juntamente com os stored procedures que implementam assim algumas das regras de negócio.

Os clientes ligam-se ao servidor, usando a aplicação e os mecanismos de acesso da mesma, fazendo pedidos e recebendo os resultados do servidor que contém a base de dados.

Neste caso é necessário aumentar o nível de protecção e segurança da aplicação, porque através de algum trabalho computacional é possível aceder não só aos dados da aplicação, bem como à forma de dar sentido a essa informação, aplicando os processos de reversed-engineering que sejam necessários.

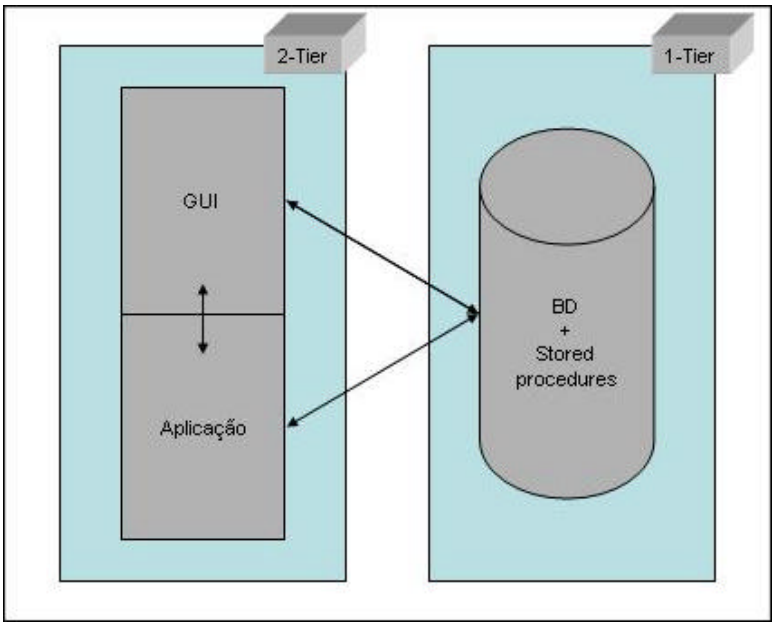


Figura VI : A estrutura 2-Tier

Três Camadas

Com a utilização desta arquitectura, passamos a ter uma camada de apresentação/interface, uma segunda camada com a lógica de negócio, e por fim, a terceira camada que será responsável pelo acesso à base de dados.

Apesar da divisão, a estrutura não é rígida, e no que diz respeito à base de dados é conveniente que esta continue a ter alguns stored procedures que implementem algumas das regras de negócio.

Esta configuração permite a utilização de diferentes interfaces, tendo como base as camadas de regras de negócio e acesso a dados, de forma simples e sem expor a vulnerabilidade que representam as regras de negócio, e os dados sensíveis na base de dados.

Cada camada comunica apenas com a camada mais próxima. Por questões de segurança, e coerência não deve ser possível que as camadas de interface e acesso a dados comuniquem directamente entre si.

A divisão lógica da aplicação permite que esta funcione da mesma forma numa única máquina, ou então dividida por várias máquinas. Em qualquer dos casos o funcionamento da aplicação é o mesmo.

Para melhor entender o funcionamento, podemos observar os seguintes casos, que clarificam o funcionamento descrito anteriormente:

- Primeiro caso: a aplicação foi instalada num único computador. Este computador contém a base de dados, juntamente com alguns stored procedures e a própria aplicação. Toda a aplicação executa no mesmo espaço de memória, mas em termos de funcionamento, a lógica das camadas funciona sempre da mesma forma.

- Segundo caso: um servidor contém a base de dados, juntamente com os stored procedures implementados pela aplicação. Um outro servidor contém a camada com as regras de negócio, e serve de ligação entre a camada da base de dados e a camada de interface. Nestes dois servidores (o primeiro pode ser chamado de servidor de dados, e o

segundo de servidor de aplicações) podem ser implementadas regras de segurança rígidas, uma vez que no conjunto representam não só a informação da organização que usa a aplicação, mas os mecanismos que se usam para dar sentido a essa informação. Depois apenas são necessários computadores cliente que fazem uso da aplicação (mais concretamente da camada de apresentação).

Terceiro caso: um servidor contém a base de dados, juntamente com os stored procedures e a camada de regras de negócio. Neste servidor, podem ser implementadas regras de segurança rígidas, uma vez que contém não só a informação da organização que usa a aplicação, mas os mecanismos que se usam para dar sentido a essa informação. Depois apenas são necessários computadores cliente que fazem uso da aplicação (mais concretamente da camada de apresentação).

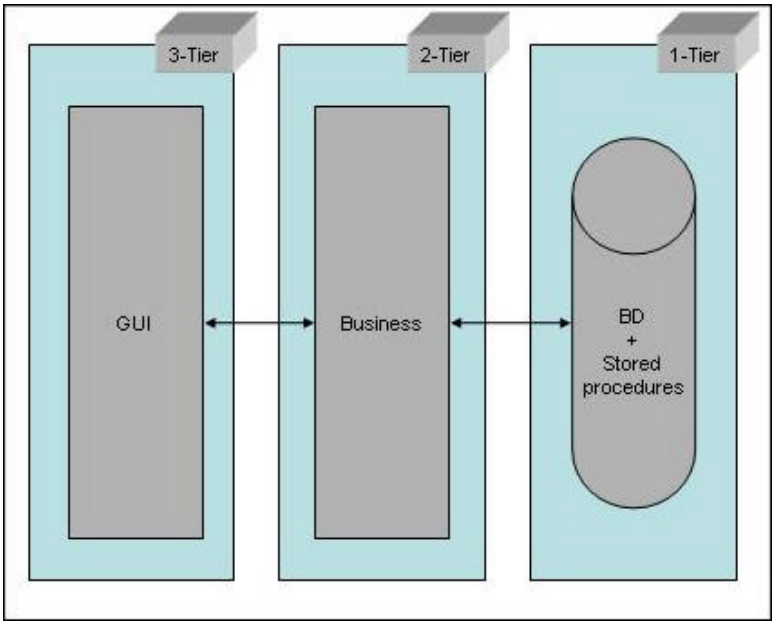


Figura VII : A estrutura 3-tier

O Visual Studio .Net

Introdução

Em finais do ano 2000 e início do ano 2001, a Microsoft apresentou uma nova ferramenta que promete revolucionar a indústria da construção de software. É então lançada a primeira versão beta de um produto que tinha sido apresentado dois anos antes a alguns programadores ainda na sua versão alfa, e que tinha causado euforia no grupo de pessoas que teve hipótese de o experimentar. Este produto era constituído pelo Visual Studio .Net e pela Framework .Net, que depressa chamou a atenção do mundo e que apontava características únicas e inéditas num sistema de desenvolvimento:

- ✓ O Common Language Runtime
- ✓ Várias linguagens de programação com uma mesma base (MSIL)
- ✓ Compatibilidade entre todas as linguagens
- ✓ Programação inteiramente orientada a objectos (OOP)
- ✓ O fim do inferno das DLL
- ✓ O fim das falhas de memória
- ✓ Ambiente de desenvolvimento (IDE) totalmente integrado

A partir deste momento, em vez de serem criados objectos binários COM que eram utilizados nas várias aplicações, passam a ser criadas classes que podem ser herdadas e estendidas por mecanismos de herança e delegação, sem qualquer preocupação relativa à linguagem de desenvolvimento em que foram construídas.

A Framework .Net

A Framework .Net disponibiliza uma base sólida e uma arquitectura poderosa para o desenvolvimento de aplicações. Uma das partes essenciais da framework é o CLR – Common Language Runtime – que permite que os programadores usem a linguagem com que mais se sentem à vontade (seja ela C#, C++, VB.Net, J#, etc.), e que ao ser compilada é transformada numa nova linguagem intermédia que não é propriamente linguagem máquina, mas sim MSIL (Microsoft Intermediate Language).

Ao resultado deste processo de interpretação dá-se o nome de assembly, que contém o código MSIL e ainda informação sobre a aplicação.

Quando a aplicação é invocada, o CLR é responsável por usar os blocos de MSIL que são necessários para executar a parte da aplicação que foi requisitada, e transforma-los em código máquina executável.

Outro aspecto importante que a framework controla, são as variáveis de ambiente e ainda um elevado número de classes que servem de suporte a todas as aplicações. Para finalizar, foi também incluído o suporte para multithreading na própria framework, em vez de estar ao cargo da linguagem, o que permite que linguagens que anteriormente não dispunham deste tecnologia (como o Visual Basic 6) passem a possibilitar a construção de aplicações "multithreaded".

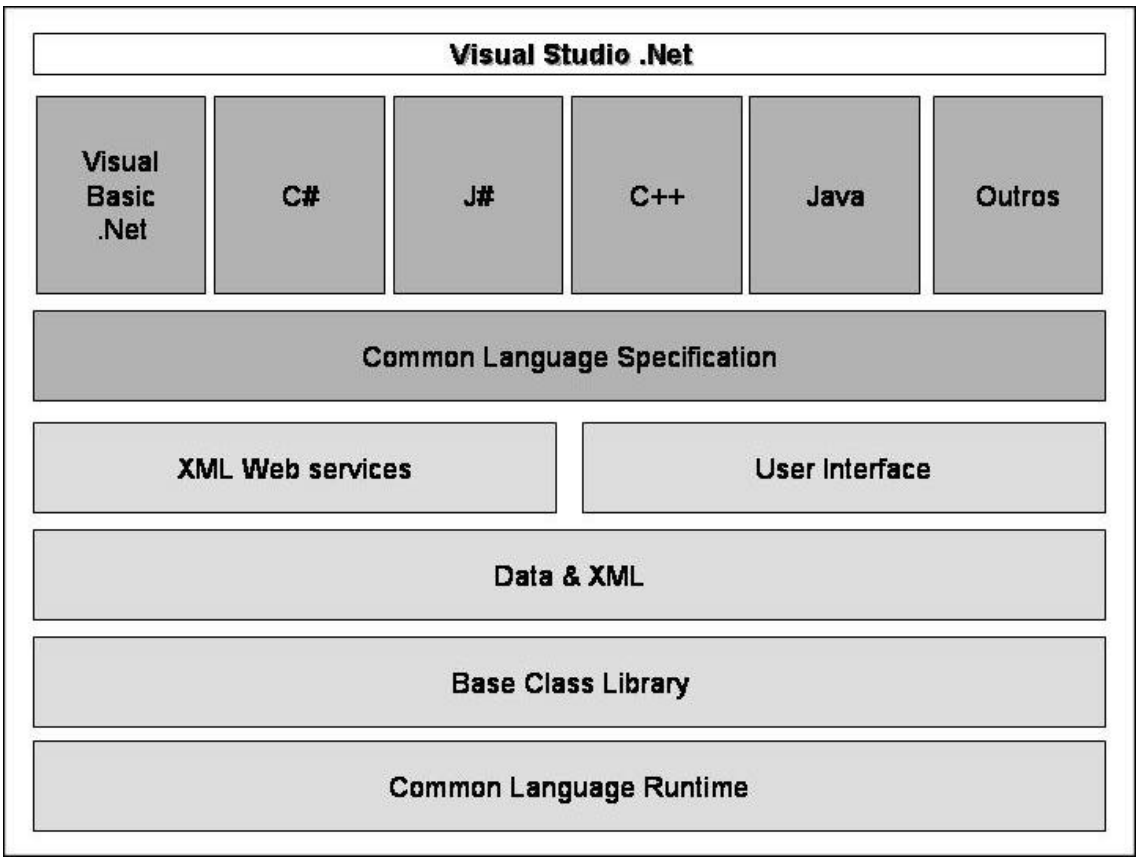


Figura VIII : A Framework .Net

O IDE

Com o novo conceito de IDE do VS.Net, é possível construir aplicações usando componentes pré-definidos em apenas alguns minutos, bastando clicar nos componetes existentes na Toolbox e arrastar os mesmos para o local no form (Windows ou Web) que se deseja, e depois, através da configuração dos atributos/propriedades, definir o comportamento do componente.

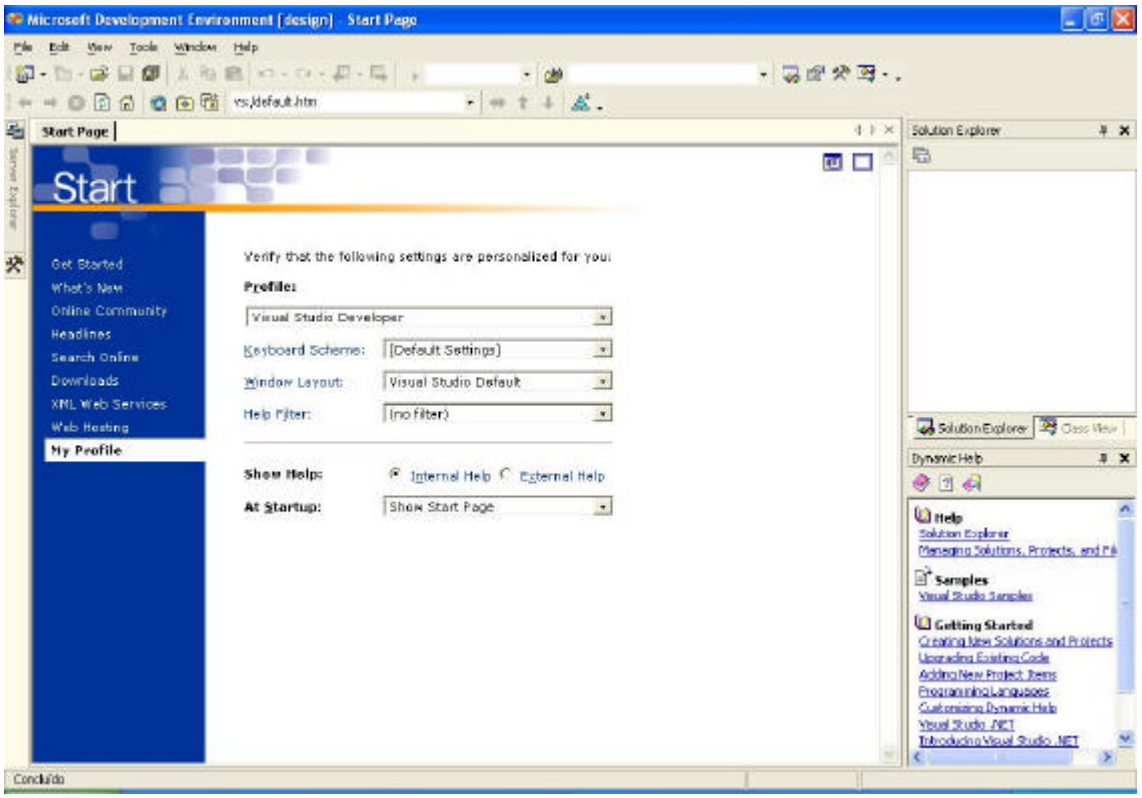


Figura IX : O IDE do Visual Studio .Net

Para melhor visualizar o funcionamento do IDE, da toolbox e do server explorer, podemos verificar a facilidade de criação de formulários sem escrever código, no projecto *licExemplo01* dos exemplos disponibilizados. Neste exemplo podem ver-se exemplos da criação de formulários, DataSets e DataAdapters, usando apenas os objectos da toolbox e do server explorer e parametrização dos mesmos objectos.

Na prática

A partir deste momento, uma equipa de desenvolvimento, constituída por programadores com conhecimentos diferentes e gostos diferentes não necessita de passar por processos de adaptação a linguagens e consequentes tecnologias de programação, porque é possível programarem na linguagem que mais gostam, sem problemas de integração das mesmas, e sem limitações a nível tecnológico, como é o caso do multithreading.

Quando compilamos e posteriormente executamos uma aplicação, ela é inicialmente traduzida para MSIL, e em tempo de execução (run-time) é transformada em código máquina. Isto permite que os assemblies sejam indiferenciados relativamente à linguagem, e também totalmente compatíveis.

Um pequeno senão, que na verdade não constitui um problema, prende-se com o tempo de execução. Na primeira vez que executamos uma aplicação, o tempo de execução é superior às execuções seguintes. Isto acontece porque o código não se encontra na forma nativa de linguagem máquina, mas sim em MSIL, e este terá que ser transformado em código binário executável. Este processo acontece apenas da primeira vez que executamos a aplicação. Nas restantes vezes, este processo não é necessário pois o assembly já contém o código traduzido para linguagem máquina da parte que foi executada anteriormente. Mas esta perda de rapidez é superada por outro mecanismo integrado na framework, que apenas executa (coloca em memória, e traduz para linguagem máquina no caso de ainda não estar) os blocos necessários para a execução da operação em curso, permitindo neste ponto um ganho a nível de rapidez.

Há ainda a possibilidade de, na altura da instalação da aplicação, gerar todo o código máquina referente à aplicação de uma só vez, usando o utilitário NGEN disponibilizado pela framework.

O Visual Studio .Net e a construção de aplicações N-Tier

A nova plataforma .Net é uma das mais vocacionadas para criação de aplicações para a Internet e aplicações que fazem uso do padrão N-Tier.

Os primeiros passos

Apesar de todo o suporte, e de toda a tecnologia existente, o passo mais importante da construção de qualquer aplicação, é a planificação.

O primeiro ponto da planificação é identificar as necessidades e objectivos dos clientes. Depois de clarificar este ponto, é tempo de seleccionar a plataforma e condições onde a aplicação irá executar, para que se possa definir a tecnologia bem como estruturar a aplicação. Depois há que escolher a base de dados (e SGBD), e definir como a informação será tratada, bem como definir os pontos de segurança, tendo em conta que o .Net disponibiliza grande parte dos mecanismos de segurança necessários.

De seguida deve começar-se por construir as classes base da aplicação e recursos do cliente. A partir do alargado suporte do .Net esta construção será acima de tudo uma combinação e por ventura uma extensão às classes existentes, uma vez que o suporte necessário encontra-se todo implementado na framework.

Partindo do suporte fornecido pela plataforma, basta definir as regras de negócio necessárias para implementar a aplicação.

Apesar da importância de alguns dos passos na criação de aplicações e da especificidade das mesmas, posso no entanto definir as fases genéricas para a construção de uma aplicação como sendo sete:

Fase 1: A analise do sistema

Para que se possa construir uma aplicação sólida é necessário que se conheçam os requisitos, os objectivos e as funcionalidades a que a aplicação deve obedecer.

Fase 2: Modelação da aplicação.

Após a construção do documento de análise do sistema, é necessário descrever as funções que a aplicação irá implementar, e dar inicio à descrição dos principais objectos que vão representar o sistema. A esta altura é já possível começar a integrar ferramentas que apesar de não serem nativas ao .Net estão totalmente desenhadas para funcionarem em conjunto com ele. Algumas dessas ferramentas são o Rational Rose ou o Visual UML.

Fase 3: Desenho da base de dados.

Partindo do ponto anterior com o sistema representado e documentado (e tendo em conta os principais objectos identificados), é a altura de desenhar a base de dados.

Fase 4: Desenho da arquitectura.

O desenho da arquitectura está especificado em todo este relatório, pelo que não vejo necessidade de voltar a explicar novamente este ponto.

Fase 5: Desenho da interface com o utilizador.

Este processo deve ser executado por um especialista em desenho de interfaces, uma vez que normalmente os programadores são pessoas familiarizadas com os processos informáticos e tendem a não conhecer as necessidades dos utilizadores finais. Além disso, como as aplicações de maior dimensão são elaboradas por vários programadores, exige-se que o conjunto da aplicação, em termos de interfaces, seja coerente e esteja largamente documentada.

Fase 6: Implementação.

A implementação ou codificação de uma aplicação assemelha-se à construção de uma casa. Primeiro vêm os alicerces e os andares de baixo e depois vamos subindo na estrutura. Numa aplicação informática N-Tier, primeiro implementam-se as classes e componentes mais genéricos, e depois de testados e de terminada uma camada mais baixa, passa-se para a camada seguinte e portanto mais alta na arquitectura.

Fase 7: Ligação das layers

É altura de ligar todos os componentes, todas as camadas e todas as funcionalidades de modo a encaixar os módulos construídos numa única aplicação.

A utilização destas fases de produção, em conjunto com a utilização do .Net permite ganhos significativos em termos de simplificação da construção, rapidez de implementação e facilidade de entendimento por parte das equipas envolvidas na criação da aplicação.

A arquitectura conveniente

Logo que estejam definidos objectivos e que estejam identificadas todas as necessidades, falta definir a arquitectura em termos de sistema distribuído, portanto vamos então ter que seleccionar o número de camadas em que devemos separar a aplicação, ou seja, definir verdadeiramente a aplicação N-Tier.

A escolha da arquitectura deve ser baseada nos pontos referidos anteriormente neste relatório (ver *"uma camada"*, *"duas camadas"* e *"três camadas"*), bem como o ponto que será abordado de seguida.

A criação da aplicação

A framework .Net e o IDE, permitem que facilmente possamos criar uma arquitectura N-Tier flexível, isto é, não precisamos de ficar presos aos modelos de uma, duas ou três camadas descritos anteriormente neste relatório, e podemos facilmente estender o conceito da divisão em camadas para o aumento do número das mesmas e para a subdivisão lógica da aplicação.

Com todo o suporte tecnológico fornecido, não há necessidade de qualquer especificação extra no que diz respeito, por exemplo, à diferenciação de tratamento das interfaces Web e Windows. Para isso basta criar uma nova camada lógica, a que a Microsoft chamou de Facade Layer, e tendo como base a camada de regras de negócio que será igual para toda a aplicação independentemente da interface, construir nesta camada as funcionalidades que diferenciam (caso isso aconteça) o funcionamento em Web e Windows, ou qualquer outro tipo de interface.

A camada Facade é implementada sobre a camada de Business Rules, e as regras de comunicação alteram-se para o seguinte estado: a camada de dados continua a comunicar com a camada de Business Rules, e esta porem passa a comunicar com a camada Facade. Por fim, esta nova camada comunica com a camada de apresentação, usando para isso a regra específica criada na mesma para satisfazer as especificações da interface em questão. Isto significa que passamos a ter além de regras de negócio, regras de interface que são definidas na aplicação.

Passamos então a ter quatro camadas lógicas na aplicação. No entanto nem sempre há necessidade de tratamento quer em termos de regras de interface, quer em termos de regras de negócio, pelo que estas camadas intermédias podem existir apenas para troca de informação e dados entre as camadas de apresentação e acesso a dados. Neste caso podemos pensar que as camadas de Facade e Business Rules são inúteis, mas no entanto se a dada altura a empresa tiver necessidade de aplicar regras de negócio ou interface, não necessita de refazer toda a aplicação, porque as camadas encontram-se já criadas e em funcionamento e assim basta implementar as regras necessárias e substituir apenas a parte que foi alterada sem afectar o normal funcionamento do conjunto das camadas e portanto da própria aplicação.

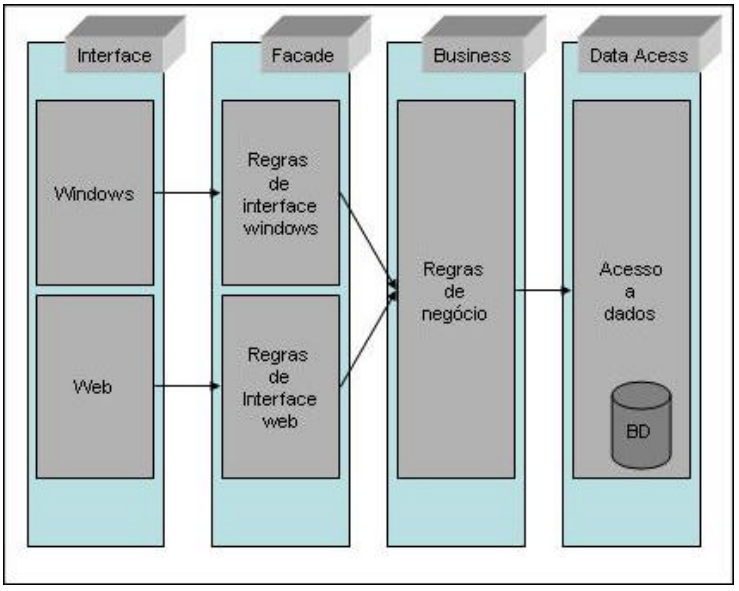


Figura X : A arquitectura flexível do .Net

No entanto, até ao momento tenho referido o suporte para construção da interface com clientes Windows, clientes usando browser e um outro tipo de clientes muito menos usado, mas que se enquadra nos clientes Windows que são os clientes de consola, mas o .Net permite a integração nativa de mais dois tipos de interfaces, são elas interface para Pocket PC, ou seja para PDA ou dispositivos compatíveis, e ainda suporte para interfaces para execução em telemóvel ou outro dispositivo compatível.

Esta possibilidade de usar vários tipos de interfaces permite que apenas tenhamos que fazer alterações na camada Facade, colocando apenas as regras especificas para a interface que se deseja.

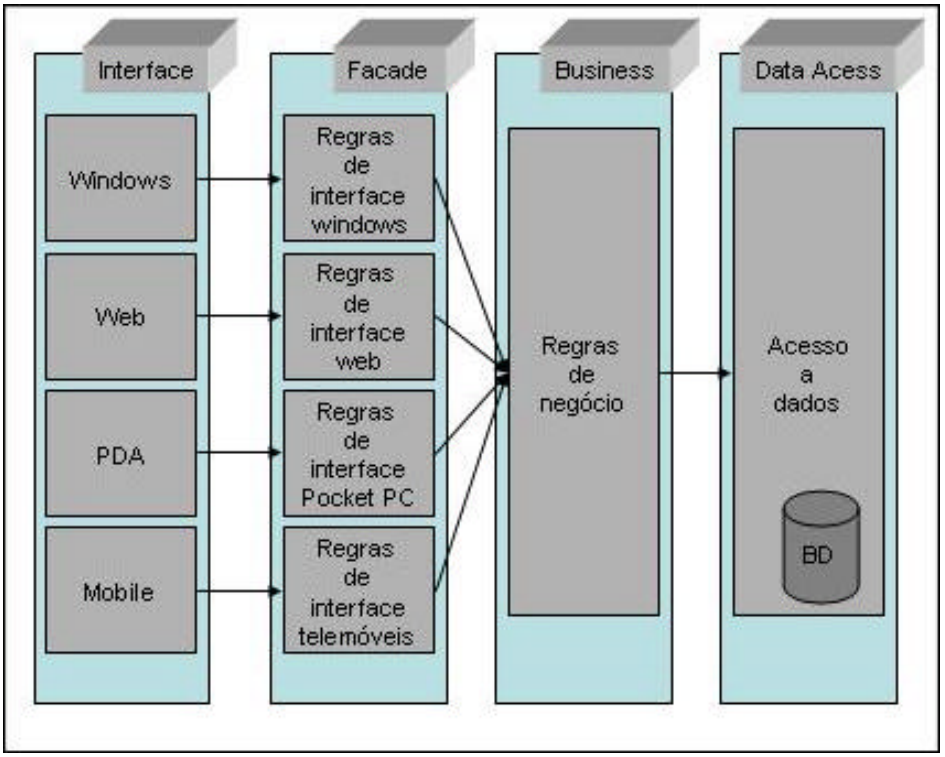


Figura XI : A arquitectura flexível do .Net ao serviço das interfaces

No exemplo *licExemplo02*, podemos visualizar esta estrutura flexível implementada. Mas a visualização não é semelhante à da representação na figura XI. Na implementação uma camada pode ser representada por um ou mais projectos. Para melhor entender essa estrutura vou pois explicar quais os projectos que constituem as camadas e quais as suas funções.

A camada Data Access que irá tratar dos detalhes de ligação, é constituída pelos projectos *licDataSets* e *licDataAccess*. O primeiro projecto contém apenas os DataSets que vão ser usados na aplicação. O segundo projecto contém os componentes com as Connections, os DataAdapters e as classes que implementam as funcionalidades necessárias para a gestão da informação nos DataSets.

A camada Business, que irá conter a implementação das regras de negócio especificadas para a aplicação, é constituída pelo projecto *licBusinessRules* que contém as classes que iram tratar, de acordo com as regras anteriormente definidas, a informação contida nos vários DataSets.

A camada Facade, que irá conter a implementação das regras de interface que sejam necessárias para o correcto funcionamento da aplicação, é constituída pelo projecto licFacade que contém as classes necessárias para a implementação das suas funcionalidades.

A camada Interface é constituída pelos projectos *licWinUi* e *licWebUi*, que são respectivamente a interface para windows e a interface para web. Nestes projectos, e portanto nos respectivos formulários são tratadas as operações de interface, e são feitas chamadas, através da camada Facade que permitem gerir a informação existente nos DataSets que reflecte as operações disponibilizadas pelos formulários. De momento, a interface web serve apenas para visualização de dados, e a interface windows permite já um vasto conjunto de operações.

Enterprise Templates

Os enterprise templates são protótipos de projectos existentes no .Net que permitem a criação da aplicação e a restrição em termos de tipos de objectos que se podem criar em cada camada, ou seja, permite que quando se define a estrutura inicial de uma aplicação, sejam definidas simultaneamente as regras arquitecturais e tecnológicas, que evitam que os programadores cometam certo tipo de erros que possam colocar em perigo a segurança da aplicação.

Como exemplo, podemos observar que não é desejável que o programador crie procedimentos e objectos que possam aceder à base de dados directamente na camada facade, a menos que sejam apenas procedimentos de leitura, mas que apesar de tudo acho que devem sempre passar por todas as camadas. Usando os enterprise templates, este tipo de situações fica controlado.

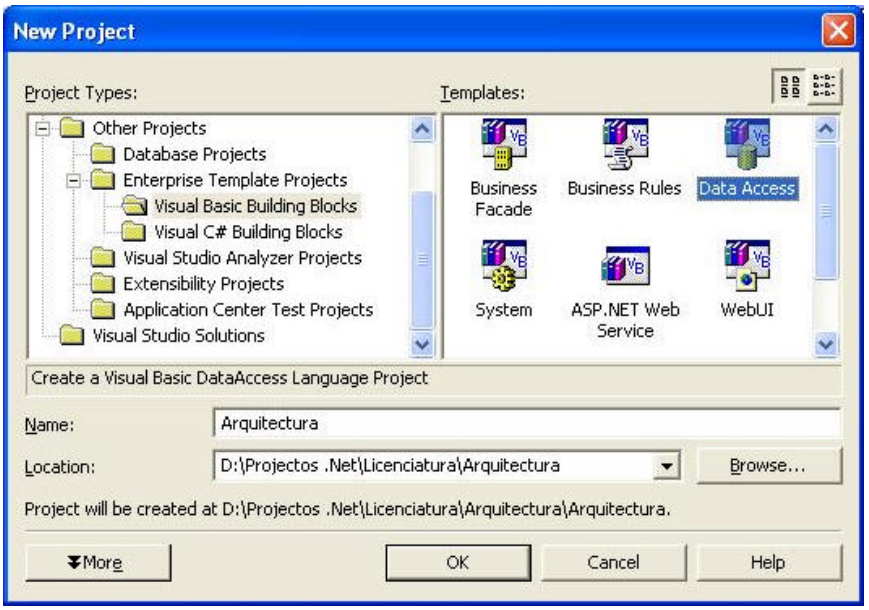


Figura XII : Enterprise Templates

Como me parece de interesse elevado, vou pois definir quais os templates, e o que cada um deles permite:

Template “System Framework”:

Esta layer trata das partes da aplicação que lidam directamente com o sistema em que estão instaladas. Contém serviços de cache e leitura de variáveis de ambiente. As funcionalidades aqui implementadas podem ser usadas por qualquer uma das camadas da aplicação.

Template “*Data Access*”:

Este layer permite o acesso directo a dados. Deve fazer uso do ADO.Net e dos objectos por ele disponibilizados.

Recebe pedidos da layer seguinte (Business Rules), pode ou não usar stored procedures implementados na base de dados ou mesmo nos objectos do próprio ADO.Net, e devolve resultados das perguntas (querys) que faz à base de dados.

Template “Business Rules”:

Este layer contém implementadas as regras de negócio referentes aos processos que executa. Um objecto presente nesta camada denomina-se “*objecto de negócio*”.

Neste template, processam-se os pedidos feitos pela camada Facade, aplicando-lhe nas regras de negócio e usam-se as funcionalidades do *Data Access* para aceder e gerir dados.

Template “Business Facade”:

Este layer é usado para providenciar consistência às interfaces, e isolar o cliente de alterações na lógica de negócio. Recebe inputs da camada de interface, e deve fazer pedidos à camada de Business Rules, embora em alguns casos possa fazer pedidos directamente à camada de Data Access. Na minha opinião, esta camada deve estar sempre presente, mesmo que não seja usada, e os pedidos devem ser feitos à camada Business Rules e nunca directamente à camada Data Access.

Template “Asp.Net Web Service”:

Este layer fornece interfaces web públicas que podem ser acedidas pelos clientes (o acesso pode ser web ou windows). Esta camada é semelhante à camada Facade, com a excepção de que os métodos que este disponibiliza podem ser acedidos por outros sistemas em vez de estar restrito apenas à aplicação em que estão construídos.

Template “WebUi” e “WinUi”:

Este layer é responsável pela implementação das interfaces dos clientes. No caso de aplicações distribuídas estes clientes são diversificados e podem ter acesso windows ou web, ambos ou simplesmente nenhum destes. Esta camada é responsável por controlar o comportamento da interface, apresentar dados e fazer alguns cálculos, validar a introdução de dados, apresentar formas de executar determinadas operações, enviar dados e pedidos para a camada de Web Services ou para a camada Facade e apresentar ao cliente o estado das operações, nomeadamente no que diz respeito a erros.

A tecnologia .Net à disposição

Todo o suporte fornecido para a construção deste tipo de aplicações parte de um conjunto de tecnologias introduzidas ou aperfeiçoadas pela Microsoft para facilitar a construção, ou simplesmente para ocultar detalhes que apenas dificultavam o seu desenvolvimento.

De seguida vou referir alguns exemplos interessantes dessas tecnologias, e explicar como se usam e para que servem aplicadas aos sistemas N-Tier. As referências que vou fazer não serão extensas, porque não estão no âmbito deste relatório.

O ADO.Net

O ADO.Net é o resultado de anos de esforço por parte da Microsoft e da evolução das antigas versões de ADO existentes.

Segundo testes efectuados e publicados pela Microsoft, o ADO.Net permite uma melhoria significativa em termos de escabilidade e fiabilidade bem como melhorias de 50% rapidez na manipulação de dados. Estes testes foram realizados usando ADO.Net sobre SQL Server, e são efectuados em comparação com outros sistemas middle tier de acesso a bases de dados, isto porque o ADO.Net usa a linguagem de comunicação nativa do SQL Server, o TDS (Tabular Data Stream) para comunicar com a base de dados.

Uma outra vantagem do ADO.Net está relacionada com as comunicações. Uma aplicação N-Tier exige que os dados sejam passados entre as camadas.

Actualmente toda a comunicação assenta sobre XML (Extensible Markup Language) e não é necessária a transformação dos objectos para que possam passar pelas várias camadas, eliminando assim a necessidade de "traduzir" esses objectos e portanto permitir um ganho em termos de tempo e mesmo compatibilidade de tipos de dados.

Para resolver alguns problemas como o número de acessos e bloqueios da base de dados, foi introduzido um novo conceito, que permite representar objectos da base de dados mas desconectado da mesma. O objecto que permite este processo é o DataSet. O que acontece no ADO.Net é que recorremos à base de dados para copiar a estrutura da mesma (pelo menos da parte que interessa para a operação, pode ser uma tabela simplesmente), e preenchemos esse objecto que tem a estrutura semelhante à da base de dados com os dados referentes, e a partir deste momento todo o trabalho é feito sobre este objecto. O estado da base de dados mantém-se inalterado, porque estaremos a trabalhar com um objecto que está offline em relação à base de dados. No final de utilizarmos o objecto, procedemos às alterações necessárias na base de dados. Com tudo isto, poupamos conexões à base de dados e locks de registos, permitindo mais acessos à mesma base de dados.

No que diz respeito a interoperabilidade entre sistemas, o facto do XML ser actualmente o standard de transporte de informação, significa que qualquer sistema que "compreenda" XML pode usar componentes ADO.Net.

Esquematizando a estrutura do ADO.Net podemos obter a seguinte figura:

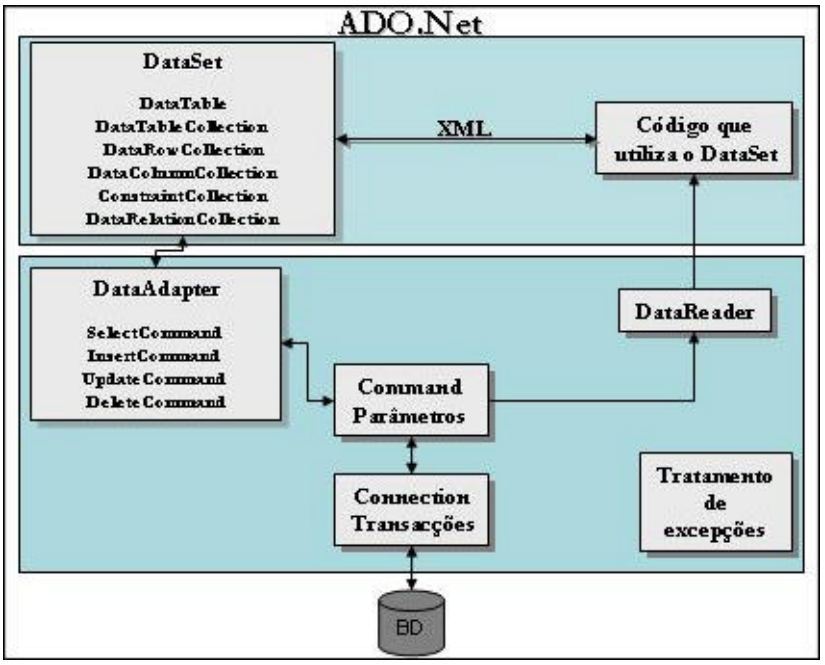


Figura XIII : A arquitectura do ADO.Net

Em conjunto com o DataSet, existem quatro objectos que formam o ADO.Net. São eles a Connection, o Command, o DataReader e o DataAdapter.

A Connection é o objecto responsável pela ligação aos dados (data source) e implementação de mecanismos de transacção, e tem um único parâmetro configurável, que é a connection string.

O Command é o objecto que possibilita o acesso aos dados e trata de operações devolução de dados (leitura), a alteração de dados, a execução de stored procedures e o envio e recepção de parâmetros. É aqui que residem os SQL Statements, ou seja, as querys que serão executadas na base de dados.

O DataReader é o objecto responsável por construir uma stream com dados referentes a um único registo em que se encontra posicionado. Este objecto é capaz de elevadas performances, mas tem a desvantagem de ter que manter a conexão à base de dados durante a sua utilização, ou seja, diminui a possibilidade de manter níveis de escalabilidade elevados.

O DataAdapter é o objecto que faz a ligação entre a base de dados e o DataSet. Este objecto preenche o DataSet usando os objectos Connection e Command, e através de mecanismos que lhe permitem reconhecer os registos que foram inseridos, alterados ou eliminados, faz novamente uso da Connection e do Command e procede às mudanças na base de dados.

Em termos de DataSet, há que dizer que é possivelmente o objecto mais importante do ADO.Net, essencialmente porque é um modelo de dados capaz de representar a base de dados mas desconectado da mesma. No entanto, toda a eficiência e potencialidade deste objecto tem um custo, custo esse que se reflecte na quantidade de memória necessária. Mas este facto é justificável pelos seguintes motivos: o DataSet não contém apenas os dados da base de dados (pode conter zero ou mais tabelas), mas contém a estrutura das linhas e

colunas de cada tabela, bem como informação sobre as relações e constraints. Uma solução para não esgotar os recursos a nível de memória, passa por implementar regras na base de dados (stored procedures), ou mesmo nos pedidos do DataAdapter (Commands), de forma a que, quando acede para carregar informação da base de dados, lhe seja devolvida apenas os dados estritamente necessários, ficando assim o DataSet com a estrutura e apenas algumas linhas preenchidas.

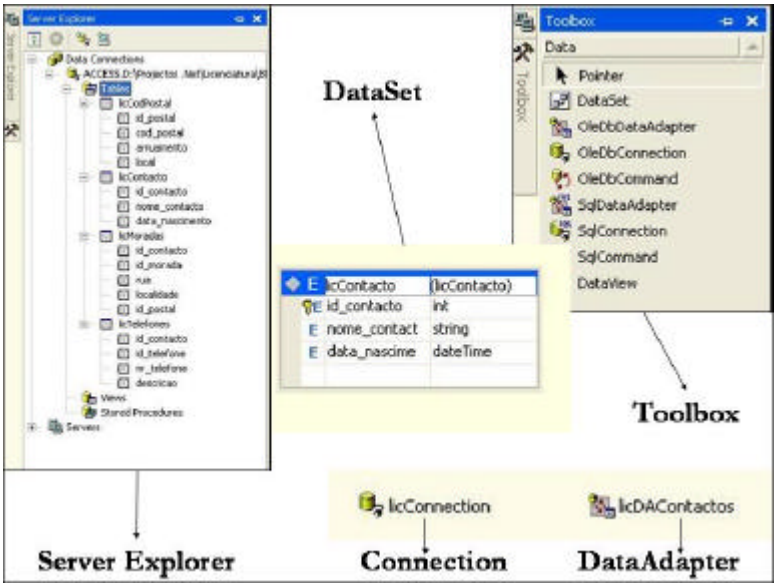


Figura XIV : O ADO.Net e o IDE do Visual Studio .Net

O IDE do .Net em conjunto com o ADO.Net, permite que apenas com algumas operações de clicar e arrastar componentes da Toolbox ou do Server Explorer, e fazendo alguma parametrização, se crie fácil e rapidamente a camada de DataAccess, sem que tenhamos que nos preocupar com conhecimento profundo da tecnologia ou mesmo do modo de funcionamento dos objectos.

No entanto até ao momento, não mencionei a questão da concorrência. Apesar dos DataSets trabalharem desligados da base de dados, em determinado momento os dados têm que ser actualizados na base de dados. E aqui volta a surgir o problema de quais os dados que permanecem sobre outros. Para isso, os próprios DataSets juntamente com os DataAdapters têm mecanismos de controlo de concorrência, que podem ser melhorados se usarmos na base de dados mecanismos de transacção e stored procedures para realizar todas as operações.

Web Services

Como podemos verificar a partir do próprio nome, os web services são serviços de software que são publicados na web, que comunicam usando o standard XML sobre Soap.

A importância dos web services deve-se essencialmente ao facto de serem uma tecnologia que permite a interoperabilidade entre aplicações que podem ser de diferentes organizações, e que estão capacitadas para consumir e fornecer serviços, do e para o exterior da organização.

Além dos serviços que disponibilizam, os web services fornecem também formas de descrever os meus serviços, para que possam ser encontrados por organizações que procuram um determinado serviço, bem como mecanismos de entrega de mensagens.

Do ponto de vista das organizações que disponibilizam os web services, eles são vistos como mecanismos de expor com relativa segurança alguns processos de negócio, para além do domínio da rede local. Ou seja, passamos a ter mecanismos que permitem expor funcionalidades à escala global, através da Internet. Assim, como disse anteriormente, as organizações usam os seus web services para disponibilizar serviços para outras organizações, mas também para poderem procurar outros web services ou aplicações que lhes permitam comunicar com outras organizações.

Existe ainda muita confusão acerca dos web services. Alguns autores e programadores referem-se a eles como a nova ferramenta para a computação distribuída. Mas esta definição e utilização que se dá aos web services está longe do objectivo para que foram criados.

Como não é simples perceber ao certo o que é um web service, o melhor é tornar as coisas simples. A tecnologia dos web services assenta sobre quatro componentes: O serviço, o documento, o endereço e o envelope.

O serviço é a funcionalidade que o componente de software proporciona. A principal exigência destes serviços é que sejam capazes de processar documentos XML (aqui podemos ver documentos como mensagens). Os detalhes do serviço não são minimamente importantes. Quer usem técnicas orientadas a objectos, quer sejam processos solitários ou parte de outras aplicações, independentemente da linguagem de programação em que são feitos, só têm que garantir o que está especificado anteriormente, ou seja, têm que ser capazes de processar documentos XML.

O documento é uma mensagem XML que é enviada a um serviço (web service) para ser processada de acordo com a lógica da empresa que é documentada pelo serviço, e que contém informação relativa ao processo que executa. A linguagem que descreve estes documentos e as formalidades para a conversação dos web services é denominada de WSDL (Web Services Description Language).

O endereço é a forma de aceder ao serviço. O endereço é definido com base no protocolo que serve de canal aos documentos XML (pode ser HTTP, TCP ou UDP, por exemplo). É através este endereço que se encontram os web services, e estes endereços estão registados no UDDI (Universal Discovery Description and Integration).

O envelope é a forma de encapsulamento do documento XML. A forma de encapsulamento nos web services é o Soap. Desta forma está definida uma maneira do documento XML viajar entre os serviços com segurança, salvaguardando protocolos de routing e de segurança, sem alterar qualquer ponto do documento.

Do ponto de vista interno da organização e da utilzação do .Net, para ligar um Web Service, à aplicação que se deseja, neste caso numa aplicação N-Tier (no âmbito deste documento), basta adicionar uma referência (web reference) no projecto que implementa o

serviço que se quer publicar. Como os Web Services têm virtualmente acesso a toda a framework .Net, podemos fazer neles tudo o que quisermos. É possível usar mecanismos de autenticação, caching, theads, etc.

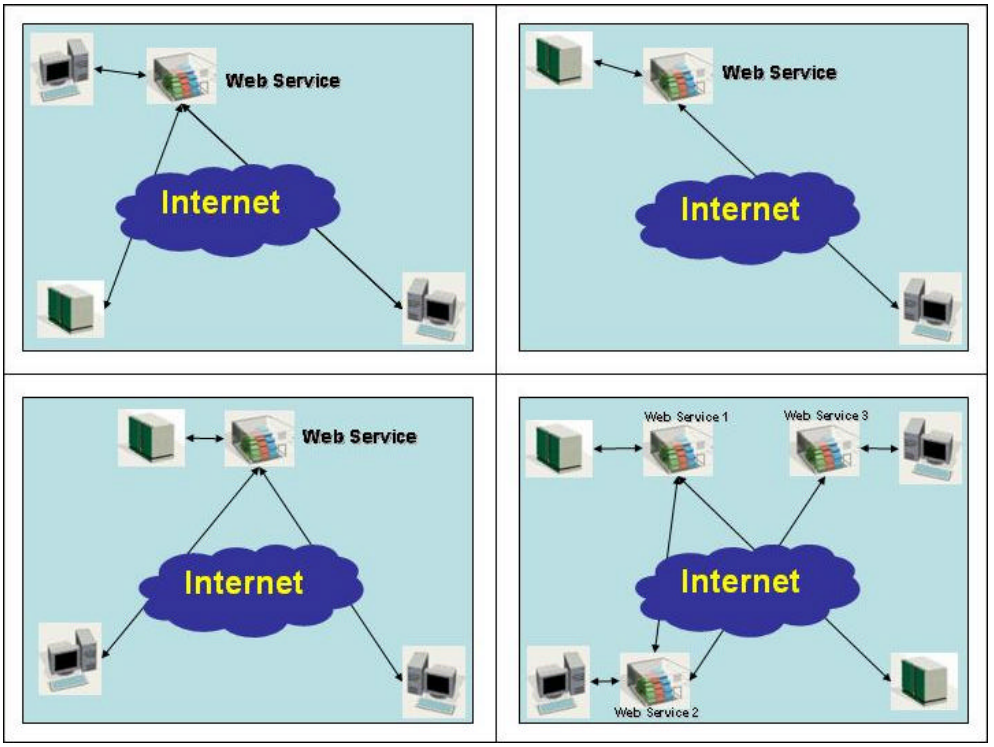


Figura XV : Web Services

Mas o processo de entender o que é um web service não é simples ao ponto de se perceber após a leitura deste texto ou de qualquer um artigo, e por isso acho que é conveniente apontar alguns erros comuns na identificação e definição dos web services. Em primeiro lugar, os web services não são objectos distribuídos (apesar de algumas semelhanças nos conceitos). Em segundo lugar, os web services não são os mecanismos de RPC (remote procedure call) sobre a Internet. Em terceiro lugar, os web services comunicam sobre Soap pelos canais HTTP, TCP e UDP e não apenas sobre HTTP. Haveria muitos outros erros a apontar no que diz respeito à interpretação que se dá aos web services, mas estaria a ultrapassar o âmbito do relatório, pelo que ficam aqui registados apenas os mais comuns.

Remoting

É comum a necessidade de comunicação entre objectos que executem em diferentes processos, quer seja na mesma máquina ou em máquinas diferentes. Esta necessidade dá-se sobretudo em aplicações distribuídas e no caso em questão, em aplicações usando o padrão N-Tier. Para isso o .Net disponibiliza o .Net Remoting.

O .Net Remoting é a nova infra-estrutura da Microsoft que disponibiliza um conjunto de classes que permite encapsular e simplificar os mecanismos de utilização de objectos remotos. O remoting permite a invocação de métodos com um nível de complexidade quase semelhante ao da invocação local.

Existem três partes integrantes no remoting: um objecto, um servidor e um cliente. O objecto implementa os métodos e funcionalidades que poderão ser invocados remotamente. O servidor é o local do domínio onde se encontra o objecto remoto, e que irá receber os pedidos para a execução do objecto remoto que contém. O cliente é o local no domínio que faz invocações ao objecto remoto.

Quando um cliente cria uma instância de um objecto remoto, ele recebe um proxy para a classe instanciada no servidor. Todos os métodos chamados através do proxy são canalizados para a classe remota e os resultados da invocação são devolvidos para o cliente.

Em remoting, a passagem de parâmetros é feita não só por valor, mas também por referência.

Existem duas formas de activar objectos remotos: Server Side Activation e Client Side Activation.

Na activação do lado do servidor (Server Side Activation), o objecto é criado e executado pelo servidor e do lado do servidor, enquanto que o cliente apenas cria o proxy para a comunicação. Neste tipo de activação, existem duas formas de controlar o estado do objecto. O modo SingleCall define que a cada chamada a um método, irá ser criada uma nova instância do objecto, e portanto, não é possível manter o estado do objecto. No modo Singleton, após a criação remota do objecto, todos os métodos vão ser executados naquele objecto, e portanto é possível manter o estado do mesmo objecto.

Na activação do lado do cliente (Client Side Activation), é o próprio cliente que invoca o construtor do objecto que quer usar, passando parâmetros para o método e portanto controlando a existência do objecto.

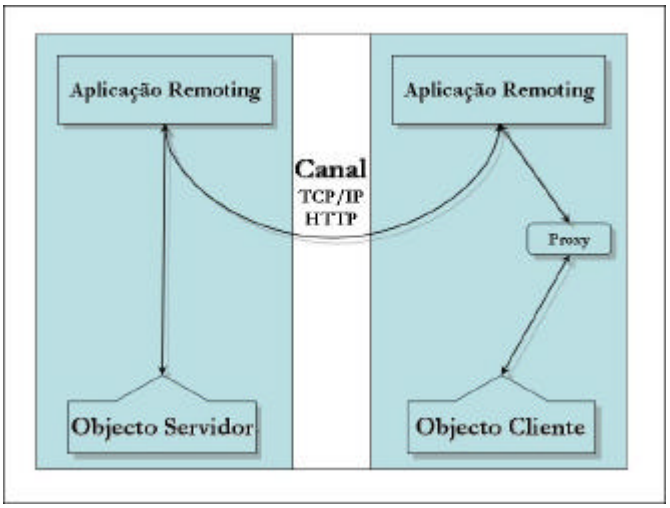


Figura XVI : Remoting

Windows DNA *Vs* N-Tier em .Net

Parece-me interessante colocar frente a frente as tecnologias que definem o conceito de N-Tier tradicional (Windows DNA) ou mais antigo como queiram chamar-lhe, que faz uso maioritariamente da tecnologia COM, contra o novo conceito de N-Tier usando a tecnologia .Net. É verdade que acho que se pode fazer tudo com as duas tecnologias, mas vamos ver alguns pontos nesta comparação entre o COM e o .Net.

A identidade dos objectos

Os objectos COM requerem a definição de um GUID. A cada interface pública é atribuído um GUID, e mesmo a própria classe a que pertencem as interfaces tem que ter um. Os GUIDs são armazenados no registry de modo a que os objectos a que se referem possam ser encontrados. De cada vez que houver necessidade de alterar um objecto COM, um novo GUID é criado, e portanto passa a haver um novo objecto.

Ao contrário, no .Net, o CLR é responsável por criar nomes únicos com assinatura digital, designados de strong names, e que identificam os objectos. As alterações não significam a criação de novos objectos.

Type Libraries Vs Assemblies

O COM usa type libraries para armazenar informação referente aos objectos. Essa informação é referente apenas às interfaces públicas.

No caso do .Net, os assemblies contêm informação sobre a classe, e os métodos públicos e privados.

Tratamento de erros

Todos os métodos COM geram retornos do tipo HResult. Este retorno indica se a execução do método teve ou não sucesso, bem como alguma informação sobre os erros que possam ocorrer.

Com o .Net, o tratamento de erros é sempre feito por excepções, o que permite que toda a informação sobre o erro seja lançada para os mecanismos de tratamento de excepções.

Tempo de vida dos objectos

No .Net. o tempo de vida dos objectos é determinado por um mecanismo chamado garbage collector, que é controlado pelo CLR. Este tipo de gestão de tempo de vida é chamado de não determinístico, uma vez que o garbage collector não é controlado pelo programador, mas pela framework, que o leva a actuar aleatoriamente (do ponto de vista do programador, já que este não tem controlo). Isto significa que a existência do objecto não termina quando ele deixa de ser utilizado, mas apenas quando o garbage collector actua.

Em relação ao COM, o tempo de vida dos objectos é determinístico e controlado por um contador simples. A cada criação de uma cópia do objecto, é invocado uma interface obrigatória dos COM chamado IUnknown que contém um método AddRef que incrementa o contador. Quando o objecto deixa de ser utilizado, deve invocar-se o método Release que decrementa o contador. Quando o contador chega a zero, o objecto é destruído.

IDispatch Vs Reflection

Os objectos COM implementam ainda uma outra interface chamada IDispatch que permite a Automação. A Automação tem como objectivo permitir que as aplicações possam expor as suas funcionalidades como componentes que possam ser utilizados por outras aplicações. Esta interface permite ainda uma forma de encontrar interfaces em run time, em vez de os encontrar apenas na altura da compilação – Late binding.

Em contraste, o .Net permite o late binding através de Reflection, que é um mecanismo com o qual se pode descobrir tudo sobre um objecto. Pode-se descobrir campos, propriedades, métodos, interfaces e a hierarquia estabelecida pela herança. Uma outra potencialidade da Reflection, é a de injectar MSIL directamente nos métodos em run time e mesmo criar novos métodos (mas há que ter cuidado com este tipo de prática).

Dificuldades

A principal dificuldade na realização deste projecto foi a de tentar conjugar as definições técnicas correctas com o entendimento que tinha em alguns assuntos relacionados com a utilização do .Net. Por exemplo, no que diz respeito aos Web Services, pude verificar que realmente tinha as mesmas dúvidas que foram apresentadas neste relatório e que verifiquei que tratava-se de uma confusão generalizada, porque apesar de podermos usar os Web Services para ligar as camadas N-Tier, o objectivo para que foram criados está para além disso.

Conclusão

Como tentei demonstrar até este ponto no trabalho, a Framework .Net fornece-nos uma base sólida, recorrendo a um conjunto bem definido de classes que implementam funcionalidades de grande importância, para o desenvolvimento de aplicações N-Tier.

Se ao vasto suporte fornecido pela Framework .Net, juntar-mos a rapidez de desenvolvimento e simplicidade de utilização dos componentes e funcionalidades do Visual Studio .Net, temos um poderoso conjunto, que facilita o desenvolvimento e possibilita grandes avanços no que diz respeito à elaboração de aplicações empresariais com padrão N-Tier.

Podemos ainda melhorar os métodos de trabalho das equipas de desenvolvimento de software permitindo que elas programem na linguagem que mais se sentem à vontade, sem prejuízo para a posterior integração das partes desenvolvidas. Estes factores aumentam os níveis de satisfação, eficiência e competência dos programadores, pormenores que só fazem aumentar a competitividade das organizações.

Resta resumir que a utilização da Framework .Net e do Visual Studio .Net são uma forma rápida e eficaz de construir aplicações com padrão N-Tier, e portanto beneficiar de todas as vantagens que este padrão oferece, bem como adicionar funcionalidades de interoperabilidade com aplicações externas, que favorecem as estruturas empresariais actuais, nomeadamente a interligação das chamadas empresas virtuais e das organizações de e-commerce.

Glossário

Business Rules

São as regras que se podem definir e pelas quais se regem os processos de negócio. Podem ser estruturas para apresentar informação, ou podem ser fórmulas a aplicar a determinados dados de forma a produzir novos dados.

Connection String

É constituída por um determinado conjunto de dados que permitem que se faça ligação a uma determinada base de dados. De entre alguns desses dados realço o login e a password autenticados para permitir a ligação, bem como o nome do servidor e do catalogo onde se encontra a base de dados à qual se quer efectuar a ligação.

CLR

Common Language Runtime

COM

Component Object Model

COM é a tecnologia que permite implementar o conceito de componente de software, que é um objecto binário que disponibiliza serviços que são apresentados sobre a forma de atributos e métodos que são acessíveis apenas por interfaces.

Fat Client

Cliente que devido à carga e número de operações que pode realizar, exige um elevado nível de recursos, e realiza um elevado número de tarefas.

Fat Server

Servidor que devido à carga e número de operações que pode realizar, exige um elevado nível de recursos, e realiza um elevado número de tarefas.

GUID

Global Unique Identifier é um código de 128 bits que serve para identificar entre outros elementos os interfaces e as classes COM.

IIS

Internet Information Service – é um componente do Windows que facilita a publicação de informações e a expansão das aplicações pela Web, e fornece uma plataforma de comunicação de rede, bem como mecanismos de autenticação e segurança próprios.

Middle Tier

Middle Tier é considerada a camada intermédia que se usa para aceder a bases de dados. Como exemplo, posso mencionar o ODBC.

MSIL

Microsoft Intermediate Language

NGEN

Native Image Generator – este utilitário cria uma imagem nativa do código existente no assembly, e coloca-o num sistema de cache (chamado native image cache) no computador em que vai executar. A execução deste utilitário sobre um assembly permite que ele seja carregado e execute mais rapidamente, porque ele vai buscar o código e as estruturas de dados ao native image cache, em vez de estar a gerar dinamicamente esse código apartir do assembly.

OOP

Object Oriented Programming (Programação Orientada a Objectos)

Soap

O Soap é um protocolo que basicamente permite a comunicação de XML sobre HTTP, e por isso tornou-se um standard para transferência de dados sobre a forma de mensagens com formato XML. Como é baseado em XML, o Soap pode ser entendido por quase todas as plataformas que consigam trabalhar com XML.

Stored Procedures

Stored procedures são conjuntos de instruções Transact-SQL aglomeradas num único plano de execução.

Thin Client

Cliente que devido à pequena carga computacional imposta pelas operações que tem que realizar, exige um nível de recursos diminuto.

XML

O Extensible Markup Language (XML) é um método uniforme para descrever e transportar dados estruturados, independentemente da aplicação que faz uso dele. Standard derivado da SGML (Standard Generalized Markup Language).

Índice de Figuras

Bibliografia

http://www.n-tier.com

http://www.microsoft.com

http://www.msdn.com

http://www.w3c.org

"OOP: Building Reusable Components with Microsoft Visual Studio .Net" – Edição Microsoft .

"Designing Enterprise application with Microsoft Visual Basic .Net" – Edição Microsoft.

Visual Studio .Net Combined Collection – Ajuda do Microsoft Visual Studio .Net