Instituto Politécnico de Viana do Castelo

Instituto Politécnico de Viana do Castelo

Escola Superior de Tecnologia e Gestão

Integração de Sistemas de informação e de Tecnologias

Aulas Teóricas

Informação Base HISTÒRICA

→ que deverá ser complementada com a vasta informação MAIS RECENTE
→ disponibilizada na página de e-learning

Escola Superior de Tecnologia e Gestão
Instituto Politécnico de Viana do Castelo
Braga, Portugal

Jorge Ribeiro
jribeiro@estg.ipvc.pt

Objectivos:

Pretende-se que a disciplina possibilite:

- Identificar necessidade de integração de Sistemas de Informação;
- Identificar tipos de integração de SI;
- Caracterizar a geografia da integração de SI;
- Identificar tecnologias de integração;

Motivação:

A capacidade de integrar sistemas de informação, isto é, sintetizar sistemas complexos, a partir da composição de subsistemas e componentes diversos, se revela como fundamental para o futuro. Verdade se diga que a maturidade científica e tecnológica da engenharia informática neste domínio de integração de sistemas se encontra ao nível do artesanato.

• Conteúdo Programático

Aulas Teóricas:

I -Integração de Sistemas e Tecnologias:
- Tipos de Integração de SI;
- Plataformas de Integração
- Impacto da Internet;

II - Tecnologias de Integração:
- Integração no Computador:
  - Ficheiros:
  - Sockets
  - Bases de Dados
  - Monitores Transaccionais
  - Componentes
  - Servidores Aplicacionais

- Integração na Organização:
  - Mensagens
  - Procedimentos remotos
  - Objectos Distribuídos
  - Código Móvel
  - Message Brokers
- Integração entre empresas: EDI
- Novas tecnologias de Integração.

III – Tecnologias Internet para a Integração:
- Arquitectura orientada aos serviços
- Web Services
- SML Schema

Aulas Práticas:

Elaboração de Exercícios Práticos utilizando:

- Ficheiros
- Sockets
- RPC/RMI
- JDBC
- XML
- JAXB
- Web Services (Java e .NET)

+ Sistemas de informação Geográfica;

+ Integração de Sistemas e Tecnologias (ex. usando python)

Bibliografia

- Silva, M. M., Integração de Sistemas de Informação, FCA Editora de Informática.
- O'Donahue, J., Java Database Programming Bible, John Wiley & Sons.
- Sperko, R., Java Persistence for Relational Databases, Apress.
- Rodrigues, L.S., Arquitecturas de Sistemas de Informação, FCA Editora de Informática.
- Linthicum, D., Enterprise Application Integration, Addison Wesley.

I -Integração de Sistemas e Tecnologias

■ **Um sistema de informação é um conjunto de aplicações integradas entre si**

- Duas ou mais aplicações estão integradas quando partilham os mesmos dados

Por exemplo, utilizando uma única base de dados

- A integração faz com que o **todo** (sistema de informação) seja maior que a soma das **partes** (aplicações)

■ **Caso de Estudo**: ERP

- Gestão Comercial Primavera
  - Facturação
  - Contabilidade
  - Salários

- Por outro lado, praticamente todas as empresas estão (mais ou menos) informatizadas hoje em dia possuem um ou mais sistemas de informação

- Além disso, existem inúmeras aplicações informáticas disponíveis no mercado:
  - Ainda por cima cheias de funcionalidades e com preços acessíveis (inclusive gratuitas)

    Por exemplo: ERP, CRM, Office, SFA, etc.
  - Mas criar novas aplicações também não é problema!

    UML, RUP, etc.

- **O grande desafio actualmente é integrar os sistemas de informação dentro e entre empresas**

Razões para Integrar Sistemas de Informação

- A quantidade e qualidade dos SI tem crescido exponencialmente
  - Tornando a integração cada vez mais interessante
- Cada vez mais faz sentido adquirir SI a terceiros
  - Mas esses SI não estão integrados
- As novas tecnologias obrigam a instalar novos SI
  - Exemplo: lojas on-line
- As novas tecnologias facilitam a integração
  - Exemplos: Internet, XML, certificados digitais
- Os ERP não resolvem todos os problemas das empresas

- As novas arquitecturas de SI (baseadas em processos de negócio) criam novas necessidades de integração

- A integração “é um mundo” por isso nada melhor que dividir o tema em várias partes

- Existem inúmeras formas de classificar a integração:

  - Por níveis conceptuais (como no OSI)
  - Por tipo de integração (directa/indirecta, ponto a ponto)
  - Por características tecnológicas (performance)
  - Por razões históricas (anos 80, 90, 2000)

- A **geografia da integração** oferece uma classificação abrangente:
  - Integração no computador
  - Integração na empresa
  - Integração entre empresas

Integração, Redes e Middleware

- **Redes** – permite trocar dados entre duas aplicações residentes em computadores distintos;Entenda-se “computadores” no sentido mais abrangente;

- **Middleware** – permite trocar dados entre duas aplicações, eventualmente com serviços adicionais;
  - Fornece uma API de alto nível, fiabilidade, performance, etc.

- **Integração de SI** – permite trocar dados entre várias aplicações de SI diferentes, normalmente sem programação;
  - Particularmente útil para integrar SI adquiridos a terceiros que não podem ser alterados;

Um sistema distribuído é um sistema de informação com aplicações residentes em vários computadores

Características da integração:

- **Facilidade** – no limite apenas com “drag & drop”

  - Importante para baixar os custos (i.e. reduzir os informáticos especializados) nos grandes projectos de integração

- **Transparência** – esconder os detalhes técnicos

  - Por exemplo, qual a tecnologia utilizada para trocar os dados, qual a API utilizada para introduzir os dados no SI, etc.

- **Aplicabilidade** – suportar um grande número de cenários

  - Comunicação ponto a ponto em tempo real, comunicação multi-ponto publish/subscribe, etc.

- **Fiabilidade** –lidar com as falhas das outras aplicações

  - Por exemplo, quando a rede “está em baixo” tentar mais tarde um determinado número de vezes e/ou durante certo tempo

- Características (cont.)
  - **Performance** – rápido e escalável

    - Em particular, tirar partido das economias de escala em situações do tipo: mesma mensagem para várias aplicações, muitas mensagens de/para a mesma aplicação, etc.
  - **Segurança** – gestão centralizada

    - Inclui autenticação de utilizadores, autorização (controlo de acessos) dos utilizadores e protecção (para evitar o acesso não autorizado)

    - Particularmente interessante entre empresas
  - **Gestão** – normal e excepções

    - Por exemplo, monitorizar a troca de mensagens, detectar se alguma ligação está quebrada, obter estatísticas e controlar os servidores em excesso para tolerância a falhas

• Tipos de integração de Sistemas de Informação

- Outra forma de classificar a integração de SI é conforme o "nível" ou "camada" onde é realizada essa integração
  - Inspirada na arquitectura em 3 camadas dos SI
    - Classificação preferida pelo David Linthicum
- Níveis ou Camadas
  - **Dados** – dados estão guardados numa base de dados
  - **Lógica** – a troca de dados é escrita numa determinada linguagem de programação, como COBOL, Basic ou Java
  - **GUI** – baseada na interface (Web) com os utilizadores
  - **Portal** – interface (Web) única para todos os utilizadores acederem aos vários SI da empresa
  - **Processos de negócio** – camada que esconde os SI

- **Orientado aos Dados**
  - Acesso (normalmente remoto) a uma base de dados

    - Pode incluir dados replicados, actualizações numa (ou várias) bases de dados, bases de dados não relacionais, acesso aos dados via componentes, etc, etc.
  - Muito simples tanto em Windows como em Java

    - Com ODBC e/ou JDBC
  - Não é aconselhável por vários motivos!

    - Inserir dados numa BD é difícil e perigoso
    - Os tipos de dados podem ser diferentes
    - Os dados não são validados pela aplicação
    - Dados replicados podem ficar inconsistentes
    - Obriga a alterar o código da aplicação

- **Orientado aos Métodos**
  - Acesso directo à lógica da outra aplicação

    - As tecnologias tradicionais (COM) apenas suportam acesso local mas as novas tecnologias (EJB, Web Services) já suportam tanto acesso local como remoto
  - Muito fácil de utilizar nas linguagens baseadas em procedimentos, objectos e/ou componentes
  - Mas existem vários problemas, sem sempre claros!

    - É necessário alterar o código do cliente, e muitas vezes também o código do servidor

    - Baseia-se normalmente numa comunicação síncrona (em tempo real) que é inadequada na maioria das situações

    - O acesso remoto é sempre lento, ineficiente e pouco fiável

- **Orientado às Interfaces**
  - Acesso à interface com o utilizador da outra aplicação seja terminal, Windows ou Web
  - Adequado para integrar aplicações legadas onde não seja possível (ou aconselhável) alterar o código:
    - Por exemplo, programas COBOL em *mainframes*
  - Vantagens
    - Não acede directamente aos dados
    - Não é necessário alterar a outra aplicação
  - Problemas
    - Limitada a tempo real e ponto a ponto
    - Não é trivial simular um utilizador numa aplicação
    - É difícil fazer *screen scraping* a interfaces Windows
    - A interface com o utilizador normalmente é bastante volátil

- **Orientado aos Portais**
  - Integração "virtual" ao nível da interface com o utilizador

    - As interfaces Web são mais adequadas

      Exemplos: CNN, Website do DEI

    - Um portal oferece aos utilizadores uma espécie de "Meta SI" que serve de porta de entrada para todos os SI da empresa
  - Fácil de construir
  - Existem enormes problemas com esta abordagem

    - Não se trata de uma integração verdadeira

    - Os dados não são partilhados entre SI

    - Não é fácil conseguir uma interface única

    -Não existe um esquema único de segurança
  - Nota: David Linthicum criou esta classificação no auge da popularidade da Web

- **Orientado aos Processos**
  - Integração ao nível do negócio, não ao nível tecnológico

    - Os principais processos de negócio atravessam várias áreas funcionais (departamentos) das empresas

    - Mas os SI normalmente são verticais porque foram desenhados para suportar uma única área funcional da empresa (por exemplo, operações, vendas ou salários)

    - Por isso, os processos de negócio são ideais para detectar as necessidades de integração entre SI
  - Actualmente é difícil suportar esta abordagem na prática
  - Mas existem algumas aproximações:

    - Workflow – produtos para “gestão documental” que permitem desenhar o fluxo de documentos digitalizados

    - Message brokers – produtos para integrar diversos SI que já oferecem mecanismos para modelizar processos de negócio

- A Internet é “apenas” uma tecnologia de redes

  -Redes já existem há dezenas de anos

- A Web é “apenas” uma tecnologia de interface com o utilizador – ainda por cima de fraca qualidade

- Mesmo assim vale a pena falarmos da Internet – de outra forma ninguém comprava o livro!

- Pequena História

  1993 – Web

  1995 – Java

  1998 – XML

  2000 – Web Services

(World Wide) Web:

- Apenas uma tecnologia para interface com o utilizador
  - À partida tem pouco interesse para integrar SI
  - Além disso já entrou na sua fase decadente
- Durante uns tempos serviu para fazer *screen scraping*
  - Embora tenha caído em desuso com os Web Services
- Também pode ser utilizada em portais
  - O que pode ter algum interesse...
- O maior impacto foi na disponibilidade do HTTP
  - Protocolo de comunicações para trocar páginas Web
  - Pode ser utilizado para pedir e enviar qualquer tipo de dados
  - Robusto e simples, embora ineficiente
  - Tem a grande vantagem de “furar” firewalls

Linguagem Java:

- Apenas mais uma linguagem de programação
  - À partida também tem pouco interesse para integrar SI
  - A linguagem de programação é cada vez mais irrelevante
- O Java não é uma linguagem mas sim um "framework" que inclui inúmeras tecnologias de informação inovadoras
  - Exemplos: componentes (EJB), objectos distribuídos (RMI), código móvel (Applets), servidores aplicacionais (J2EE), acesso a base de dados (JDBC), mensagens assíncronas (JMS), acesso a aplicações legadas (JCA), etc, etc, etc.

- Muitas destas tecnologias podem ser utilizadas (directa ou indirectamente) para integrar SI
- Com a resposta da Microsoft (.NET) em breve será irrelevante tanto a linguagem como o sistema operativo

■ XML e Web Services

■ XML é uma linguagem para formatar documentos

- Na realidade é uma meta-linguagem para criar linguagens

  Cada linguagem define um tipo de documentos que pode ser construído com essa linguagem (por exemplo, encomendas)

- Em 1998 o XML começou a ser utilizado para formatar dados na integração entre SI -
- Simples, fácil de aprender e utilizar, adequado tanto para aplicações como para ser transformado em HTML

■ O termo "Web Services" agrupa um conjunto de protocolos para trocar XML pela Internet, normalmente utilizando HTTP

- A norma principal chama-se SOAP, mas existem outras
- Utilizado normalmente para invocar serviços remotamente

  Basicamente uma (pequena) evolução dos objectos distribuídos

II –Tecnologias de Integração

Resumo:

- Ficheiros;
- Sockets;
- Bases de Dados;
- Monitores Transaccionais;
- Componentes;
- Servidores Aplicacionais;

Informação Base

- Universal - suportado por todos os sistemas operativos e linguagens de programação.

- Simples - uma sequência de *bytes*.

- Sequência de eventos.

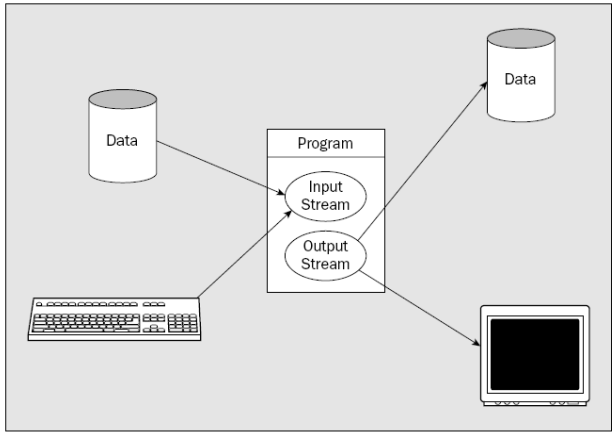
  - Codificação – objecto para ficheiro
  - Descodificação – ficheiro para objecto

- A complexidade da codificação e descodificação aumenta exponencialmente com a complexidade do objecto

  - Por isso apenas podem ser trocados objectos muito simples

- Muito utilizado na prática!

Informação Base



■ Streams:

- Uma stream é um conjunto de bytes que pode ser enviada para ou pelo programa.

- Uma ***stream*** é uma representação abstracta de um dispositivo de entrada ou saída que é a fonte ou destino dos dados. Podem escrever-se dados para uma ***stream*** ou efectuar a leitura de dados da ***stream***. Podemos visualizar uma ***stream*** como uma sequência de bytes que flui de/para o programa.

- Quando se escrevem dados numa ***stream***, a ***stream*** é do tipo *output stream*. A output stream pode ir para qualquer dispositivo para o qual pode ser transferida uma sequência de bytes, como por exemplo, um ficheiro (e.g. ***FileOutputStream*** - envio de dados para um ficheiro através de uma stream)

- A razão principal para usar ***streams*** como base para operações de input/output é tornar a codificação destas operações independente do dispositivo envolvido.

Class	Description
`InputStream`	The base class for byte stream input operations.
`OutputStream`	The base class for byte stream output operations.

Informação Base

```
 1  import java.io.*;
 2
 3  public class Terminal {
 4
 5      public static void main(String[] args) throws IOException
 {
 6          int b;
 7
 8          while ((b = System.in.read() ) != -1) {
 9             System.out.print((char)b);
10          }
11      }
12  }
```

Informação Base

```
 1 import java.io.*;
 2
 3 public class WriteFile {

 4
 5     public static void main(String[] args) throws
 6                                             IOException{
 7         String text;
 8         int i;
 9
10         File outputFile = new File(''File1.txt");
11         FileWriter out = new FileWriter(outputFile);
12         for (i=1; i<11; i++) {
13             text = "Line " + i + " of text\n";
14             out.write(text);
15         }
16         out.close();
17     }
18  }
```

Informação Base

```
 1  import java.io.*;
 2
 3  public class WritePrintFile {
 4
 5      public static void main(String[] args) throws
 6                                              IOException{
 7          String text;
 8          int i;
 9
10          File outputFile = new File("File1.txt");
11          FileWriter out = new FileWriter(outputFile);
12          PrintWriter p = new PrintWriter(out);
13          for (i=1; i<11; i++) {
14            p.println("Line " + i + " of text");
15          }
16          p.close();
17          out.close();
18      }
19  }
```

Informação Base

ReadFile

```
 1  import java.io.*;
 2
 3  public class ReadFile {
 4
 5      public static void main(String[] args) throws
 6                                                 IOException {
 7          File inputFile = new File(''File1.txt");
 8          if ( ! inputFile.exists() ) {
 9              System.out.println("File does not exist");
10              System.exit(1);
11          }
12          FileReader in = new FileReader(inputFile);
13          int c;
14          while ((c = in.read()) != -1){
15              System.out.print( (char) c );
16          }
17          in.close();
18      }
19  }
```

Informação Base

ValueTooSmallException

```
1  public class ValueTooSmallException extends
2                                          Exception {
3      public ValueTooSmallException(){}
4
5      public ValueTooSmallException(String message) {
6          super(message);
7      }
8  }
```

Informação Base

Account

```
 1  import java.io.*;
 2
 3  class Account implements Serializable {
 4      int accountNo;
 5      String accountName;
 6      double balance;
 7
 8      Account(int accountNo, String accountName, double balance
)
 9                                      throws ValueTooSmallExcepti
on {
10          this.accountNo = accountNo;
11          this.accountName = accountName;

12        if (balance < 0) {
13            throw new ValueTooSmallException(''Negative Balance
");
14        else {
15          this.balance = balance;
16        }
17     }
18  }
```

Informação Base

WriteAccount

```
 1  import java.io.*;
 2
 3  class WriteAccount {
 4
 5      public static void main(String[] args) throws
 6                      IOException, ValueTooSmallException {
 7          Account account1 = new Account(1, "account1", 40);
 8          Account account2 = new Account(2, "account2", 100);
 9          FileOutputStream out = new FileOutputStream("acc.dat"
);
10          ObjectOutputStream outob = new ObjectOutputStream(out
);
11          outob.writeObject(account1);
12          outob.writeObject(account2);
13          outob.close();
14          out.close();
15      }
16  }
```

Informação Base

ReadAccount

```
 1  import java.io.*;
 2
 3  class ReadAccount {
 4
 5      public static void main(String[] args) throws
 6                          IOException, ClassNotFoundException {
 7          FileInputStream in = new FileInputStream(''acc.dat");
 8          ObjectInputStream inobj = new ObjectInputStream(in);
 9          Account acc1 = (Account) inobj.readObject();
10          Account acc2 = (Account) inobj.readObject();
11          System.out.println(" 1st number : " + acc1.accountNo);
12          System.out.println(" 2nd balance : " + acc2.balance);
13          inobj.close();
14          in.close();
15      }
16  }
```

Dificuldades:

- As vantagens do Java normalmente não existem noutras linguagens de programação

- Pode ser utilizado para (facilmente) trocar dados apenas entre aplicações escritas na mesma linguagem

- A codificação pode ser lenta

- A codificação de um objecto pode envolver uma enorme quantidade de objectos (no limite, toda a base de dados!)

Existem objectos que não podem ser codificados na origem e/ou recriados no destino

O objecto no destino é outro objecto! (embora tenha os mesmos valores quando é criado)

Não oferece nenhuma das funcionalidades das tecnologias mais avançadas

- Suporta a troca de dados em forma de *bytes* directamente entre aplicações

  - Pode ser utilizado para trocar *strings* ou qualquer outro tipo de dados desde que codificado numa sequência de *bytes*

- Exemplo: Java

  - Os objectos podem ser codificados e descodificados automaticamente, tal como vimos no caso dos ficheiros

- Dificuldades: Basicamente as mesmas dos ficheiros!

- Comparação com ficheiros: Ficheiros são assíncronos enquanto os sockets são síncronos, o que tem vantagens e desvantagens

Comparação Síncrono VS Assíncrono:

Sockets/Síncrono

Entrega imediata (se o receptor estiver disponível)

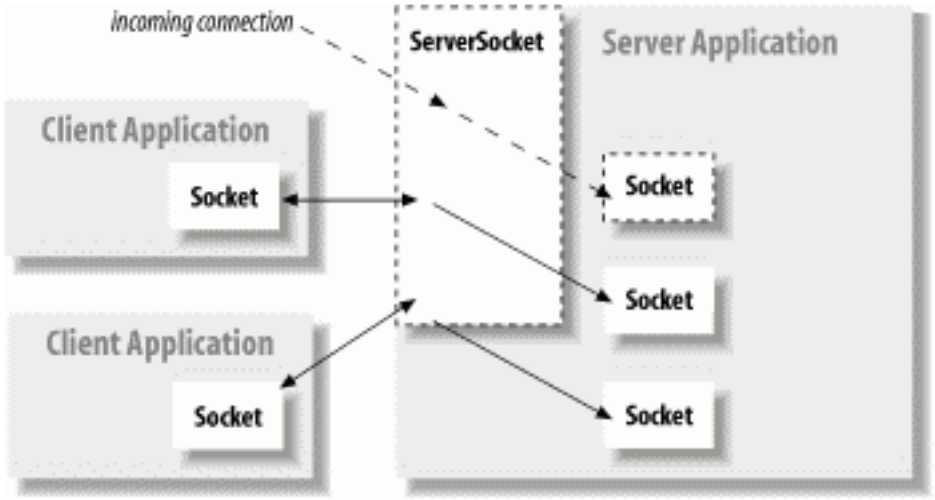
Confirmação automática que os dados foram entregues

Ficheiros/Assíncrono

- O emissor (também designado cliente) não depende da disponibilidade do receptor (servidor) nem da rede.
- O emissor não interrompe o servidor quando quer enviar dados
  - O servidor é que vai ler o ficheiro quando quer receber dados
- Pode tirar partido das economias de escala (por exemplo, enviar muitos dados no mesmo ficheiro).
- O facto da entrega não ser imediata não quer dizer que não seja rápida (por exemplo, em menos de um segundo)
- A confirmação pode ser feita facilmente

- Sockets

Informação Base



- Sockets

Informação Base

```
try {
    Socket sock = new Socket("wupost.wustl.edu", 25);
} catch ( UnknownHostException e ) {
    System.out.println("Can't find host.");
} catch ( IOException e ) {
    System.out.println("Error connecting to host.");
}
```

- Sockets

Informação Base

```
try {
    Socket server = new Socket("foo.bar.com", 1234);
    InputStream in = server.getInputStream( );
    OutputStream out = server.getOutputStream( );

    // write a byte
    out.write(42);

    // write a newline or carriage return delimited string
    PrintWriter pout = new PrintWriter( out, true );
    pout.println("Hello!");

    // read a byte
    byte back = (byte)in.read( );

    // read a newline or carriage return delimited string
    BufferedReader bin =
      new BufferedReader( new InputStreamReader( in ) );
    String response = bin.readLine( );

    // send a serialized Java object
    ObjectOutputStream oout = new ObjectOutputStream( out );
    oout.writeObject( new java.util.Date( ) );
    oout.flush( );

    server.close( );
}
catch (IOException e ) { ... }
```

Informação Base

```
// Meanwhile, on foo.bar.com...
try {
    ServerSocket listener = new ServerSocket( 1234 );

    while ( !finished ) {
        Socket client = listener.accept( );  // wait for connecti

        InputStream in = client.getInputStream( );
        OutputStream out = client.getOutputStream( );

        // read a byte
        byte someByte = (byte)in.read( );

        // read a newline or carriage-return-delimited string
        BufferedReader bin =
          new BufferedReader( new InputStreamReader( in ) );
        String someString = bin.readLine( );

        // write a byte
        out.write(43);

        // say goodbye
        PrintWriter pout = new PrintWriter( out, true );
        pout.println("Goodbye!");

        // read a serialized Java object
        ObjectInputStream oin = new ObjectInputStream( in );
        Date date = (Date)oin.readObject( );

        client.close( );
    }
```

- Os ficheiros podem ser considerados bases de dados extremamente simples, e vice-versa

- As bases de dados têm enormes vantagens:
  - Suportam naturalmente a troca assíncrona de dados
  - São sistemas robustos e "nunca" perdem dados
    - Ao contrário dos protocolos de redes (e.g. Mail)
  - Normalmente já existe uma base de dados
    - Todas as aplicações normais precisam de bases de dados
    - A base de dados tem sempre a última versão dos dados
    - Os dados para trocar costumam estar numa base de dados
  - Oferecem API simples, conhecidas e universais (xDBC)
    - Que podem ser utilizadas remotamente
  - Suportam um modelo (relacional) simples e conhecido

Outras Vantagens

- As bases de dados são sistemas cada vez mais baratos e no futuro serão parte do próprio sistema operativo
  - Linux, Longhorn
- As *stored procedures* permitem executar aplicações dentro da própria base de dados
  - As aplicações poderão assim ser partilhadas para evitar a troca (e duplicação) dos dados
- O armazenamento de objectos dentro das bases de dados evita a codificação/descodificação de/para relacional
  - Embora poucas aplicações actualmente utilizem esta funcionalidade devido à simplicidade do modelo relacional

- **Linguagem SQL**
  - Utilizada no acesso às bases de dados relacionais
    - Tem a grande vantagem de estar normalizada
- **Interfaces de Acesso**
  - Proprietárias
    - Exemplo: Oracle
      - Flexível e potente, mas complicada
  - Normalizadas
    - Exemplo: ODBC (Windows)
      - Muito simples, mas pouco eficiente
    - Exemplo: JDBC (Java)
      - Ainda mais simples

■ **Produtos Comerciais**: Existem inúmeros produtos para aceder a base de dados com funcionalidades avançadas, incluindo:

- Data Integration
- Data Transformation
- Data Warehouses
- ETL
- Metadata
- Data Access/Connectivity
- Data Integrity
- Data Cleansing
- Data Replication

Informação Base

- **Transacção**:
  - Uma aplicação composta por várias operações
  - Rápida;
  - Concorrente;
  - Tudo ou nada;
  - Escalável;
  - Durável;
  - Distribuída;

- Não confundir com transacção de base de dados

  - Estas transacções podem aceder a várias bases de dados ou não aceder a nenhuma, mas podem usar as transacções das bases de dados.

- Monitor Transaccional

  Um motor (*engine*) de execução de transacções

- Funcionalidades

  - Servidor de apresentações
  - Controle de workflow
  - Acesso a base de dados
  - Comunicações com outros MT
  - Gestão de transacções

  - E ainda disponibilidade, performance e escalabilidade!

Relação com integração

- Servem para integrar aplicações escritas em linguagens diferentes, eventualmente em MT diferentes

- Cada programa é uma transacção mas pode ser chamado por, e fazer parte de, outra transacção

- Acede virtualmente a qualquer base de dados de forma eficiente (partilha das conexões)

- Suporta comunicação transaccional com outras aplicações, eventualmente noutro MT

- Garante que todas estas funcionalidades não comprometem a disponibilidade, performance e escalabilidade

- **Dificuldades**
  - "Rei das mainframes": Imagem bastante degradada
  - Pode ser substituído por outras tecnologias em ambientes menos exigentes (que são a grande maioria)
    - **Bases de Dados**

      Também suportam transacções, executam aplicações, acedem a outras bases de dados, e fornecem interfaces com o utilizador
    - **Componentes**

      Os componentes modernos (COM+, EJB) também são baseados em transacções e suportam basicamente as mesmas funcionalidades
    - **Servidores Aplicacionais**

      Os componentes ainda mais modernos (.NET, J2EE) fornecem um ambiente completo para construção e execução de aplicações baseadas em transacções

Informação Base

- O que são componentes
- Conceito anterior aos procedimentos e objectos

  Um programa COBOL pode ser considerado um componente
  - Mas simultaneamente um conceito posterior

    Um componente é composto por vários objectos que por sua vez oferece vários métodos/procedimentos
  - Componente não é objecto!

    Objectos pode haver muitos da mesma classe, cada qual com os seus próprios dados (o que tem vantagens e desvantagens)

- **Definição**:

  Reusable program building block
  - Nem todos os componentes podem se acedidos remotamente
  - Por isso vamos assumir que quando dizemos “componente” não estamos a considerar essa funcionalidade

Informação Base

História na Microsoft:

- Tudo começou para aceder a documentos de outro tipo dentro de um documento
  - Por exemplo, uma folha Excel numa página Word
  - Para isso ser necessário era preciso colocar o Word a comunicar com o Excel
  - A Microsoft copiou a ideia da Apple e chamou-lhe DDE

    Tecnicamente utiliza memória partilhada
- COM aparece como um modelo de "objectos"

  A tecnologia chamava-se OLE
- ActiveX aparece como resposta aos Applets

  Generalização do COM que permite trocar componentes (agora chamados controlos) pela Internet

- DCOM aparece como resposta ao CORBA

  - Extensão ao COM para aceder remotamente aos componentes

- COM+ aparece como resposta ao EJB

  - Extensão ao COM para suportar segurança, notificação e (principalmente) transacções

- .NET
  - Resposta ao "desafio" da Internet, XMl e Web Services
  - O COM nunca foi pensado para ser chamado remotamente nem suportava XML
  - O .NET é uma estratégia ou visão

    - Não apenas uma tecnologia ou mesmo um modelo

- Apenas consideramos componentes tradicionais o DDE, OLE e COM+

História no Java

- O Java não é uma linguagem, é uma plataforma completa para construção e execução de aplicações.

- JavaBeans
  - Aparece em 1996, logo a seguir ao próprio Java (1995) que já era orientado por objectos
  - COM está limitado ao Windows, JavaBeans ao Java
- EJB
  - Acrescenta transacções, segurança e acesso a base de dados
- J2EE
  - Comparável ao .NET porque ambas pretendem substituir o sistema operativo
  - As aplicações no futuro não serão “Java” mas sim “J2EE”
  - E tanto podem estar num Windows como num Linux

- São basicamente monitores transaccionais com suporte para as novas tecnologias da Internet
  - Oferecem o mesmo tipo de funcionalidades

    - Transacções, acesso a bases de dados, acesso a outros servidores, disponibilidade, performance e escalabilidade
  - Correm em sistemas operativos de nível médio

    - UNIX e Windows

- Exemplos:

  - Produtos compatíveis J2EE

    - WebSphere, Weblogic, etc
  - Microsoft MTS

Exemplo: J2EE

- Um conjunto de interfaces programáticas (API) para construir e executar aplicações em 4 camadas:

- **Enterprise Information System** – bases de dados (JDBC) e pacotes de gestão empresarial (JCA)
- **Server-side Business Logic** – organizada em componentes (EJB) que podem ser acedidos localmente ou remotamente
- **Server-side Presentation** - páginas HTML (JSP ou Servlets) e/ou Applets
- **Client-side presentation** – *browser* e/ou aplicações Java

- Exemplos de Produtos
  - Sun ONE
  - IBM WebSphere
  - JBOSS

Índice:

Informação Base

Motivação:

- As ligações directas "ponto a ponto" só funcionam quando o número de aplicações é pequeno
  - Só funcionam para poucas aplicações
    - 2 aplicações à 2 * 1 / 2 = 1 ligação
    - 10 aplicações à 10 * 9 / 2 = 45 ligações
    - 50 aplicações à 50 * 49 / 2 = 1225 ligações!
- Solução: usar "filas de espera" (queues) baseadas em canais de comunicação (bus)
  - Existe apenas um canal onde todas as aplicações se ligam
    - Para 50 aplicações apenas são precisas 50 ligações!
  - Para cada aplicação o canal tem duas filas de espera:
    - Uma para enviar mensagens para essa aplicação
    - Outra para receber mensagens dessa aplicação

Informação Base

Informação Base

Implementação:

- As aplicações comunicam entre si trocando mensagens indirectamente através do canal
  - As mensagens normalmente são assíncronas
    - O emissor não fica à espera que a mensagem seja entregue
      - Esta abordagem é rápida, eficiente, escalável e robusta
      - Além disso, o canal "garante" que a mensagem será entregue
  - Mas, as mensagens também podem ser síncronas!
    - Por exemplo, quando é preciso obter a resposta imediatamente
      - Em geral isso deve ser evitado ao máximo porque uma aplicação não deve ficar dependente da rede e/ou de outra aplicação!

- O emissor envia a mensagem para a **fila de saída** no canal que depois coloca na **fila de entrada** do receptor

Informação Base

- O receptor deve contactar o canal para receber as mensagens que estão na sua **fila de entrada**

  - Desta forma o receptor não está constantemente a ser interrompido com novas mensagens. Mas, o receptor pode pedir ao canal para avisar sempre que tiver mensagens para entrega

- As mensagens são apropriadas quando:

  - A rede não é fiável e/ou não está sempre ligada

    Por exemplo, empresas dispersas geograficamente

  - Essa integração é crítica

    Por exemplo, ordens de débito e crédito

  - Existem muitas mensagens

Informação Base

Canal de comunicação:

- É preciso distinguir o "conceito" da "implementação"
- Conceito
  - Canal único (BUS)
- Implementação
  - Pode estar distribuído pelas aplicações
  - Pode ter (ou não) um "servidor"
  - Pode ser replicado para garantir maior robustez e/ou performance

  Etc.
- Em grandes empresas o "canal" pode estar organizado numa hierarquia, tanto em termos conceptuais como em termos de implementação

- **Funcionalidades**:
  - Suporta a troca de mensagens indirecta entre aplicações
    Em principio assíncronas (ver próximo slide)
  - Compatível com qualquer tipo de:
    Rede
    Sistema operativo
    Linguagem de programação
    Monitor transaccional
  - Configurável – na instalação
  - Oferece uma API – para as aplicações
  - Controlável – pelo administrador
  - Eficiente, robusto, escalável, seguro, etc

Informação Base

- Porquê mensagens assíncronas (*deferrable*) ?

  - Porque oferecem fiabilidade na integração!
    - Mesmo quando a rede e/ou a outra aplicação não é fiável
  - Porque são muito eficientes
    - Enviar uma mensagem é muito rápido e "barato"
    - E quanto mais mensagens mais rápido e barato

      Quando dividido pelo número de mensagens
  - Porque o "servidor" não fica escravo do "cliente"
    - As mensagens são processadas no servidor quando convém ao servidor, não quando convém ao cliente
  - Porque as filas de espera podem ter nomes lógicos e ficar assim independentes da rede física e/ou servidores

    Por exemplo "OrderReception"

Informação Base

Outras Funcionalidades:

- Confirmação de entrega
- "Garantia" de entrega
- Tempo limite de entrega (*timeout*)
- Entregas múltiplas (*multicast*)
- Leitura selectiva e/ou prioritária
- Comunicação transaccional

  - Quando uma transacção envia uma mensagem, essa mensagem só é efectivamente enviada se/quando essa transacção terminar à isto não está no livro!

  - Um conjunto de mensagens será todo entregue ou nenhuma mensagem desse conjunto será entregue

    - Mesmo quando existem aplicações dispersas geograficamente!

- **Desvantagens**:

  - Todos os produtos utilizam formatos de mensagens e protocolos de comunicação proprietários

    -Todas as aplicações são obrigadas a utilizar o mesmo produto

  - Os produtos tradicionais apenas trocam *bytes*

    - Não codificam ou descodificam a mensagem automaticamente

      O programador tem mais trabalho e pode cometer erros

      A produtividade é largamente diminuída

      Limita seriamente a complexidade das mensagens

    - Provavelmente contribuiu para o aparecimento dos procedimentos remotos

      Que simplificam a vida do programador

    - Mas nada tem a ver com a teoria das mensagens!

Informação Base

Desvantagens:

- Troca assíncrona
  - Mais apropriada para processamento em *batch*

    - Que nos anos 80 começou a cair de moda
  - Ao contrário do fenómeno "cliente/servidor" – computadores pessoais interligados através de redes locais

    -Que popularizou o acesso em tempo real aos dados

- Falta de normas sobre mensagens
  - Tem muitas desvantagens

    - Diminui a flexibilidade, aumenta os preços, reduz a visibilidade, dificulta a formação dos técnicos, etc.
  - Existem (muitas) normas sobre integração

    - Procedimentos remotos, objectos distribuídos, Web Services, etc.

- **Exemplo: JAVA JMS API**

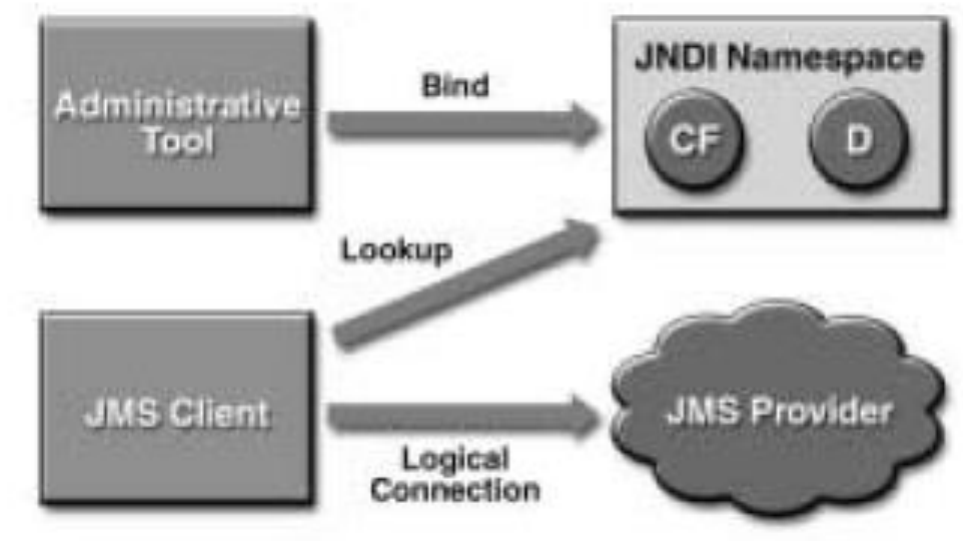
  - A Java Message Service é uma API que permite às aplicações a criação, envio, recepção e leitura de mensagens.

  - A API JMS permite comunicação:

  - Assíncrona: O JMS provider envia mensagens para um cliente quando elas chegam; um cliente não necessita de requisitar mensagens para as receber;

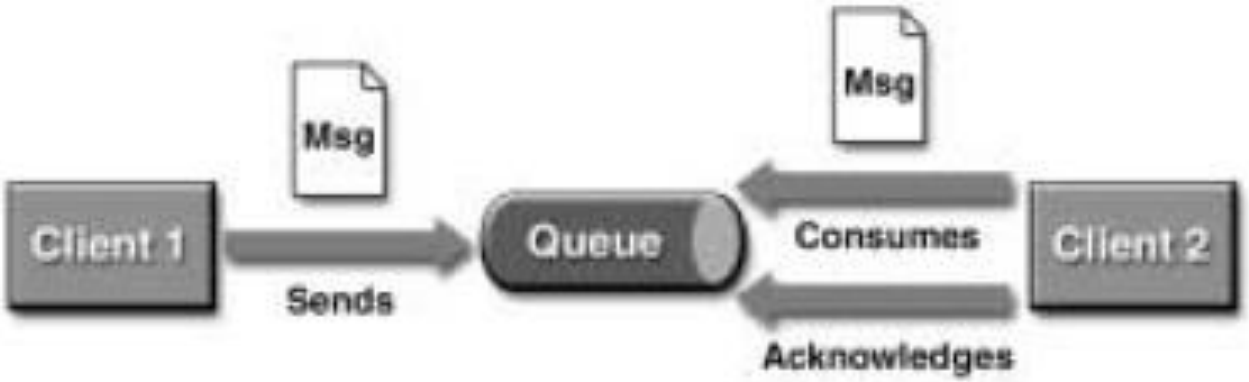
    Reliable: A JMS API pode assegurar que uma mensagem é enviada uma e apenas uma vez.

- Exemplo: JAVA JMS API



JMS API Architecture

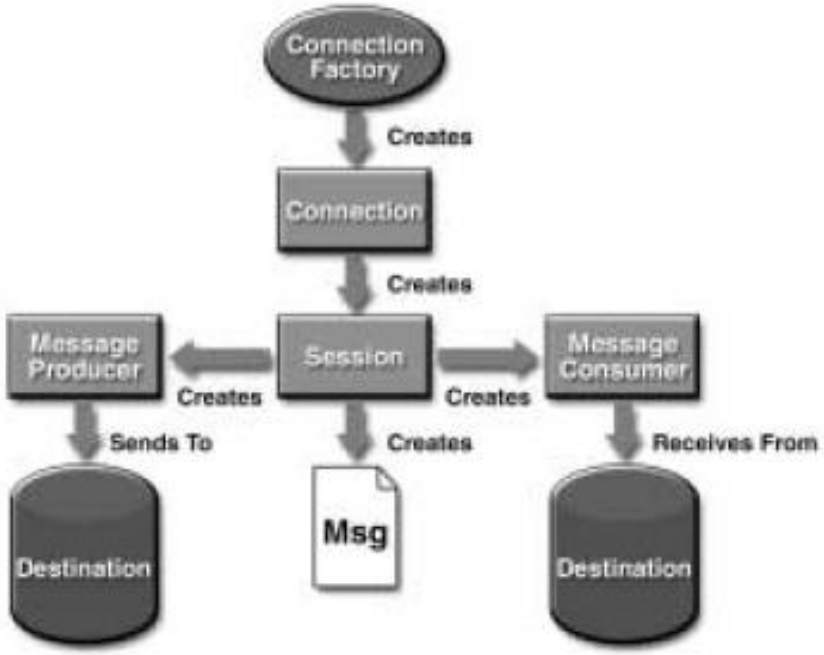
- Exemplo: JAVA JMS API



- Exemplo: JAVA JMS API

Informação Base

- Exemplo: JAVA JMS API



• Procedimentos Remotos - RPC

Informação Base

• **Bibliografia**:

- *Integração de Sistemas de Informação – FCA*

- Professional Services Web XML - Wrox

• Hoje em Dia:

PC isolado

- Não partilha de recursos
- Orientado ao Procedimento
- Sockets – Comunicação inter-processos
- Cliente Servidor – RPC – abstracção da localização

PC ligado em Rede

- Orientado a Objectos
- CORBA, RMI e COM/DCOM
- Sistemas Distribuídos

PC "ligado" via Web

- Web Services – EAI/IAC/B2B/etc…

Middleware – Integração ao nível dos dados…(nível 5-7 OSI):

Tecnologia/Mecanismo que possibilita a comunicação entre duas entidades

- Pretende “esconder” a complexidade do processo de “comunicar”
- Pode implicar alterar aplicações já existentes
- Actualmente procura-se evitar “mexer” nas aplicações existentes
- Hoje o cenário é a empresa no seu todo ou mesmo entre empresas: EAI e B2B

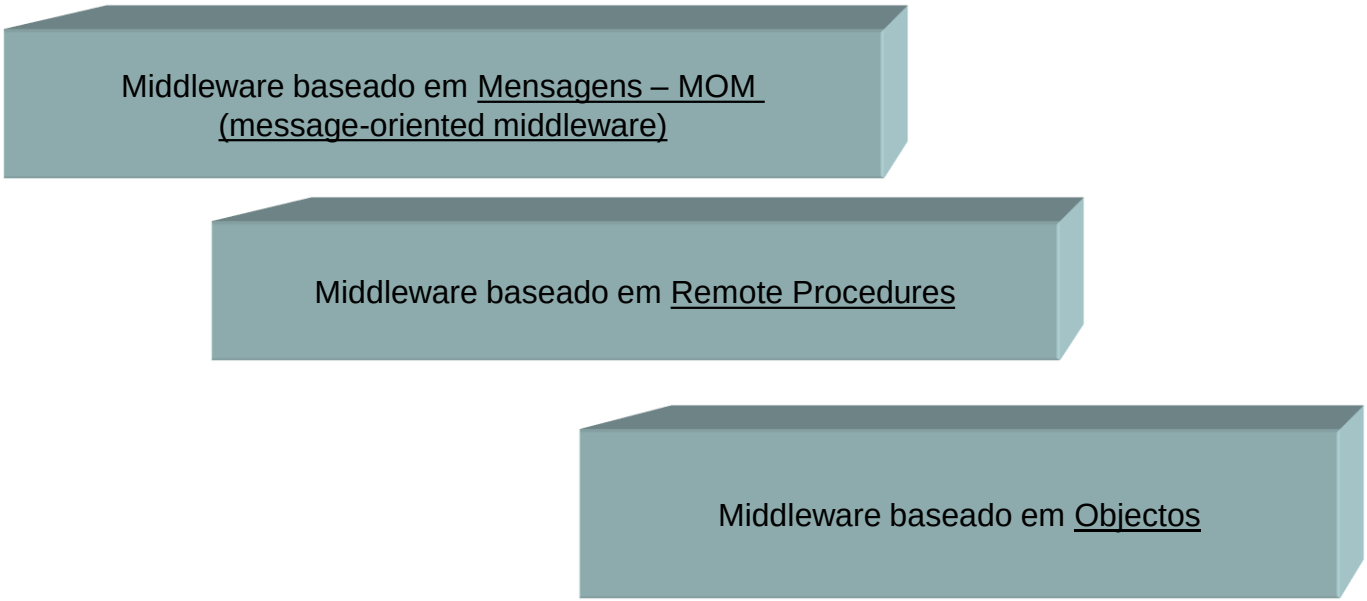
• Procedimentos Remotos - RPC

Informação Base

- Aparece nos anos 80
  - Época da popularização das redes locais
  - Extensão natural ao conceito de procedimento
- Serve de base para as "novas" tecnologias de integração
  - Por exemplo, Web Services
- Actualmente muito utilizada
  - Principalmente em redes locais
    - Acesso remoto a bases de dados
    - Base tecnológica dos servidores Web
  - Uma pesquisa no google devolve 500,000 referências!

Informação Base

Tipos de Middleware:



- Database-oriented Midddleware
- Transactional Middleware
- Message Brokers

Informação Base

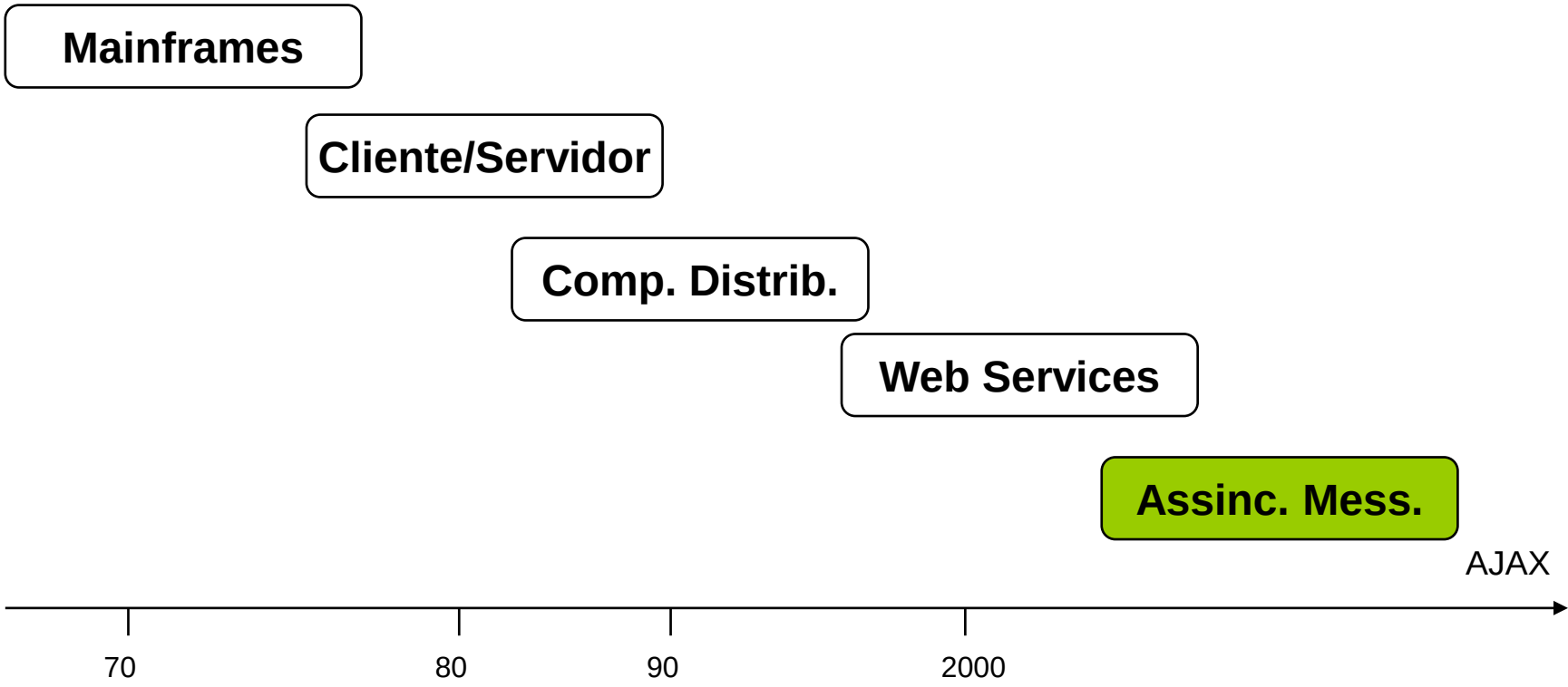
Middleware:

- *Modelos*
  - Lógico – perceber como a informação "passa" de uma entidade para outra
  - Físico - entender a tecnologia e princípios envolvidos
- *Configurações*
  - Ponto-a-Ponto : exemplo: RPC
  - Muitos-para-Muitos: mais complexo. Ex: DCOM, Message Brokers
  - Sincrono-Assíncrono
  - Com conecção: directo
  - Sem conecção: utiliza queues

Informação Base

Arquitecturas de Mensagens Assíncronas:

Evolução:



Informação Base

Middleware baseado em Remote Procedures:

. Um modelo Cliente-Servidor:

•No middleware baseado em mensagens sabíamos que:

•A aplicação é a unidade de distribuição

•Contudo a unidade ideal é o próprio procedimento

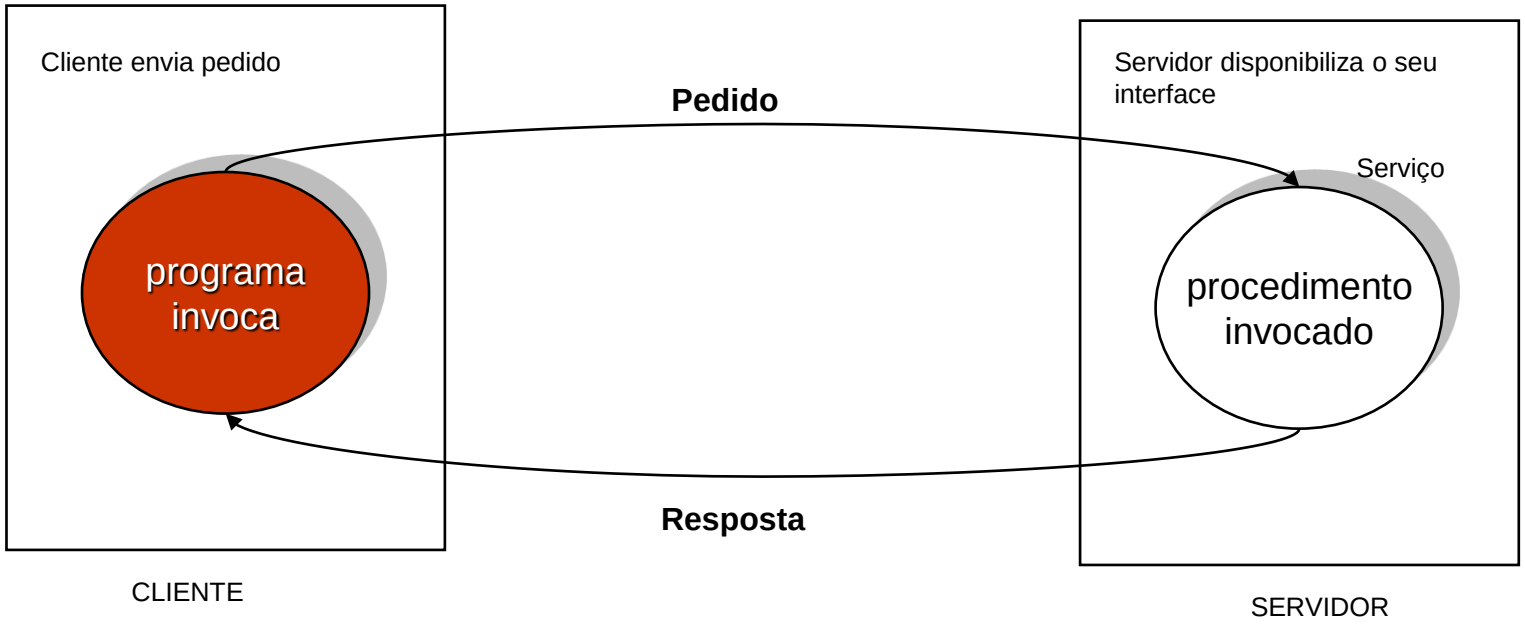
•Não há restrições para enviar mensagens (qualquer um pode)

•O RPC vem estipular regras de diálogo do tipo *Pergunta-Resposta*

Informação Base

Middleware baseado em Remote Procedures:

. O modelo Cliente-Servidor



Informação Base

Middleware baseado em Remote Procedures:

. *RPC – Remote Procedure Call*

- *Código das aplicações mantêm-se inalterável*
- *Código não se "preocupa" com Middeware*

•*Como funciona:*

- *O programa invoca um procedimento (remoto)*
- *Os procedimentos no server foram gerados (compilados + linkados)*
- *O server está "atento" à rede*
- *Os interfaces são as Assinaturas dos procedimentos* ***ex: int adiciona (int x, int y)***

Informação Base

Middleware baseado em Remote Procedures:

. *Arquitectura*

- O middleware é externo às aplicações
- ***Cliente RPC -***
  - ***Client Stub***
    - "falsos" procedimentos representam os reais (no server)
    - mesmas assinaturas
    - o código original é substituído por código middleware:
      - ligar à rede - connect
      - Pack e Envia os parâmetros
      - Recebe e UnPack o resultado

Informação Base

Middleware baseado em Remote Procedures:

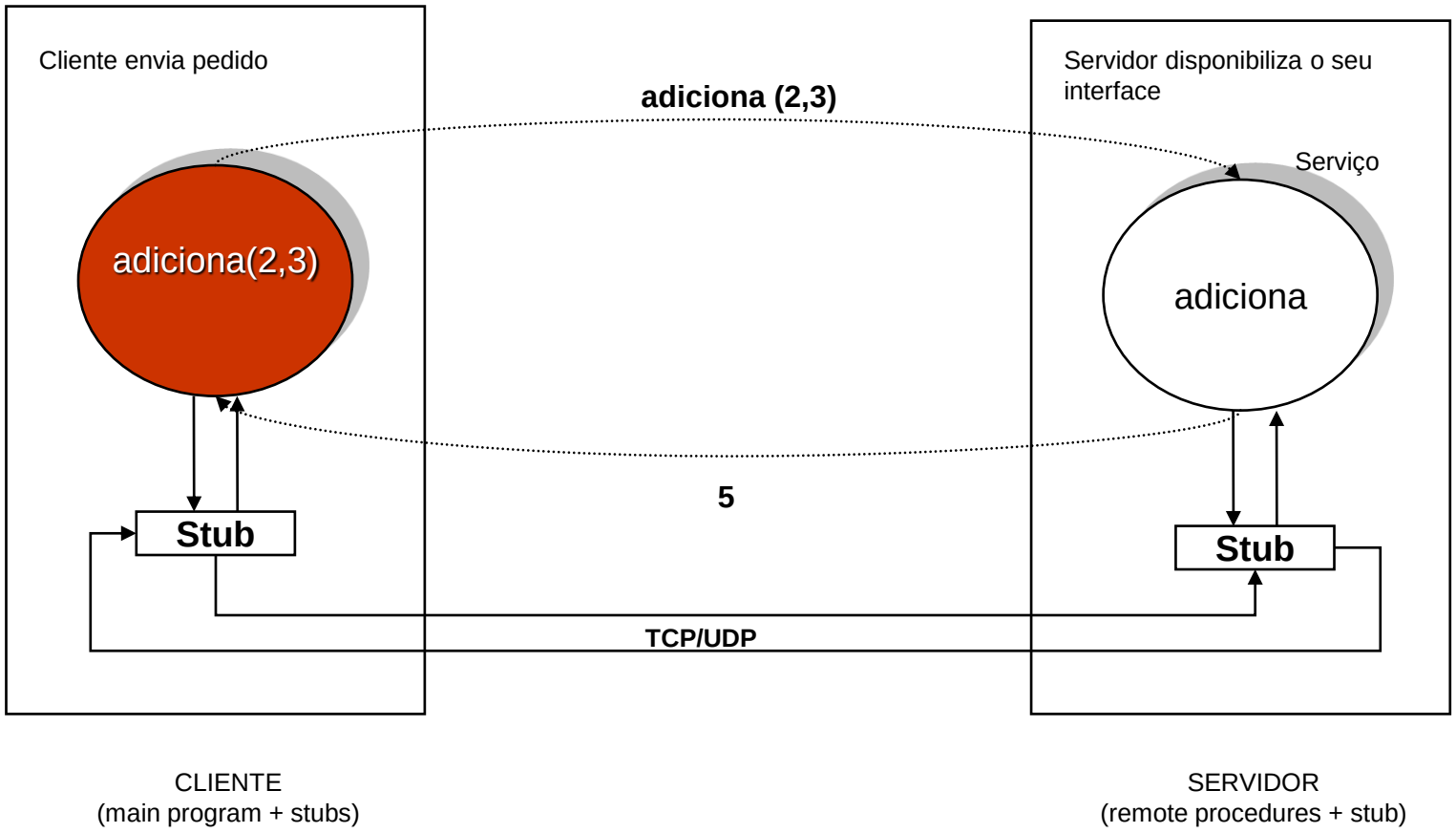
. *Arquitectura*

- ***Server RPC***
  - *Contém a implementação dos procedimentos*
  - ***Server Stub***
    - liga-se à rede e “escuta”
    - Recebe e UnPack os parâmetros
    - Escolhe o procedimento pretendido
    - Pack e Envia os resultados

• Procedimentos Remotos - RPC

Informação Base

Middleware baseado em Remote Procedures:



Informação Base

Middleware baseado em Remote Procedures:

- *"Contrato" entre Cliente-Servidor*
  - *Identificador único*
  - *Um interface IDL*
    - ***nome*** *e conjunto de* ***assinaturas*** *de procedimentos*
    - *Tipos de Dados*
    - *parâmetros caracterizados por* ***[in]*** ou ***[out]***

int adiciona ([in] int x, [in] int y, [out] int soma)

Informação Base

Middleware baseado em Remote Procedures:

- *Construção*
  - *Escrever 3 componentes:*
    - **Programa *main* (o cliente)**
    - **Conjunto de procedimentos (o servidor)**
    - **O interface (em IDL)**
  - *Depois:*
    - **Gerar os stubs do Cliente e Servidor**
    - **Gerar o cliente executável (com os stubs)**
    - **Gerar o servidor executável (com os stubs)**

Informação Base

Middleware baseado em Remote Procedures:

- *Ligação ao Servidor*
  - *Praticamente é automático*
  - *Baseado em "Nomes de Servidor" – um nome um IP*
  - *Como:*
    - **Deverá existir e ser conhecido o IP do Servidor de Nomes**
    - **O cliente pede o "Nome" para o Servidor X**
    - **O cliente liga-se ao servidor "Nome"**

Informação Base

Middleware baseado em Remote Procedures:

• *Diálogo*

•*O cliente assume que o Servidor está operacional. Caso contrário recebe ERROR MSG…*

•*Principais passos:*

1. Client program: evoca o procedimento romoto (localmente)
2. Client stub: codifica e envia a mensagem ao server
3. Server stub: recebe e descodifica a mensagem pedido
4. Server stub: evoca o procedimento (real)
5. Server stub: codifica e envia o resultado ao cliente
6. Client stub: recebe e descodifica a mensagem resultado
7. Client stub: devolve o resultado ao cliente

Comparação com as mensagens:

- Extremamente simples de perceber e utilizar
  - Semelhante às chamadas a procedimentos locais
- Menor granularidade
  - Ao nível do procedimento, não da aplicação
- Servidor tem de publicar os procedimentos que suporta
  - As mensagens podem ser enviadas a qualquer aplicação
- Servidor recebe as chamadas e executa o procedimentos quando o cliente decide
  - Nas mensagens a decisão do "quando"é feita pelo receptor se e quando quiser
- Codificação e descodificação das mensagens é realizado automaticamente

Codificação e Descodificação

- São realizadas automaticamente

  Talvez a maior vantagem dos RPC

- Oportunidade criada pelo facto de ser necessário conhecer previamente quais os programas e os procedimentos que podem ser chamados remotamente

  Nas mensagens isso não é necessário, tornando mais difícil (mas não impossivel) essa automatização

- Impacto do XML

  Esta vantagem dos RPC anula-se em grande parte

  A maioria das linguagens suporta XML directamente

  O esforço de codificação e/ou descodificação é muito menor

  Os dados em XML podem ser manipulados pela aplicação

Informação Base

- **Servidor de Nomes:**
  - Método manual

    - O cliente conhece a localização do servidor

  - Método automático
    - O servidor regista-se num "servidor de nomes"
    - O servidor publica os procedimentos que suporta
    - O cliente contacta o servidor de nomes para saber que servidores existem e que procedimentos suportam

      - O cliente saberá a localização do servidor de nomes?

    - O cliente envia o pedido directamente para o servidor que suporta o procedimento

      A ligação directa tem vantagens mas também desvantagens

      Impede que sejam prestados serviços de valor acrescentado

Informação Base

- **Vantagens**:
  - Simplicidade do conceito
  - Sincronismo da comunicação
    - Ideal para aceder a bases de dados departamentais
    - Elimina a necessidade de garantir a entrega da mensagem
    - Os clientes podem participar em transacções distribuídas
  - Granularidade dos procedimentos
  - Sinergia criada com o modelo cliente/servidor
    - Em voga desde meados dos anos 80
    - Mas fora de moda (outra vez) com o aparecimento da Web
  - Normas (DCE)

- **Desvantagens**:
  - Sincronismo da comunicação!
    - Obriga a interrupções no cliente demoradas quando comparadas com os procedimentos locais
    - Diminui drasticamente a fiabilidade do cliente que passa a depender da rede e do servidor
    - Indefinição do resultado quando ocorrem falhas
    - Consumo de mais recursos (banda, CPU, etc.)
    - Aumento dos custos
  - Simplicidade!
    - A “falsa semelhança” pode ser muito perigosa - tecnicamente uma chamada a remota é diferente de uma chamada local

      É necessário prever as falhas e lidar com a lentidão da chamada

Desvantagens (cont.):

- *Timing* do processamento decidido pelo cliente
  - Interrupções constantes para executar pequenas transacções
  - Não existe possibilidade de optimizar o processamento
  - O servidor transforma-se num potencial *bottleneck*
  - Aumento dos recursos – e portanto dos custos
- Cliente e servidor na mesma linguagem de programação
  - Teoricamente não tem de ser assim
  - Mas, na prática, linguagens diferentes limitam os tipos de dados que podem ser trocados
    - Mais uma vez, o XML ajuda a resolver este problema
- Gestão da mudança
  - Alterações nas interfaces e nos próprios procedimentos

Informação Base

Exemplo:

- Nos anos 80 foi criada a OSF (actual *The Open Group*) para normalizar a tecnologia dos procedimentos remotos
- Objectivos
  - Identificar e normalizar as funções básicas
  - Construir uma "implementação de referência"
- Começou por definir uma **arquitectura** para desenvolver aplicações distribuídas
  - Um objectivo bastante mais ambicioso
  - Tornou a norma demasiado complicada
    - Como voltou a acontecer no CORBA
    - Um erro evitado nos Web Services
- O maior impacto da norma foi popularizar a tecnologia

Informação Base

Motivação:

- No anos 90 os objectos estiveram na moda

  - A moda deveu-se principalmente ao C++

    C à C++ à Java/C# à ???

    De 8 em 8 anos cria-se uma nova linguagem

    Portanto a próxima deve estar a aparecer...

  - Também existiram BDOO e SOOO

- Os objectos têm algumas vantagens interessantes

  - Desenho da solução técnica, fluxo de execução dentro do programa, reutilizar e alterar código facilmente, etc.

- Nada mais natural que estender o conceito de “procedimento remoto” para “método remoto”

  - Que abusivamente foi chamado “objecto distribuído” visto que os objectos podiam ser acedidos remotamente

Informação Base

- **Implementação**:
  - Tecnicamente semelhante aos procedimentos remotos embora conceptualmente seja bastante diferente
  - Um "objecto remoto" pode ser implementado por uma **referência** composta por:

    pelo **identificador do objecto** no programa remoto mais um **identificador do programa** remoto
  - Essa referência é depois passada como parâmetro
  - A única dificuldade reside em obter essa referência antes de chamar o método no objecto remoto
  - No final dos anos 90 os objectos começaram a ser substituídos pelos componentes (vai tudo dar ao mesmo)

    COM, EJB, .NET, etc.

Informação Base

- **Vantagens**:
  - Estendem um conceito já conhecido e muito popular
    - Utilizam as suas vantagens e popularidade
    - São mais fáceis de entender e utilizar
    - Permitem reutilizar objectos já existentes
  - Oferecem um nível de granularidade médio
    - Entre os procedimentos e os programas
  - Criou-se logo uma norma (CORBA)
  - Foram adoptados pela Microsoft (COM)
  - Conceito parecido com componentes
  - Independentes da linguagem de programação
    - Desde sempre e muito mais que os procedimentos

- **Desvantagens**:
  - Nunca funcionaram muito bem entre computadores
    - O COM não suportava essa funcionalidade
    - O CORBA era apenas uma “arquitectura”
  - COM e CORBA sempre foram incompatíveis
    - COM (quase) só funciona em Windows
    - Os produtos CORBA eram incompatíveis entre si
  - Herdaram as desvantagens dos procedimentos remotos
    - Sincronismo, falsa simplicidade, *timing* das chamadas, etc.

Conclusão:

Desadequados para integração entre computadores

Totalmente inapropriados para integração entre empresas

- **Exemplo: CORBA**
  - Proposto em 1990 pela OMG
    - Uma organização para criar normas “anti-Microsoft”
  - Apenas uma “arquitectura”
    - A tecnologia não foi normalizada criando muito produtos compatíveis CORBA mas incompatíveis entre si
  - ORB – *Object Request Broker*
    - Um intermediário entre o cliente e o servidor
      - Teoricamente é um conceito muito interessante
      - Mas na prática funciona como um “servidor de nomes”
  - IDL – *Interface Definition Language*
    - Tal como nos RPC (nada de novo, portanto)
      - Difícil mapear para várias linguagens (grande aposta do CORBA)

- Exemplo: CORBA (Cont.):
  - CORBA 2.0
    - Proposto em 1994
    - Objectivo: normalizar a troca de dados entre aplicações
      - Tanto o protocolo de comunicação como o formato dos dados
    - Uma tentativa para compatibilizar os vários produtos CORBA
      - Tal como o DCE tinha feito
      - Veio um pouco tarde demais
  - CORBA 3.0
    - Proposto em 1998
      - Em plena euforia da Internet e do Java
      - Já com o CORBA em declínio
    - Novidade: comunicação assíncrona (tipo mensagens)

• Exemplo: RMI

- Proposto em 1995
- Diferenças para o CORBA
  - Específico para Java
    - Não pode ser utilizado com outras linguagens
    - Mas utiliza muitas funcionalidades do Java como a serialização automática dos objectos
  - Extremamente simples de utilizar
  - Pouco mais que um RPC

    Por exemplo, não suporta transacções como o CORBA
  - Não tem ORB, mas tem servidor de nomes
- Compatível com CORBA
- RMI suporta a integração entre componentes EJB

Comparação entre: RPC vs JAVA RMI:

- RPC não é orientado a objectos
  - Não existe contexto de objecto – podem ser evocadas funções individuais mas o estado não pode ser mantido entre evocações.
  - Não se pode transferir comportamento com dados

- RPC tem uma arquitectura diferente à do RMI
  - Devolver uma chamada no servidor requer um mecanismo RPC a funcionar em ambos os lados.
  - Cliente e Servidor podem ser implementados em linguagens diferentes e residir em SO diferentes

Comparação entre: RPC vs Outros Protocolos

•RPC vs DCOM

- DCOM opera em cima de RPC
- DCOM tem um robusto modelo de segurança
- DCOM requer servers Microsoft em ambos os lados

•RPC vs CORBA

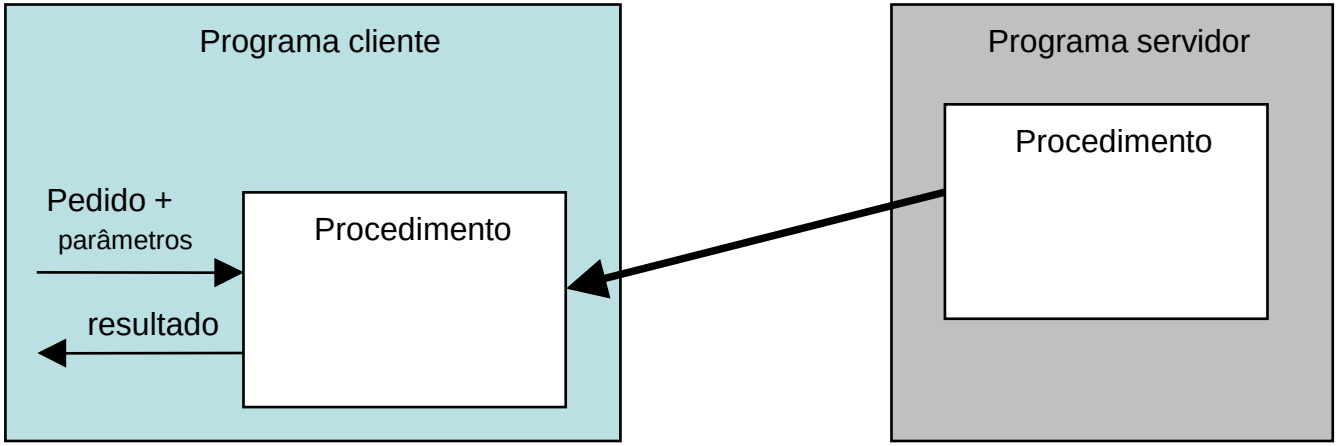
- CORBA não lida muito bem com firewalls
- CORBA tem um robusto e dinâmico mcanismo de nomes
- CORBA requer ORB compatíveis em ambos os lados

Conceito

"RPC funciona ao contrário do código móvel"

Em vez de transferir os dados para junto do código...

... o código é transferido para junto dos dados



Informação Base

- **Vantagens**:
  - Evita as desvantagens dos procedimentos remotos e objectos distribuídos
  - Um programa normalmente é mais fácil de codificar e descodificar que uma estrutura de dados complexa
  - Útil quando existem muitos dados e/ou pretende-se uma interface rica com o utilizador

    Por exemplo, *plug-ins* do tipo Shockwave Flash
- **Problemas**:
  - Todos os programas interessantes são demasiado grandes
  - Levanta enormes questões de segurança
  - Existem poucas necessidades práticas

    à Discussão no próximo slide

Informação Base

- Existem poucas aplicações práticas de código móvel:
  - Se o código for estático (ou raramente alterado) então faz mais sentido substituir toda a aplicação
    - Exemplo: Internet Explorer
  - Se o código for dinâmico (alterado frequentemente) então torna-se complicado estar a substituir partes da aplicação
    - Qualquer programa minimamente interessante é grande
- Java
  - Desenhado de raiz para permitir actualizações de software
    - Applets

Informação Base

Exemplo: Applets

- Programa escrito em Java que pode ser executado no contexto de um *browser*
- Por razões de segurança os applets estão muito limitados
- Os applets interessantes são demasiado grandes para serem transferidos pela Internet
- Na prática existem poucas aplicações para esta tecnologia que já foi considerada como sucessora do Windows

Exemplo: ActiveX

- Tecnologia Microsoft equivalente aos applets
- Os programas ActiveX não estão limitados
- Podem ser escritos em qualquer linguagem
- Apenas funcionam sobre Internet Explorer

- **Conceito**:
  - Tecnologia anterior à Internet
  - Evolução das mensagens assíncronas

    - O canal de comunicação (*bus*) oferece agora uma série de serviços de valor acrescentado
  - Baseados em intermediários

    - O *bus* é substituído pelo *broker*

      -Embora na realidade a diferença seja conceptual, porque as duas implementações podem ser muito semelhantes
  - O nível de abstracção é muito mais alto

    - O principal objectivo é integrar sem alterar as aplicações

    -"Linguagens de programação" de alto nível
  - Ponto de gestão/controlo único

- **Motivação**:
  - O número e complexidade das aplicações nas médias e grandes empresas é cada vez maior
    - A necessidade de integrar essas aplicações tem aumentado exponencialmente (mas o custo também)
    - As tecnologias síncronas (RPC, objectos distribuídos, componentes e Web Services) não resolvem o problema
  - As mensagens têm grandes vantagens

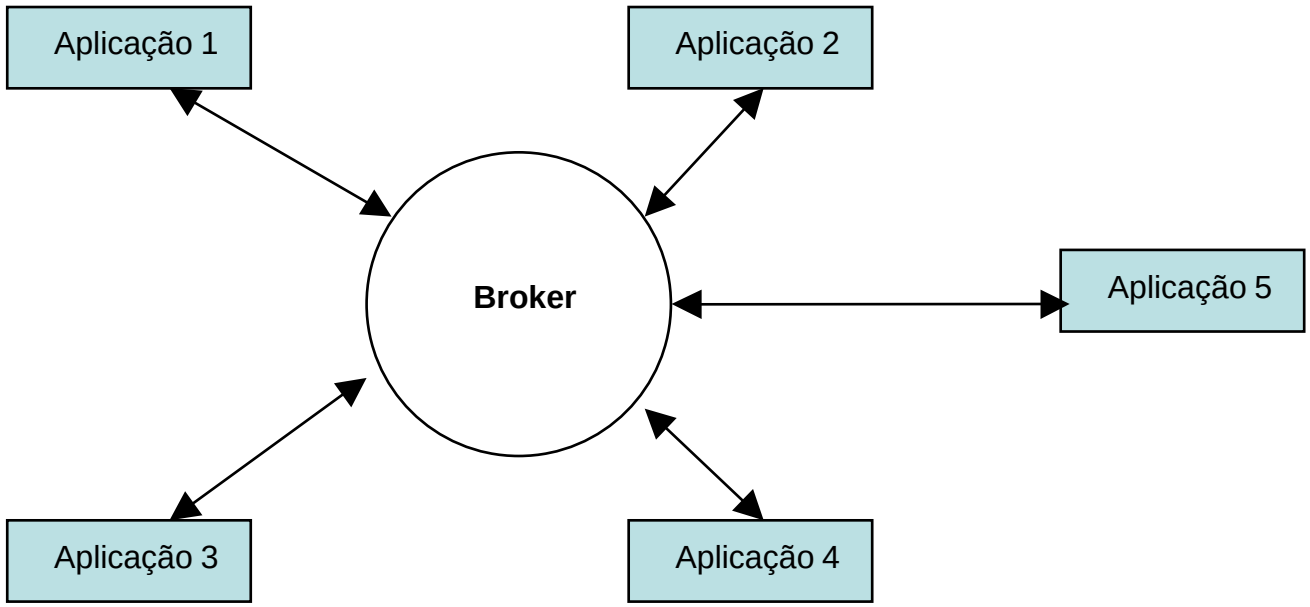
    As aplicações precisam ser integradas apenas uma única vez (com o *bus*, agora chamado *broker*)

    O assincronismo permite oferecer performance, escalabilidade e comunicação transaccional
  - As mensagens "antigas" precisavam de nova designação

- **Motivação**:
  - O conceito de *broker* é muito atractivo
  - Permite obter grandes economias de escala
    - Uma aplicação integrada com o broker fica disponível para todas as outras aplicações
      - O investimento é logarítmico com o numero de aplicações
  - Reduz (ou elimina) as desvantagens das mensagens
    - Normalização dos formatos das mensagens
    - Integração sem programação (adaptadores)
    - Assincronismo “em tempo real”
  - Oferece um ponto único de controlo e gestão

Informação Base

Implementação:



- **Implementação**:
  - Todas as mensagens são enviadas/recebidas para/do broker, sem excepção
    - Mas podem existir vários brokers interligados entre si
  - A troca de mensagens com o broker pode ser:
    - Sincrona – utilizando uma API Java ou Web Services
    - Assíncrona – utilizando uma API Java
  - Existem várias maneiras de trocar mensagens
    - Sincrona ou Assincrona
    - Ponto a ponto ou Multiponto
    - Publish & subscribe
    - Sem ou Com confirmação
    - Sem ou Com resposta

Informação Base

- **Funcionalidades**:
  - Transformação das Mensagens
    - As mensagens podem ser transformadas no broker
      - Utilizando uma interface gráfica – que pode ser estendida com programação quando necessário
    - Aconselha-se utilizar um "formato neutro" para cada tipo de mensagem
      - Exemplos: encomenda, cliente, etc.
      - Todas as mensagens vindas das aplicações são convertidas para este formato neutro, e vice-versa
    - Existem dois tipos principais de transformação
      - Esquemas – estrutura da mensagem e formato dos campos
      - Dados – alteração dos dados propriamente ditos

        - Texto, números, datas, formatos legados e proprietários, estruturas de dados complexas (e.g. XML)

■ Introdução

■ As tecnologias necessárias para integrar aplicações e SI entre empresas ou dentro das empresas é basicamente a mesma.

■ Excepto na Segurança que entre empresas terá de ser garantida.

■“Dentro” <> “Entre”.

As empresas estão muitas vezes divididas em “centros de negócio”, que são na práticas “empresas dentro de empresas”, com departamentos funcionais próprios.

Introdução

- Aprofundamento das relações entre organizações.

  Ex. Consórcios.

- Entre empresas significa que existe uma separação bastante forte – em termos tecnológicos e organizacionais – entre as "empresas" responsáveis pelos SI que precisam de ser integrados.

  - Uma integração entre SI de dois departamentos funcionais de uma empresa pode ser considerada integração entre empresas.

Informação Base

- **Definição**
  - EDI - Electronic Data Interchange
    - Transferência de mensagens normalizadas entre sistemas informáticos de parceiros comerciais utilizando telecomunicações com a menor intervenção humana
  - EDI Tradicional
    - Quando esta transmissão é realizada sem tecnologias, formatos ou normas baseadas na Internet
  - EDI "normalizado"
    - Teve início nos anos 60

      Embora já fosse praticado com formatos proprietários

- **Vantagens**
  - Eliminação do trabalho manual
    - Redução de custos
  - Redução do tempo de transmissão
    - Redução de inventário (logo custos)
  - Aumento da quantidade e qualidade das mensagens
    - Informação mais actualizada
    - Necessário para certos processos de negócio

      Just in time, Vendor managed inventory, etc.
  - Eliminação dos erros manuais
    - Redução de custos
  - Pressão do mercado
    - Os grandes clientes preferem mandar as encomendas por EDI

- **Dificuldades**
  - Investimento:
    - Entre 5,000 e 200,000 euros
    - Proporcional aos tipos de mensagens
    - Anda maior se houver integração com ERP

      Condição necessária para tirar partido do EDI
  - As PME não conseguem tirar partido do EDI:
    - Não estão suficientemente informatizadas
    - Não percebem para que serve o EDI
    - O numero de documentos trocados não compensa
  - Tecnologias desactualizadas:
    - Difíceis de instalar, manter e utilizar

- Dificuldades (cont.)
  - Nicho de mercado
    - Poucos fornecedores
      - Pouca experiência, pouca concorrência
    - Há poucas economias de escala
    - Custos elevados
  - Formatos tradicionais demasiado rígidos
    - Não se adaptam a situações particulares
    - Não permitem criar vantagens comparativas
  - Demasiado complexo
    - Muitas mensagens, muitos campos, muitas opções
    - Cada projecto é quase “à medida”

Informação Base

- **Tipos de Mensagens**
  - Retalho

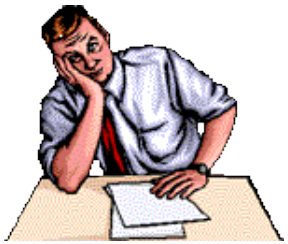
    Catálogos, encomendas e facturas
  - Logística

    Avisos de expedição e recepção
  - Alfândegas
  - Impostos
  - Saúde

    nos Estados Unidos, onde a saúde é privada

  - Cada tipo de mensagem tem a sua particularidade!

Informação Base



Fornecedor

Informação Parceiro Comercial

Catálogo de Produtos (c/ preços)

Nota de Encomenda

Aviso de Expedição

Aviso de Recepção

Factura

Relatório de Vendas

Cliente

Informação Base

Cadeia de Valor

- Os processos de negócio entre organizações são extremamente complexos
  - Não se limitam a encomendas e facturas entre duas empresas

    Ver figura no slide seguinte

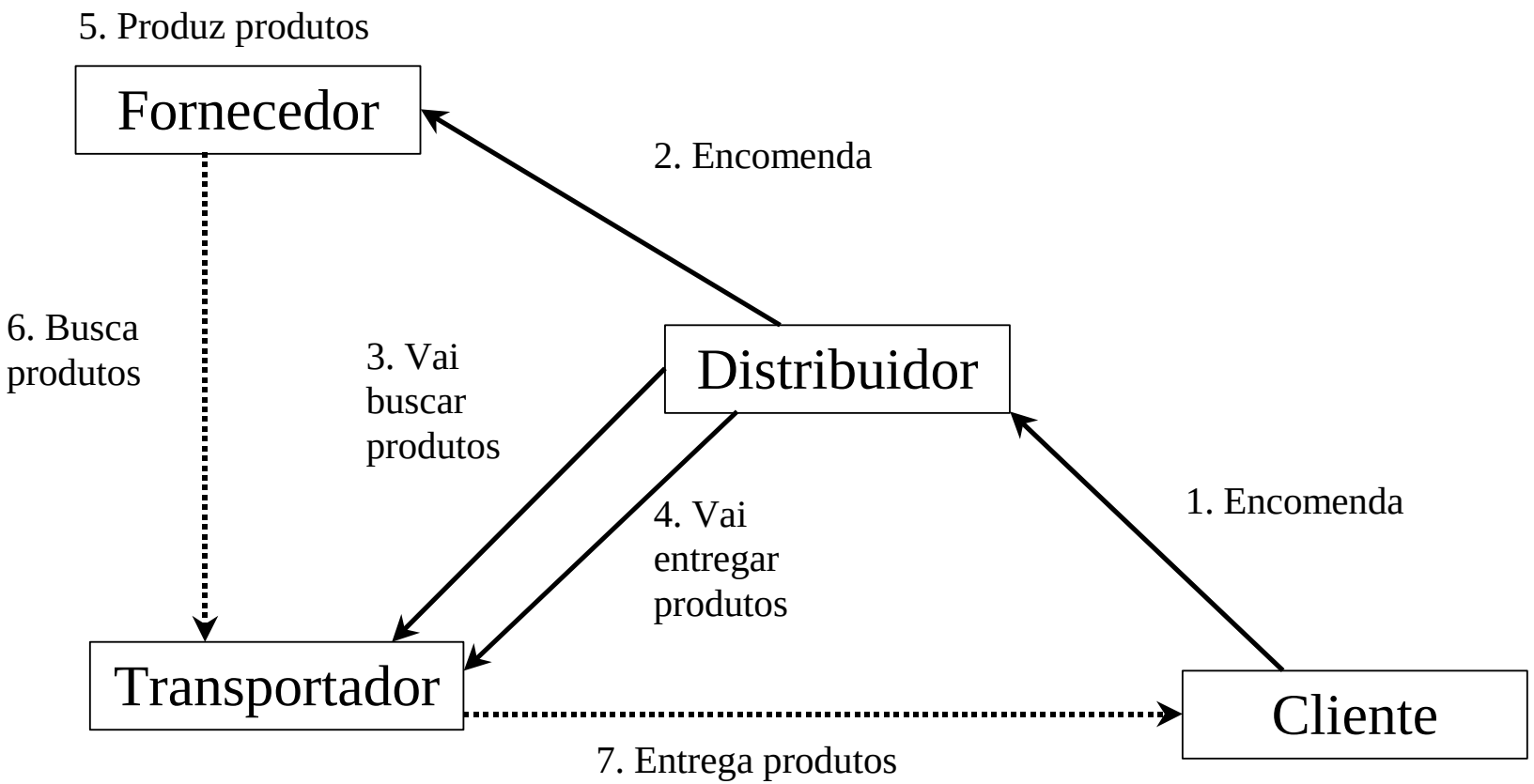
- A maioria do EDI limita-se à troca de documentos apenas entre duas empresas
  - O que não permite explorar a tecnologia ao máximo

    É como ter um Ferrari mas não ter lugar para estacionar o carro!

- A competição hoje em dia é feita entre cadeias de valor!
  - Por isso o ideal seria suportar os principais processos de negócio das cadeias de valor com EDI

- Os custos, tal como os benefícios, devem ser repartidos por todas as empresas da cadeia de valor!

Informação Base



Informação Base

- EDIFACT
  - Motivação
    - Trocar dados em formato electrónico é relativamente fácil
      - Existem inúmeros protocolos normalizados - X.400, Internet, etc.
    - Formatar dados é mais complicado
      - E não existem muitas normas

  - EDIFACT é uma norma para formatar documentos de negócio promovida pelas Nações Unidas
    - X12 é uma norma equivalente, mas americana

  - As mensagens EDIFACT foram desenhadas para ser escritas e lidas por computadores
    - Não são nada agradáveis de visualizar
      - Ver exemplo no slide seguinte

Informação Base

```
UNA:+.?
UNB+UNOA:2+5600000000427:14+5600000002179:14+000905:1309+06021E00041174++ORD
UNH+1+ORDERS:D:96A:UN:EAN008
BGM+220+010400767331+9
DTM+2:200009061000:203
DTM+137:200009050000:203
NAD+SU+5600000002179::9
NAD+BY+5600000000137::9
CUX+2:PTE:9
LIN+1++5603089000029:EN
PIA+1+2211012:BP
IMD+F+DSC+:::Produto Teste1
QTY+21:3600
PRI+AAA:124.8
PAC+5++BX
TAX+7+VAT+++:::5
LIN+2++5603089000036:EN
PIA+1+2211013:BP
IMD+F+DSC+:::Produto Teste2
QTY+21:1440
PRI+AAA:122.2
PAC+2++BX
TAX+7+VAT+++:::5
LIN+3++5603089010202:EN
PIA+1+2419243:BP
```

Papel da CODIPOR

- Nas mensagens electrónicas toda a informação deve estar codificada em números

  Produtos, empresas, moradas, unidades de medida, etc.

- No comércio electrónico entre empresas a codificação deve ser gerida por uma entidade terceira CODIPOR
  - Emite códigos de barras para identificar empresas e produtos

    Que também são utilizados no EDI
  - Emite códigos de "localização"

    Basicamente moradas
  - Aprova e promove o EDI

    Por exemplo, tem cursos sobre EDI

Informação Base

- **Exemplo: SONAE**
  - Teve início nos anos 90
  - Maior projecto de EDI em Portugal
  - Em 1998 30% das encomendas foram enviadas por EDI
    Mas apenas quatro fornecedores receberam relatórios de inventário por EDI
  - Em 2000 50% das encomendas foram enviadas por EDI
    Apenas 500 fornecedores - para um total de 5000
    Apenas 2/3 tinham o EDI integrado com o ERP
  - Algumas empresas fazem EDI
    EDP, SIBS, DGITA, etc.
  - Muitas fazem "EDI proprietário"
    Por exemplo, Auto-europa (ODETTE)

- Introdução
  - Nos últimos anos vários foram iniciados vários projectos para trocar documentos electrónicos entre empresas utilizando a Internet
    - Principalmente desde 1998 com o aparecimento do XML
  - Os Web Services não podem ser utilizados entre empresas
    - Sincronismo não funciona

      Redes estreitas e pouco fiáveis
    - Não oferecem segurança
    - Problemas de interoperabilidade
    - Tecnologia ainda recente
  - Solução
    - Formatos e protocolos proprietários

- Protocolos de Comunicações
  - TCP/IP
    - Universal
    - Síncrono, baixo nível, pouco robusto, não tem segurança
  - FTP
    - Robusto
    - Síncrono, pesado, lento, pouco seguro
  - HTTP
    - Universal, flexível (com POST), seguro (com SSL), atravessa firewalls
    - Síncrono

- Protocolos de Comunicações
  - SMTP
    - Assíncrono, atravessa firewalls, seguro (confidencialidade, integridade, autenticação do emissor e do receptor)
    - Pouco fiável
  - Protocolos proprietários
    - Suportam funcionalidades adicionais

      Assincronismo, fiabilidade, etc.
    - Utilizando um (ou vários) dos anteriores

      Tipicamente HTTP e SMTP

- Formato das Mensagens
  - Actualmente é (quase) sempre XML
    - Fácil de aprender e usar em qualquer linguagem
    - Existem inúmeras ferramentas para XML
    - Permite criar facilmente novos tipos de mensagens
    - Fácil de converter para outros formatos (com XSL)
  - Dificuldades
    - XML é apenas uma linguagem
      - Continua a ser difícil integrar com ERP
    - Não existem normas de tipos de mensagens
      - Apesar de várias tentativas

        Commerce One, Ariba, etc.

        Mais recente → UBL

- Segurança
  - Dois tipos básicos
    - Baseada no protocolo de comunicações
    - Implementada em cima do protocolo
      - Na própria mensagem XML
      - No envelope da mensagem
  - HTTP
    - SSL

      Garante confidencialidade, integridade e autenticação do receptor

      Também é possível garantir autenticação do emissor
    - Não suporta assinaturas digitais

      Impossivel garantir não repúdio

      Necessárias para facturas digitais
    - Existe uma norma (XML-Signature) para incluir assinaturas digitais nos documentos XML

- Segurança
  - SMTP
    - S/MIME
      - Confidencialidade
      - Integridade
      - Autenticação do emissor
      - Autenticação do receptor
      - Não repúdio
    - Só falta mesmo “prova de entrega”
      - O equivalente à carta regista com aviso de recepção

- Exemplo: Business Integrator
  - Produto da THINK criado originalmente para o PAPINET
  - Em 1999 já suportava troca de encomendas reais entre empresas pela Internet
  - Em 2003 existem dezenas de utilizadores
  - Algumas funcionalidades
    - Suporte para HTTP, SMTP e outros protocolos
    - Integrado com SAP, BAAN, Primavera, PHC e Sage/Infologia
    - Implementa a legislação das facturas electrónicas
    - Utilizado nos maiores projectos de EDI em Portugal
    - Implementado em Java (corre em qualquer sistema operativo)

Informação Base

• Descrição

ebXML - Electronic Business using eXtensible Markup Language ou e-business XML

- "Iniciativa" para desenvolver normas que facilitem os negócios entre empresas baseados na Internet

  Norma para EDI na Internet – muito completa

- Patrocinado por
  - OASIS – organização muito respeitada na área do XML
  - UN/CEFACT – departamento das Nações Unidas responsável pelas mensagens do EDI tradicional (EDIFACT)
- Vários tipos de documentos
  - Especificações técnicas
  - Relatórios técnicos
  - Documentos de referência
  - White papers

- Grupos de trabalho
  - Registry – produz normas para registos e repositórios
  - Messaging – produz normas para troca de mensagens assíncronas pela Internet
  - Collaborative Partner – produz normas para definir contratos de colaboração
  - Implementation – ajuda a implementar a norma e garantir que essas implementações são interoperáveis
  - Business Process – produziu uma norma para especificar processos de negócio em XML (BPSS)
  - Core Component – produziu uma norma que permite trocar metadata entre empresas

- Comparação com Web Services
  - Objectivos completamente diferentes
    - Web Services – troca síncrona de dados em formato XML entre aplicações dentro da empresa

      Não está vocacionado para processos de negócio entre empresas
    - ebXML – troca assíncrona de mensagens de negócio entre empresas no contexto de processos de negócio

      Não está vocacionado para integração de aplicações na empresa
  - Ponto em comum
    - ebXML utiliza SOAP como protocolo de comunicações

      Facilita a implementação de ebXML

- Implementações
  - HERMES
    - Implementação open source
    - Suporta transporte, segurança, entrega garantida, tratamento de erros, resposta síncrona, armazenamento de mensagens e qualidade de serviços
    - Utilizado num piloto com o metro de Hong Kong
    - Existem mais seis implementações na Ásia

      Todas interoperáveis
  - JAXM
    - Interface Java para trocar mensagens XML assincronamente
    - Está preparada para ebXML

- Vendor Managed Inventory (VMI)
  - O inventário do cliente passa a ser gerido pelo fornecedor
    - Permite reduzir drasticamente os níveis de inventário
    - Mas necessita trocar grandes quantidades de informação

      Nível de inventário de todos os produtos todos os dias
  - No cliente pouco se altera
    - A encomenda continua a ser calculada com base no inventário
  - No fornecedor tudo se altera
    - No limite o fornecedor pode eliminar todo o inventário

      Pois sabe com grande antecedência quando será necessário
  - Para que o inventário do cliente se mantenha é preciso alterar o processo de negócio
    - O cliente só paga o produto depois de vender ao cliente final

- Facturas Electrónicas
  - A factura é um tipo de documento comercial muito especial
    - A factura electrónica tem legislação específica se quisermos acabar com a factura em papel
  - Legislação
    - Decreto-lei 375/99 e decreto regulamentar 16/2000
    - Sem consequências práticas porque obrigam a utilizar certificados emitidos por entidades credenciadas

      Que não existem!
  - Vantagens
    - Fundamental para “digitalizar” o processo de negócio mais comum entre empresas

      Encomenda seguida de factura

- Facturas Electrónicas (cont.)
  - Custos da factura em papel
    - 1 empregado-ano para processar 10,000 facturas

      1 empregado-ano custa 50,000 euros

      Processar uma factura custa 5 euros
    - Na Europa são trocadas 12 mil milhões de facturas por ano

      O custo de processar facturas em pape é 60 mil milhões de euros!
    - Portugal é 3% da Europa

      O custo em Portugal é 1800 milhões de euros por ano
  - Dificuldades
    - Falta corrigir a legislação de acordo com a Directiva 2001/115
    - Falta credenciar as autoridades que emitem certificados
    - Falta a DGCI responder aos pedidos – obrigatórios por lei

- Pagamentos Electrónicos
  - Pagamentos na Internet são baseados em cartão de crédito
    - No entanto, as empresas não pagam com cartão de crédito pagam com cheques e transferências bancárias
  - Tanto os cheques como as transferências podem ser facilmente digitalizados
    - Não passam de documentos assinados que podem ser assinados digitalmente
    - As propostas de “cheques digitais” nunca foram populares
  - As transferências podem ser implementadas com EDI entre empresas e bancos
    - SIBS oferece “EDI financeiro” mas é pouco utilizado

- Documentos Electrónicos
  - Qualquer tipo de documento pode ser digitalizado e assinado digitalmente
  - Já existe legislação – Decreto-lei 290-D/99
  - Existem documentos especiais
    - Podem precisar de ser validados – e.g. notários
      - Facturas, cheques, contratos, etc.
  - Contratos electrónicos
    - Um exemplo de documento especial muito útil
    - Já existem ferramentas tecnológicas

      COSMOS, OCTANE, etc.
    - Já existe legislação – directiva europeia 2000/31 que não foi implementada em Portugal

Informação Base

- XML
  - http://www.w3.org/XML/
  - http://www.xml.com/
  - http://www.w3schools.com/xml/

- Web Services

Informação Base

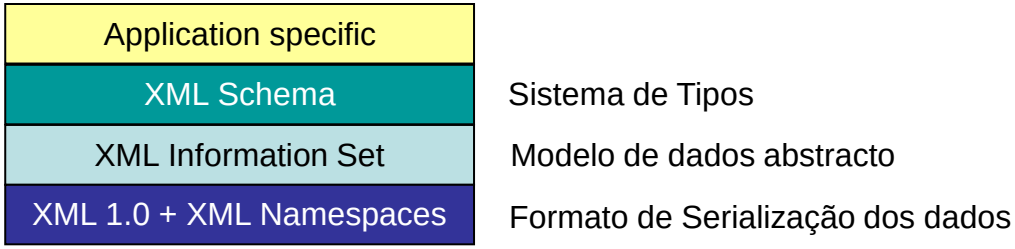
• Extensible Markup Language (XML) é linguagem de marcação de dados (meta-markup language) que fornece um formato para descrever dados estruturados. Isso facilita declarações mais precisas do conteúdo e resultados mais significativos de busca através de múltiplas plataformas. O XML também vai permitir o surgimento de uma nova geração de aplicações de manipulação e visualização de dados via internet.

• O XML permite a definição de um número infinito de tags. Enquanto no HTML, as tags podem ser usadas para definir a formatação de caracteres e parágrafos, o XML fornece um sistema para criar tags para dados estruturados.

• Um elemento XML pode ter dados declarados como sendo preços de venda, taxas de preço, um título de livro, a quantidade de chuva, ou qualquer outro tipo de elemento de dado. Como as tags XML são adoptadas por intranets de organizações, e também via Internet, haverá uma correspondente facilidade em manipular e procurar por dados independentemente das aplicações onde os quais são encontrados. Uma vez que o dado foi encontrado, ele pode ser distribuído pela rede e apresentado num browser de várias formas possíveis, ou então esse dado pode ser transferido para outras aplicações para processamento futuro e visualização.

Informação Base

- Conjunto de especificações mantidas e publicadas pelo *World Wide Web Consortium* (W3C) – http://www.w3.org/;
- Tecnologia que associa estrutura e tipo a dados;
- Descreve um conjunto de objectos de dados denominados por documentos XML;
- Constituído por camadas.



Informação Base

• Exemplo:

```
<?xml version="1.0" standalone="yes"?>
<RECEITAS>
  <TITULO>Livro de receitas de F. Lobo</TITULO>
  <RECEITA>
     <NOME>Bolo de chocolate</NOME>
     <INGREDIENTE>500g de farinha</INGREDIENTE>
     <INGREDIENTE>200g de açucar</INGREDIENTE>
     <INGREDIENTE>300g de manteiga</INGREDIENTE>
     <INGREDIENTE>1 tablete de chocolate</INGREDIENTE>
  </RECEITA>
</RECEITAS>
```

Informação Base

• Objectivos:

- Facilmente utilizável na Internet;
  - Simplicidade:
    - Na criação de documentos XML;
    - No desenvolvimento de aplicações que os manipulam;
  - Compatível com SGML (*Standard Generalized Markup Language*);
  - Utilizável numa grande variedade de aplicações.

Informação Base

- Documento de texto (*Unicode character set*);

- Estrutura de um documento XML:

  document ::= prolog ? Element

Ex:

```
<?xml version="1.0"?>
<elemento_raiz>
        ...
</elemento_raiz>
```

- Implicações desta estrutura:
  - Um documento XML contém um ou mais elementos;
  - Existe exactamente um elemento que não está contido por nenhum outro:
    - Este elemento é denominado por ***root*** ou ***document element***

Informação Base

- O prólogo (*prolog*) é opcional e pode conter:

  - *XML declaration*

    Contém informação, relativa ao documento, a ser utilizada pelo *parser.*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

    - *version* - versão da especificação que o documento cumpre;

    - *encoding* - tipo de codificação utilizado (opcional);

    - *standalone* - indica se a informação que resulta do processamento do documento depende ou não de declarações externas ao mesmo (opcional).

  - *Document type declaration*

    - Indica quais as restrições a aplicar ao documento através de DTD (*Document Type Definition*) interna e/ou externa ao documento.

Informação Base

• Estrutura de um elemento:

element ::= (STag content ETag) | EmptyElementTag

Exs:

<nome> Paulo Pereira </nome> - Elemento com conteúdo

<email /> - Elemento sem conteúdo

• O conteúdo (*content*) de um elemento pode ser qualquer combinação de texto e outros elementos;

• O texto utilizado nas marcas (*STag*, *ETag* ou *EmptyElementTag*) é *case-sensitive*.

Informação Base

- Um elemento pode conter atributos. A sintaxe é a seguinte:

  Attribute ::= attName '=' AttValue

Ex:

```
<contacto tipo = "pessoal" >
    <nome> Paulo Pereira </nome>
    <email endereço = "palbp@isel.pt" />
</contacto>
```

AttValue ::= '"' charSequence '"' | "'" charSequence "'"

- Atributos num elemento:
  - com conteúdo

    STag ::= '<' elementName (S Attribute)* S? '>'
  - sem conteúdo

    EmptyElementTag ::= '<' elementName (S Attribute)* S? '/>'
  - Separadores: S ::= {#x20 | #x9 | #xD | #xA}+

Informação Base

- Nos nomes de elementos e atributos:
  - Name ::= (Letter | '_') (Letter | Digit | '_' | '-' | '.')*
  - Nomes que comecem por uma sequência de caracteres que cumpra a seguinte produção, estão reservados.

    ('x' | 'X') ('m' | 'M') ('l' | 'L')

- No conteúdo de elementos:
  - Todos excepto '<' e '&', que têm um significado específico.

- Nos valores de atributos:
  - Todos excepto:

    '<' e '&', que têm um significado específico;

    Carácter utilizado como delimitador.

Exs:

attr1 = '"I can use quotes here", he said.'	OK
attr2 = "I don't thing this is wrong."	
attr3 = 'I can't do this.'	Erro

Informação Base

- *Character References*:

  - Possibilitam utilizar no documento o código *Unicode* de um carácter em vez do seu literal.

  CharRef ::= (‘&#’ decimalCode ‘;’) | (‘&#x’ hexadecimalCode ‘;’).

- *Entity References*:

  - Mecanismo semelhante às macros da linguagem C.
  - Permitem inserir no documento o conteúdo de entidades pré-definidas ou definidas através de DTD.
  - EntityRef ::= (‘&’ entityName ‘;’).
  - Nomes das 5 entidades pré-definidas:

    apos - ‘’’

    amp - ‘&’

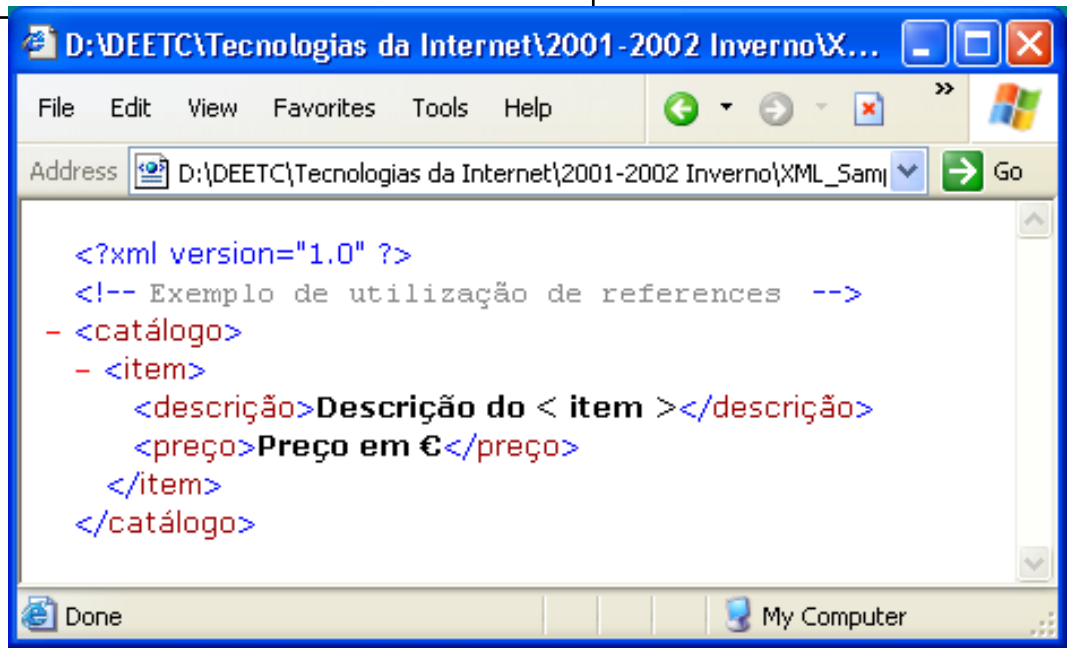
    quot - ‘””

    lt - ‘<‘

    gt - ‘>’

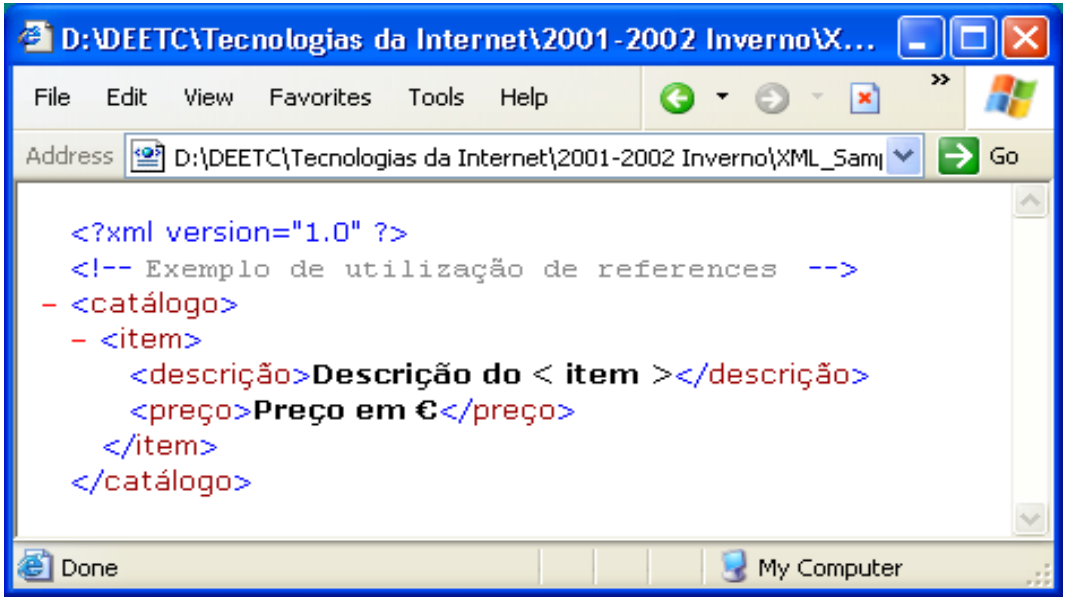
Informação Base


```
<?xml version="1.0"?>
<!-- Exemplo de utilização de references -->
<catálogo>
  <item>
    <descrição> Descrição do &lt; item &gt; </descrição>
    <preço> Preço em &#x20ac; </preço>
  </item>
</catálogo>
```



Informação Base


```
<?xml version="1.0"?>
<!-- Exemplo de utilização de references -->
<catálogo>
  <item>
    <descrição> Descrição do &lt; item &gt; </descrição>
    <preço> Preço em &#x20ac; </preço>
  </item>
</catálogo>
```



• O prólogo (*prolog*) é opcional e pode conter:

- *XML declaration*

- Contém informação, relativa ao documento, a ser utilizada pelo *parser.*

Ex:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

- *version* - versão da especificação que o documento cumpre;
- *encoding* - tipo de codificação utilizado (opcional);
- *standalone* - indica se a informação que resulta do processamento do documento depende ou não de declarações externas ao mesmo (opcional).

- *Document type declaration*

Indica quais as restrições a aplicar ao documento através de DTD (*Document Type Definition*) interna e/ou externa ao documento.

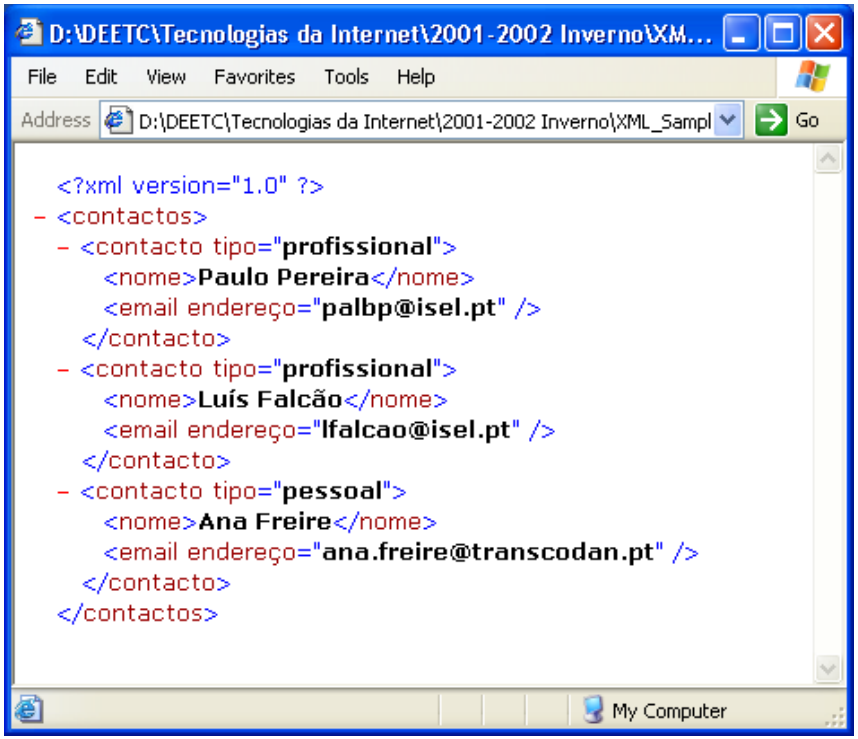
Informação Base

```
<?xml version="1.0"?>
<!-- Exemplo de utilização de references -->
<catálogo>
  <item>
    <descrição> Descrição do &lt; item &gt; </descrição>
    <preço> Preço em &#x20ac; </preço>
  </item>
</catálogo>
```

Resultado no IE6

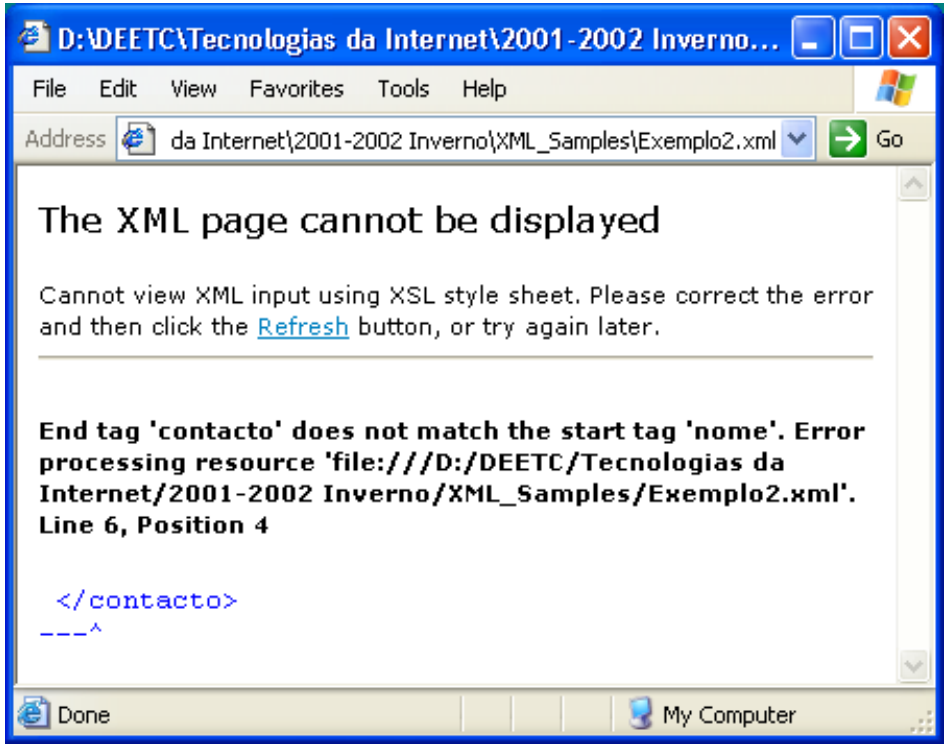
Um documento que cumpra a estrutura e a sintaxe descritas é designado por documento XML bem formado (*well-formed*).


```
<?xml version="1.0"?>
<!-– Documento XML bem formado -->
<contactos>
  <contacto tipo="profissional">
    <nome> Paulo Pereira </nome>
    <email endereço="palbp@isel.pt" />
  </contacto>
  <contacto tipo='profissional'>
    <nome> Luís Falcão </nome>
    <email endereço="lfalcao@isel.pt" />
  </contacto>
  <contacto tipo="pessoal">
    <nome> Ana Freire </nome>
    <email endereço="ana.freire@transcodan.pt" />
  </contacto>
</contactos>
```



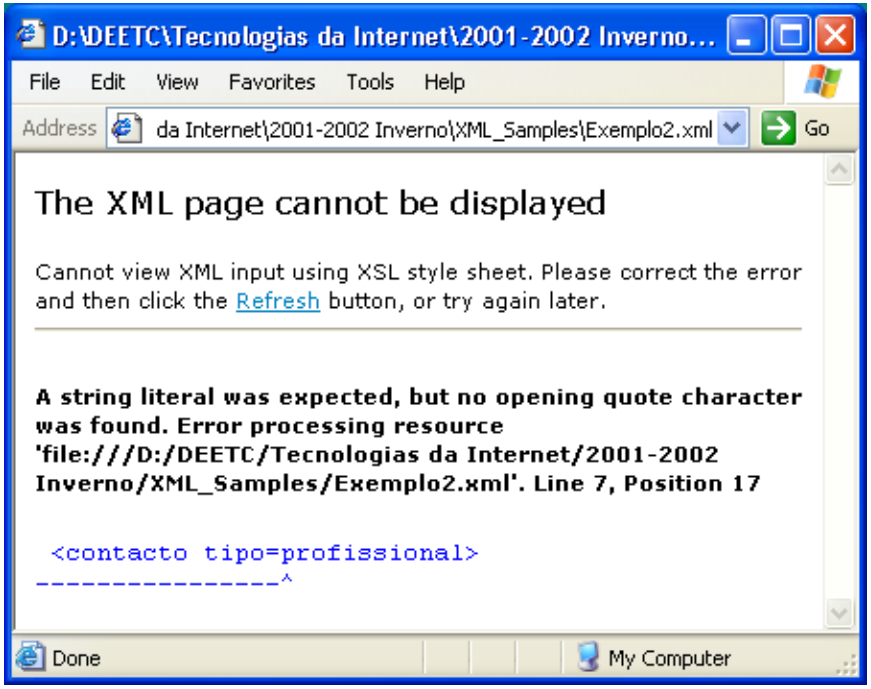
Informação Base


```
<?xml version="1.0"?>
<!-- Documento com erro de estrutura -->
<contactos>
  <contacto tipo="profissional">
    <nome> Paulo Pereira
    <email endereço="palbp@isel.pt" />
  </contacto>
  </nome>
  <contacto tipo='profissional'>
    <nome> Luís Falcão </nome>
    <email endereço="lfalcao@isel.pt" />
  </contacto>
</contactos>
```



Informação Base

```
<?xml version="1.0"?>
<!-– Documento com erro sintáctico -->
<contactos>
  <contacto tipo="profissional">
    <nome> Paulo Pereira </nome>
    <email endereço="palbp@isel.pt" />
</contacto>
  <contacto tipo=profissional>
    <nome> Luís Falcão </nome>
    <email endereço="lfalcao@isel.pt" />
  </contacto>
</contactos>
```



Informação Base

Web Services

- Definição
  - Um conjunto de normas que definem como descrever, publicar e utilizar determinado tipo de componentes
  - Principais normas
    - **SOAP** – um mecanismo (tipo RPC) para invocar métodos sincronamente em componentes remotos
    - **WSDL** – uma linguagem (tipo IDL) para descrever formalmente a interface de um "serviço" (conjunto de métodos oferecidos por um componente)
    - **UDDI** – um serviço (implementado por um componente) que armazena e disponibiliza interfaces de outros serviços
  - SOAP é 99% dos Web Services

    - WSDL é invisível e UDDI não é utilizado

Informação Base

Web Services

- Bibliografia
  - NET XML Web Services step by step, Microsoft
  - Professional Java XML, WROX
  - Professional Servicios Web XML, Wrox

  - http://www.webservices.org
  - http://www.service-architecture.com/

Informação Base

Web Services

• Nível do Negócio:

- Maravilhoso mundo novo que permitirá criar novas aplicações utilizando funcionalidades residentes em aplicações legadas e componentes remotos
- O modelo ASP (*Application Servive Provider*) prometia aplicações remotas que seriam alugadas
  - Mas falhou redondamente
- Perguntas
  - Serão os Web Services uma "reinvenção" dos ASP?
    - Também os ASP eram baseados em software remoto e alugado
  - Qual o "modelo de negócio" para os Web Services ?
    - Também o CORBA prometia um "mercado" de componentes
  - Fará sentido "alugar" componentes remotos ?

Informação Base

Web Services

- Nível Técnico
  - Os Web Services são apenas uma tecnologia de integração "tipo RPC" baseada em XML
    - *Provavelmente* os Web Services são apenas outra tecnologia de componentes remotos como o CORBA, EJB e DCOM
    - Não trazem nada de fundamentalmente novo
      - Bem pelo contrário, até podem ser considerados um retrocesso tecnológico pois recuperam as tecnologias que deram origem ao falhanço do CORBA
  - Grande Mentira
    - Web – não têm nada a ver com HTML
    - Services – são componentes vulgares com métodos que podem ser invocados remotamente
  - Mas afinal o que é um serviço ??

Web Services

- Motivação
  - Lado da Procura
    - Não havia nada de novo ultimamente
      - Depois da euforia da Web, bug do ano 2000 e Euro
    - A maioria das empresas tinham apostado em ERP
      - Que não ficaram integrados com as outras aplicações
    - Preferência por projectos pequenos de retorno rápido
      - Os projectos de integração enquadram-se nesta categoria
  - Lado da Oferta
    - Era preciso inventar uma nova buzzword
    - De preferência baseada na Web - que entretanto tinha substituído o próprio termo Internet
    - A integração está na moda

• Conclusão

- As “novas tecnologias” pouco trazem de novo
- O XML é interessante
  - Mas já existiam muitas linguagens para formatar dados
- Os Web Services prometem um mundo novo
  - Mas são apenas os velhinhos RPC de cara lavada
    - A simplicidade apenas reflecte a falta de funcionalidades
    - À medida que as funcionalidades vão sendo acrescentadas mais se parece com o CORBA
      - Que teve um triste destino...
- A segurança continua a ser uma grande lacuna
- Continua a não existir quase nada para ligar a tecnologia (Web Services) à gestão (processos de negócio)

Informação Base

III –Tecnologias Internet para Integração

Informação Base

- A Engenharia de Software para Aplicações Internet
- XML
  - Introdução
  - Standards fundamentais e específicos
  - XML Schema
  - Desenho de estruturas de dados em XML
- Web Services
  - Ciclo de Vida
  - Standards
    - UDDI
    - SOAP
    - WSDL
    - ebXML

Informação Base

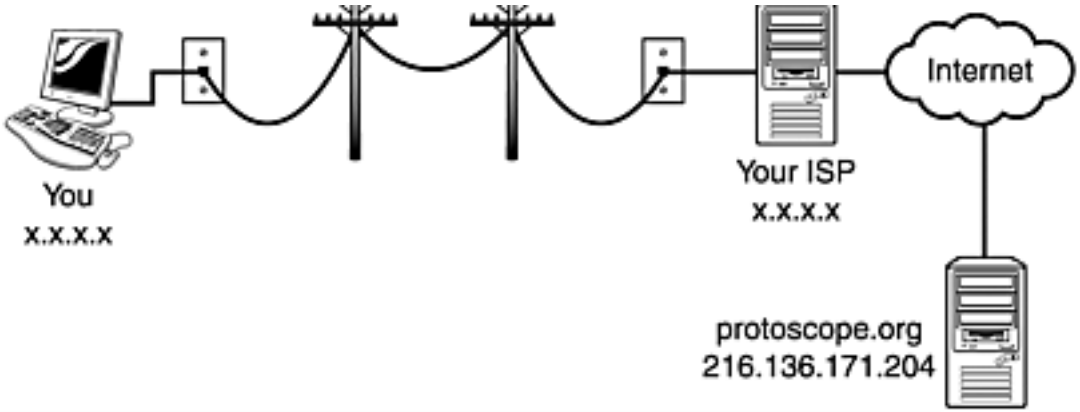
- Lopes, C. e Ramalho, J.C., Web Services, Aplicações Distribuídas sobre Protocolos Internet, FCA

- Pressman, R. S., Software Engineering – A Practitioner's Approach 6th edition, McGraw Hill.

- Short, S., Building XML Web Services for the Microsoft .NET Platform, Microsoft Press. WakeField, C., Sonder H., Lee, W. M., VB.NET Developer's Guide, Syngress.

- Yao, P., Durant D., .NET Compact Framework Programming With Visual Basic.NET, Addison Wesley

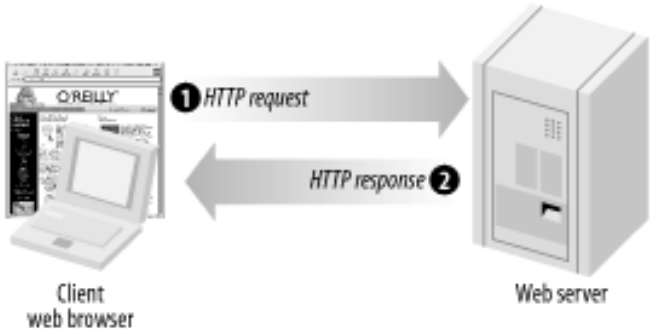
Web

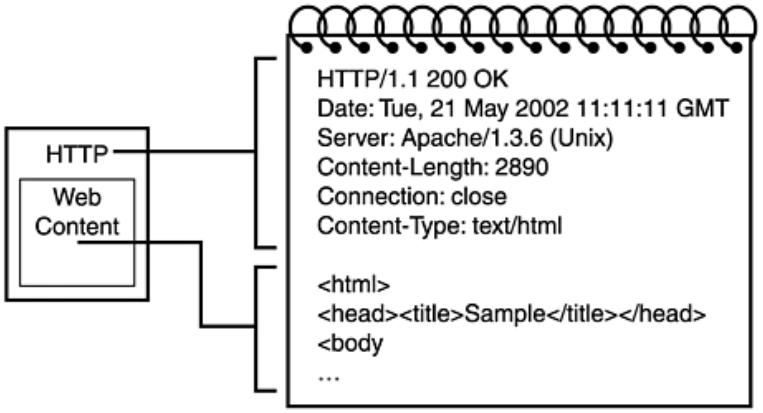

Internet
You
x.x.x.x
Your ISP
x.x.x.x
protoscope.org
216.136.171.204

- Conceitos

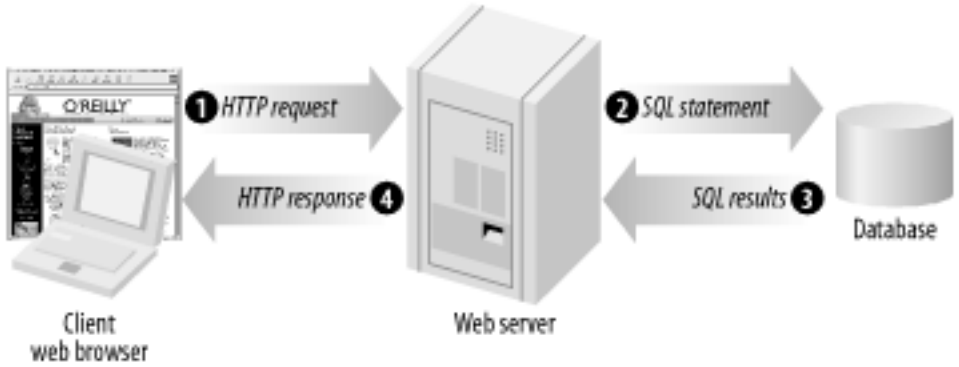
Web

Informação Base



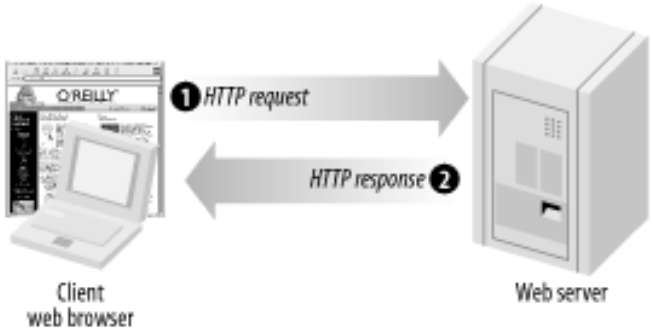


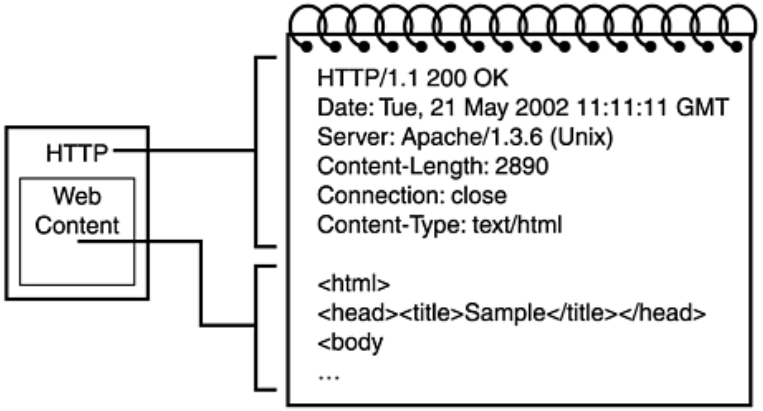
Arquitectura 3 camadas



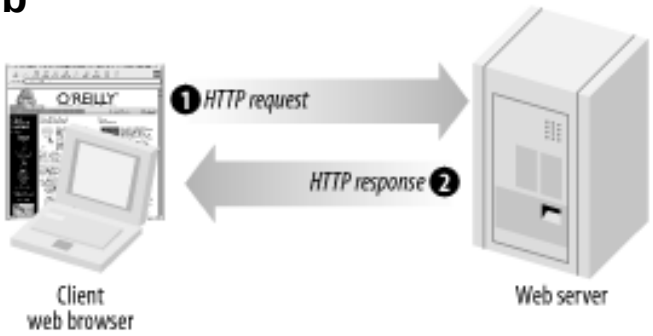
Informação Base

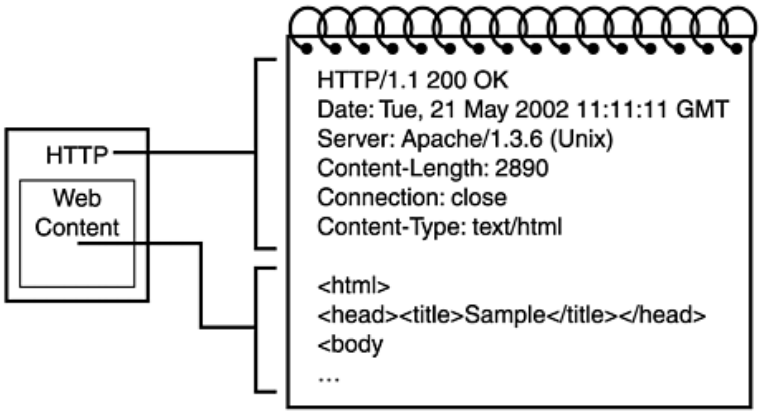
Web





Web





HTTP/1.1 200 OK

Date: Tue, 21 May 2002 12:34:56 GMT

Server: Apache/1.3.22 (Unix) (Red-Hat/Linux) mod_python/2.7.8 Python/1.5.2 mod_ssl/2.8.5 OpenSSL/0.9.6b DAV/1.0.2 PHP/4.0.6 mod_perl/1.26 mod_throttle/3.1.2

Last-Modified: Thu, 01 Nov 2001 20:51:45 GMT

ETag: "df6b0-b4a-3be1b5e1"

Accept-Ranges: bytes

Content-Length: 2890 Connection: close

Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

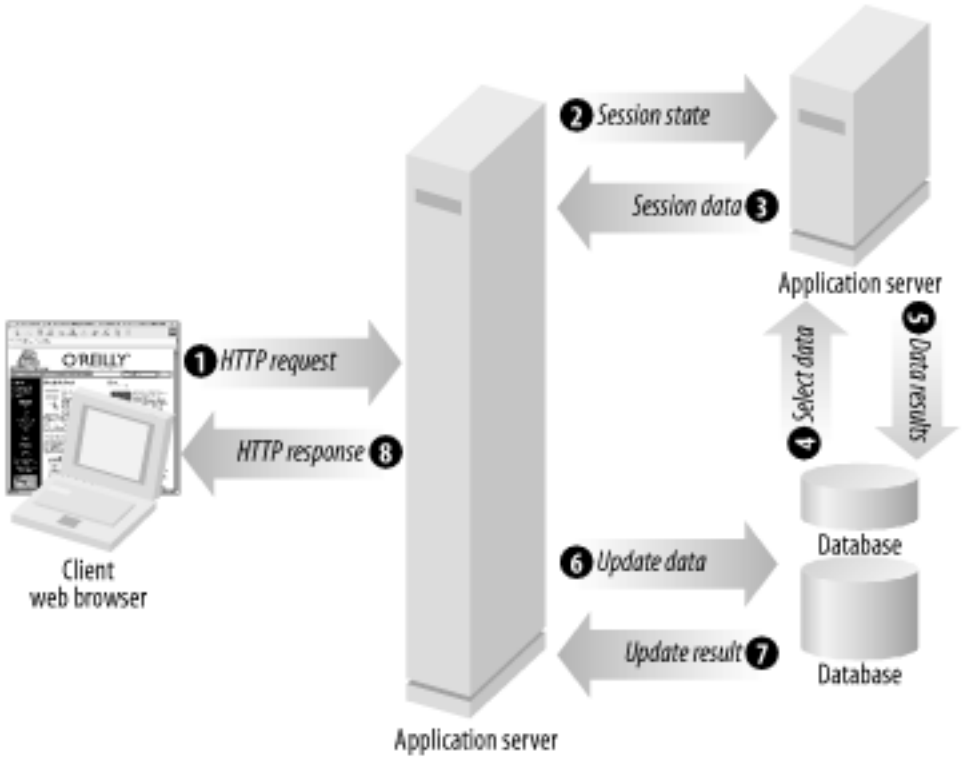
<html>

<head> <title>Test Page for the Apache Web Server on Red Hat Linux</title> </head> <body bgcolor="#ffffff"> (...)

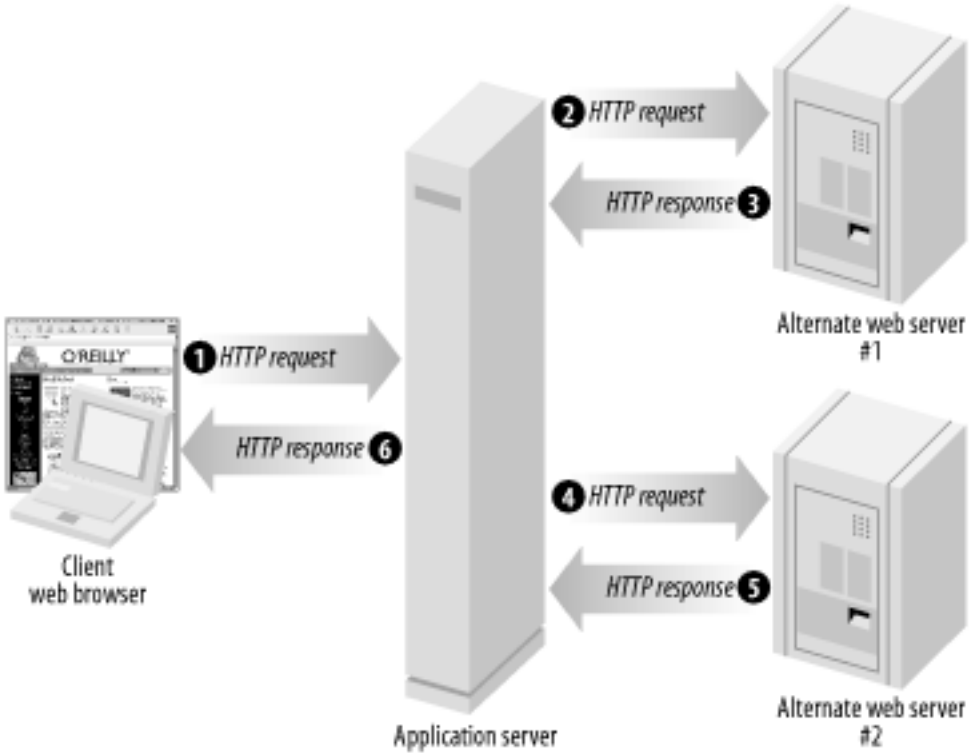
</body>

</html>

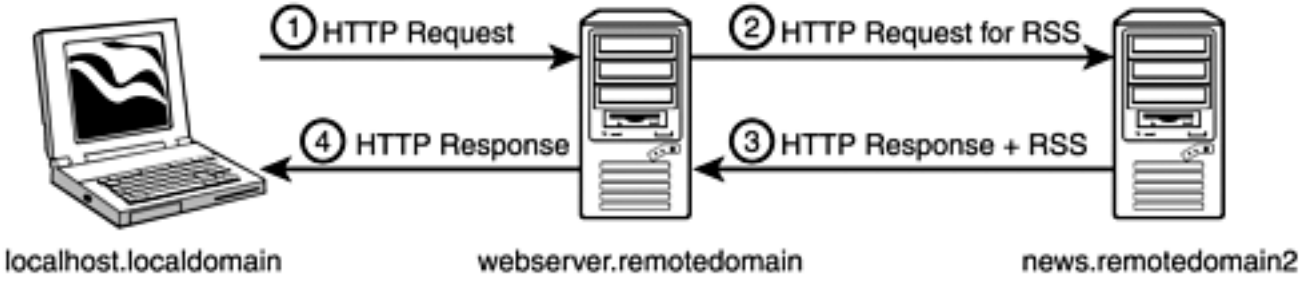
Arquitectura *n* camadas



Informação Base



Informação Base



ISO's OSI stack

Layer	Example(s)
Physical layer	RJ-45 connectors, CAT5
Data Link layer	Ethernet (IEEE 802.2)
Network layer	IP, IPX
Transport layer	TCP, UDP
Session layer	Socket connections
Presentation layer	Byte ordering
Application layer	HTTP, FTP, SMTP, POP3, IMAP, etc.

Informação Base

	Latency	Bandwidth	Reliability
Dial-up	Poor	Poor	Moderate
DSL/cable	Good	Good	Good
Cell phone PDA	Poor	Poor	Poor
802.11x laptop	Good	Moderate	Moderate
Colocated server	Excellent	Excellent	Excellent
Handwritten letter	Poor	Poor	Excellent
A file copied to a floppy	Poor	Poor	Excellent
A burned and priority-mailed DVD	Terrible	Excellent	Excellent

• ASP – Application Service Provision

Informação Base

- Um ASP é uma organização externa fornecedora de componentes de software (serviços), que desenvolve, gere e aloja aplicações de software, cedendo a sua utilização através de aluguer ou leasing.

ASP Industry Consortium

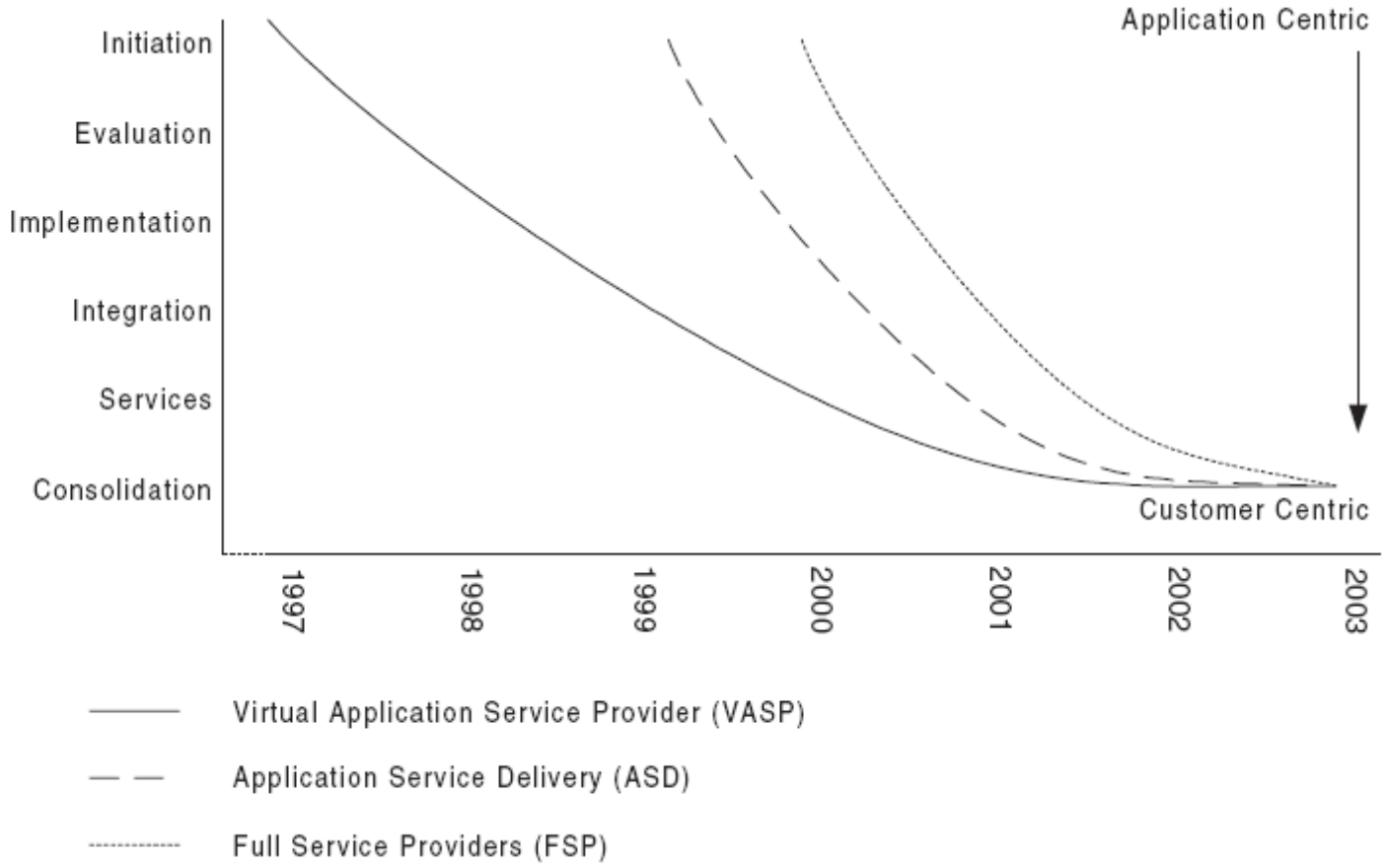
- **Application outsourcing**

- Independência geográfica, cultural, organizacional e técnica.

• ASP – Application Service Provision

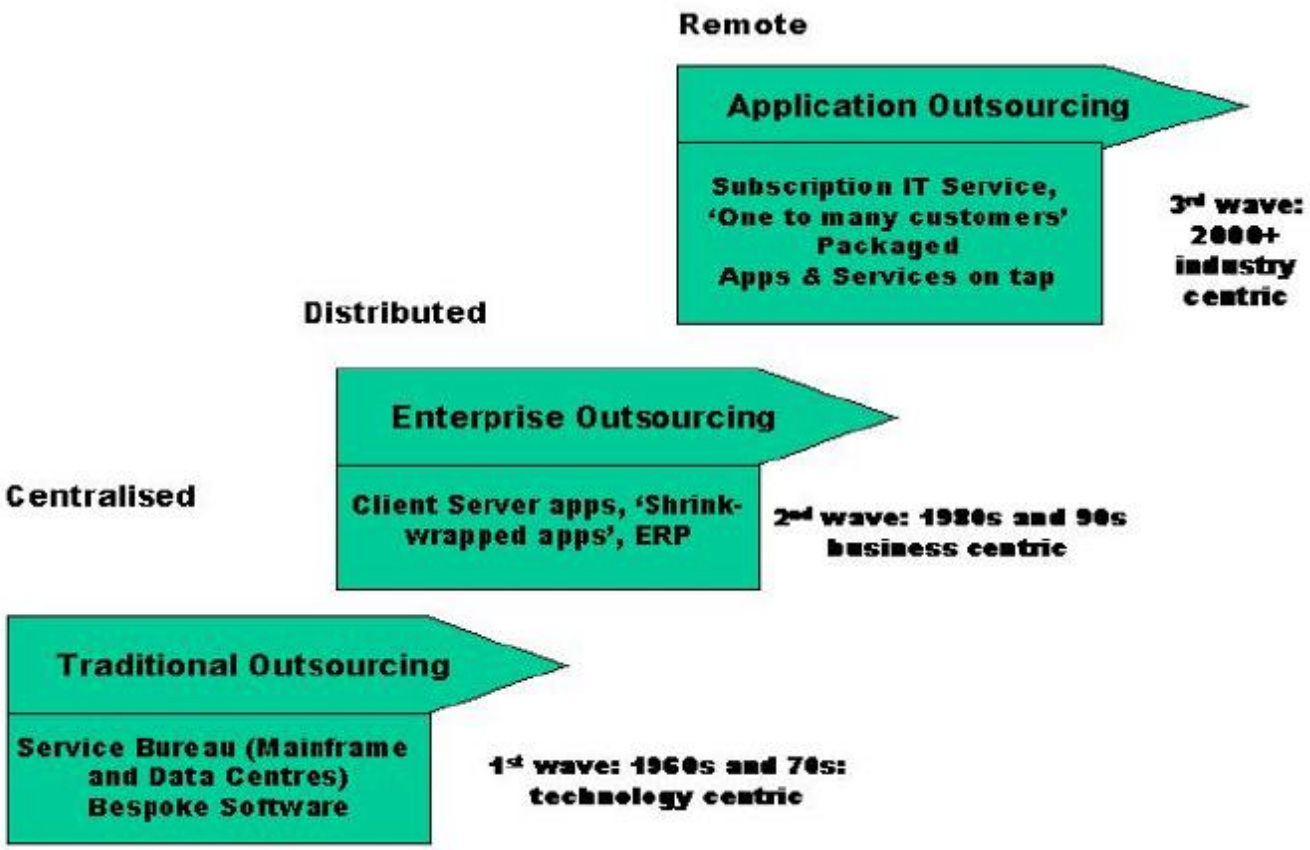
Evolução dos modelos de ASP

Informação Base



Etapas de outsourcing de TI

Informação Base



Informação Base

- Web-based systems and applications (WebApps) deliver a complex array of content and functionality to a broad population of end-users.

- Web Enginnering (WebE) is the process that is used to create high-quality WebApps.

- WebE applies a generic approach that is tempered with specialized strategies, tactics and methods. The WebE process **begings** with a formulation of the problem to be solved by the WebApp. The WebE project is **planned**, and the requirements and the design of the WebApp are modeled. The system is **constructed** using specialized techniques and tools associated with Web.

- *Because WebApp evolve continuously, mechanisms for configuration control, quality assurance, and on-going support must be established.*

Informação Base

- **WebApps attributes:**
  - Network intensiveness
  - Concurrency
  - Unpredictable load
  - Performance
  - Availability
  - Data Driven
  - Content Sensitive
  - Continuous evolution

Informação Base

WebApps attributes:

- **Immediacy** – WebE must use methods for planning, analysis , design, implementation, and testing that have been adapted to the compressed time schedules required for WebApp development.

- **Security**

- **Aesthetics.**

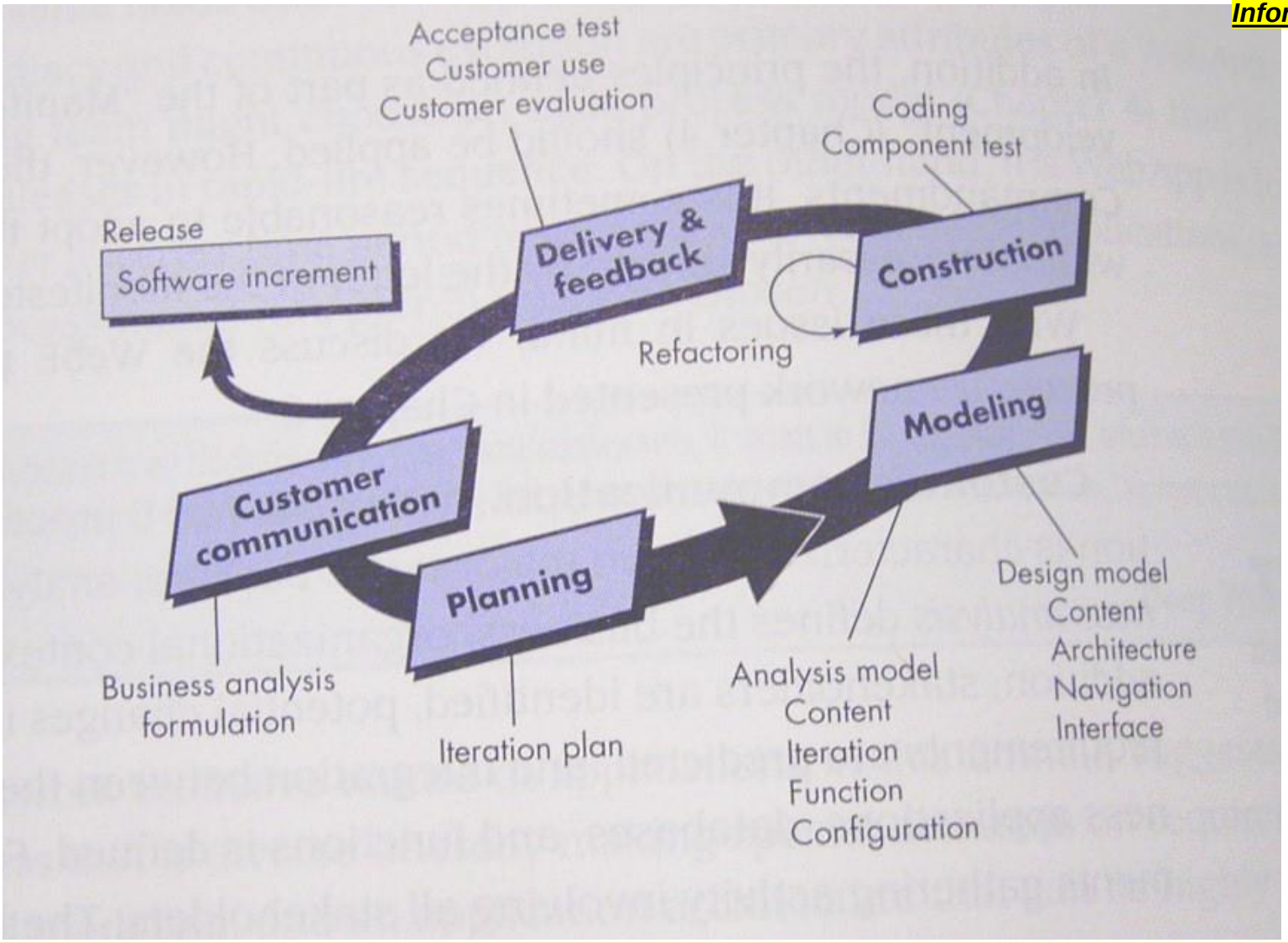
Informação Base

- **Application categories**
  - Informational
  - Download
  - Customizable
  - Interaction
  - User input
  - Transaction-oriented
  - Service-oriented
  - Portal
  - Database access
  - Data Warehousing.

Informação Base

- WebApps are often dilevered incrementally.
- Changes will occur frequently.
- Timelines are short.

Informação Base



Best Practices:

Informação Base

- Take the time to understand business needs and product objectives, even if the details of the WebApps are vague.
- Describe how users will interact with the WebApp using a scenario-based approach.
- Develop a project plan,even if it is very brief.
- Spend some time modelling what it is that you're going to buid.
- Review the models for consistency and quality.
- Use tools and technology that enable you to construct the system as many reusable components as possible.
- Don't rely on early users to debug the WebApp – design comprehensive tests and execute them therefore releasing the system

- Web Engineering
  - Planning for WebE Projects

Informação Base

	Traditional projects	Small e-Projects	Major e-Projects
Requirements gathering	Rigorous	Limited	Rigorous
Technical specifications	Robust: models, spec	Descriptive overview	Robust: UML models, spec
Project duration	Measured in months or years	Measured in days, weeks or months	Measured in months or years
Testing and QA	Focused on achieving quality targets	Focused on risk control	SQA as described in Chapter 26
Risk management	Explicit	Inherent	Explicit
Half-life of deliverables	18 months or longer	3 to 6 months or shorter	6 to 12 months or shorter
Release process	Rigorous	Expedited	Rigorous
Post-release customer feedback	Requires proactive effort	Automatically obtained from user interaction	Obtained both automatically and via solicited feedback

- **WebE team**

  - Content developers/providers
  - Web publisher
  - Web Enginneer
  - Business domain experts
  - Support specialist
  - Administrator.

Informação Base

Design issues for WebE

- Security
- Availability
- Scalability
- Time-to-market

Design Goals

- Simplicity
- Consistency
- Identity
- Robustness
- Navigability
- Visual Appeal
- Compatibility

Design Activities

- Interface design
- Aesthetic Design
- Content Design
- Navigation design
- Architecture design
- Component design

• Service-Oriented Architecture (SOA)

Informação Base

- O termo **"Service-Oriented Architecture" (SOA)** ou Arquitectura Orientada a Serviços expressa um conceito de arquitectura onde aplicativos ou funções são disponibilizadas como serviços numa rede de computadores (Internet ou Intranets) de forma independente utilizando padrões abertos para comunicação. A maior parte das implementações de SOA utilizam de Web services (SOAP, WSDL,…). Entretanto, uma implementação de SOA pode fazer uso de qualquer tecnologia padronizada baseada na web.

- **Serviço** É uma função independente, sem estado (stateless) que aceita uma ou mais requisições e retorna uma ou mais respostas através de uma interface padronizada e bem definida. Serviços podem também realizar partes discretas de um processo tal como editar ou processar uma transacção. Serviços não devem depender do estado de outras funções ou processos. A tecnologia utilizada para prover o serviço, tal como uma linguagem de programação, não pode fazer parte da definição do serviço.

- Programação Cliente/Servidor
- Programação Distribuída
- Portabilidade de código
- Applets
- Servlets
- …

• Service-Oriented Architecture (SOA)

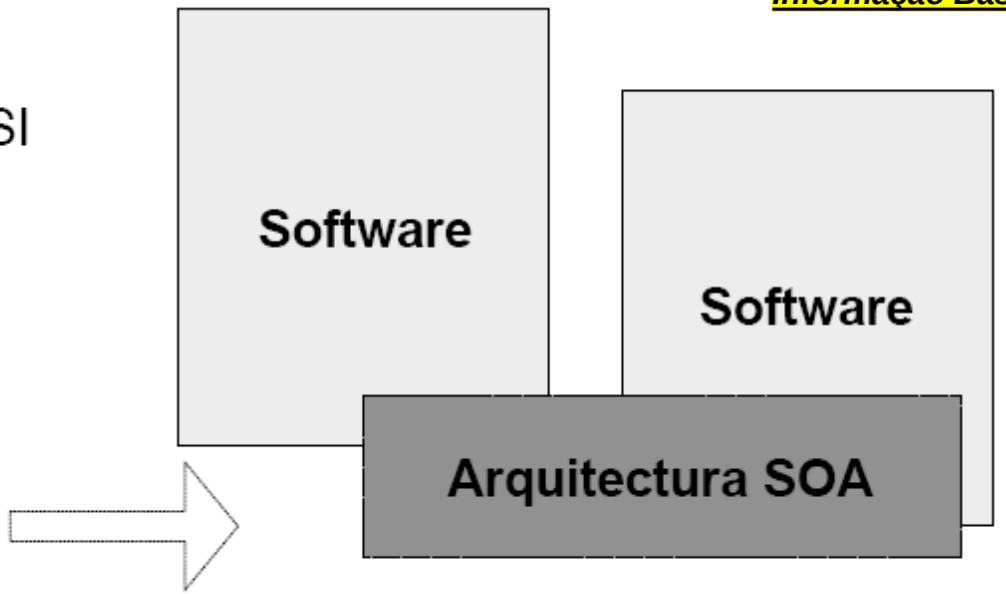
Informação Base

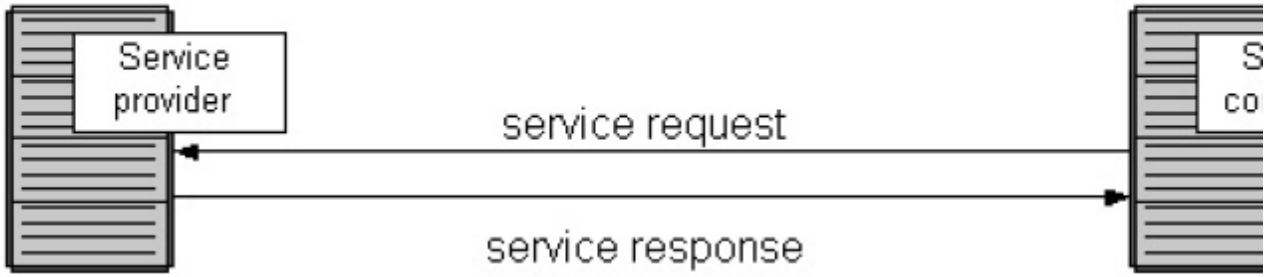
- Tal como *objectos* e componentes, os *serviços* representam organizações de código que nos permitem organizar capacidades numa forma conhecida e controlada
- Tal como *objectos* e *componentes*, os serviços:
  - Combinam informação com comportamento
  - Escondem estado interno da utilização externa
  - Apresentam um interface simples com o exterior
- *Objectos* utilizam abstracção de tipos e dados; os *serviços* fazem algo semelhante com orientação ao aspecto ou contexto.
- *Objectos* e *componentes* estão organizado em classes ou hierarquias com herança de comportamentos; os *serviços* podem ser publicados e "consumidos" simples ou como hierarquias ou colaborações;

• Service-Oriented Architecture (SOA)

Informação Base

- Necessidade de Integração de SI
- eBusiness (B2B)
- Redução de Custos
- Redução Tempos de produção
- Integração Inter-empresas
- Modelos de negócio adaptativos
- Respostas rápidas
- Maior *ROI*





• Service-Oriented Architecture (SOA)

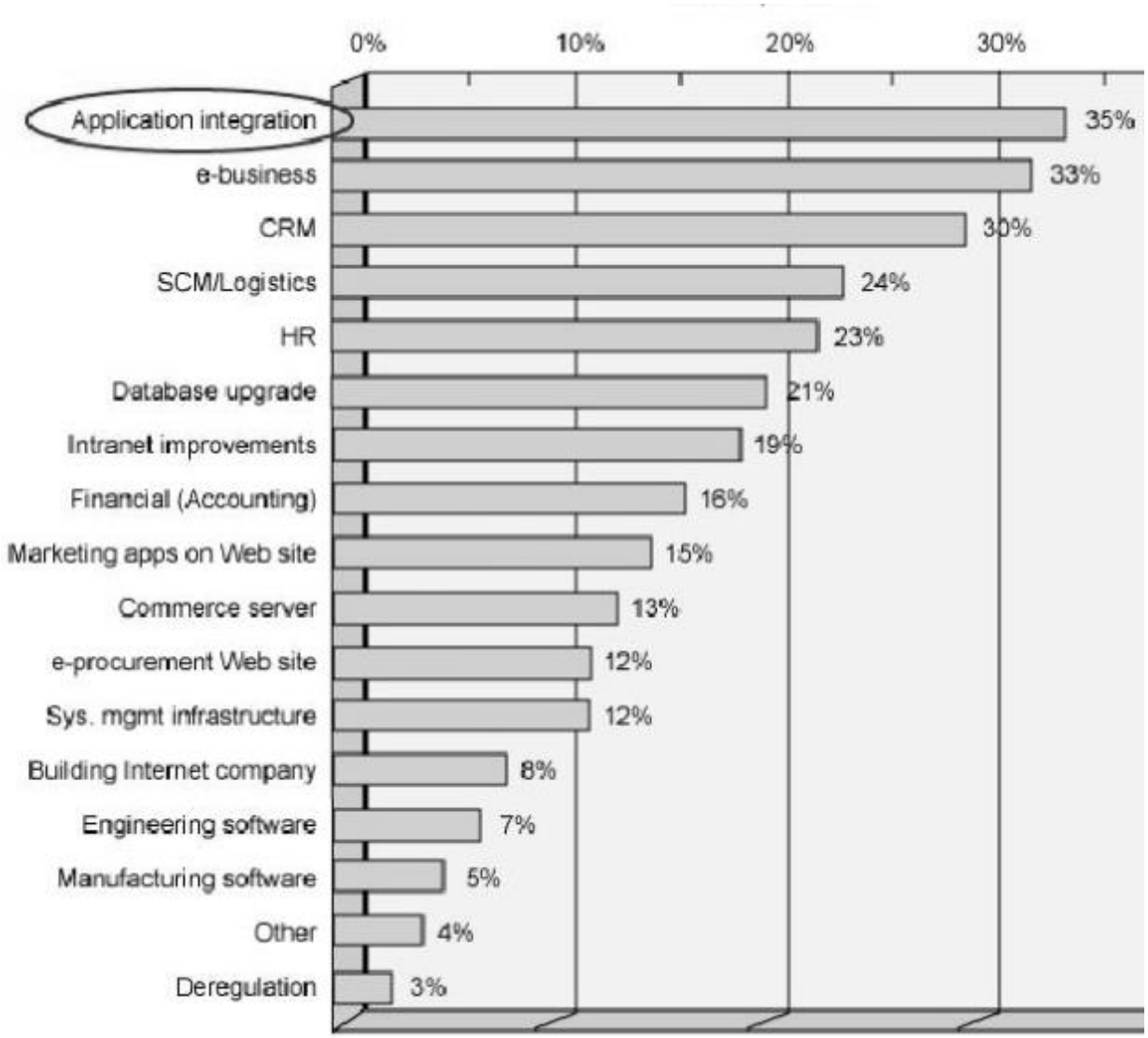
Informação Base

- Fusão/Integração de empresas
- Gestão corporativa
- Integrar sistemas legados
- Soluções pontuais
- Heterogeneidade
- SO; Hardware; LP; Middleware; BD
- Respostas rápidas
- Complexidade

Prioridade → INTEGRAR

• Service-Oriented Architecture (SOA)

Informação Base



• Service-Oriented Architecture (SOA)

Informação Base

- Potenciar os recursos existentes
- Suportar todo o tipo de integração
  - Utilizador; Aplicacional; Processos; Informação
- Permitir implementações e migração de recursos de uma forma incremental
- Ambiente de desenvolvimento - *frameworks*
- Permitir novos modelos de computação: *Grid Computing* e *On-Demand Computing*

• Service-Oriented Architecture (SOA)

Informação Base

Uma arquitectura aplicacional dentro da qual todas as funções são definidas como serviços independentes com interfaces de invocação (IDL) bem definidas e que possam ser chamadas em sequências definidas para formar processos de negócio

• Service-Oriented Architecture (SOA)

Informação Base

- Em SOA o *serviço* é a entidade principal
- WebServices são um conjunto de protocolos pelos quais os Serviços podem ser divulgados (publicados), "descobertos" e utilizados de uma forma standard e independente da tecnologia

- Um novo paradigma Web?
- Globalização → migrar de B2C para B2B
- Serviços e Produtos de qualidade a preços competitivos e na devida altura;
- Preocupações com *core businesses* e *outsourcing* de produtos não estratégicos a fornecedores externos;
- Exigem-se eficientes mecanismos de partilha de dados com parceiros de negócio, fornecedores, revendedores, etc.
- Cada entidade é autónoma nos sistemas informáticos...
- São necessários interfaces eficientes. Implementam-se todos? → EAI/IAC

Informação Base

- Porquê Web Services?
- Aparecem para colmatar algumas das principais dificuldades manifestadas com o DCOM e CORBA:
  - Lidar com Firewalls
  - Complexidade dos protocolos
  - Integrar plataformas heterogéneas
- CORBA e DCOM : ideais para aplicações EAI
- CORBA e DCOM: não estão preparadas para escalabilidade, internet, multi-platafroma

Informação Base

" Web Services are **small software components** that are **available over the Internet**. Publishing as Web Services makes the software applications **more reusable and shared by many mores users**. Web Services **enable business partnering** and thereby generate a great way of revenue streaming for the companies. It also helps in **reducing the development, integration, and maintenance cost of the software application**."

Informação Base

- Necessito de conhecer um fornecedor de um produto XPTO com as condições mais vantajosas....vou à Web...estou feito! Um caos!

- Hipótese 1:

  Google →

  Repetir {

  N Fornecedores → Para cada um verificar se é o XPTO que eu quero → ver preços e condições → comparar

  }até estar convencido

- Hipótese 2:

  e-MarketPlace → Categoria de XPTO → item e quantidade → recebe uma lista com todos os fornecedores, preços e condições

Informação Base

- Suponha que um dia adquire um computador que lhe permite o acesso imediato à Internet. A sua dependência à Internet é cada vez maior…

- Caso necessite de escrever umas coisas, terá ao seu dispôr vários processadores de texto, tipo MS Word. Não o tem que ter instalado no seu computador.,.

- Se necessitar de guardar um ficheiro no qual estava a trabalhar, poderá fazê-lo num espaço apropriado, algures numa máquina, devidamente encriptado e protegido.

- Para além disso, conhecer o estado da Bolsa, analisar os movimentos das suas contas serão tarefas vulgares…

Estamos no mundo dos WebServices!

- Problema: Troca de dados e RPC entre entidades
- Podem ser implementados via protocolo HTTP da Internet
- Soluções integradas com uma infra-estrutura comum
- Induz a **novos modelos de negócio** e **reduz custos**
- São *registados* e depois *anunciados*

Definição:

Interface que descreve uma colecção de serviços acessíveis via rede, através de mensagens standards em XML…logo independente da plataforma

Informação Base

- Utilizam a Internet em vez de LANs ou WAN dedicadas
- Universalização da linguagem via utilização de standards – XML
- Abstrai o conteúdo da mensagem, transporte e linguagem de implementação
- Promove a standardização da troca de dados via SOAP – Simple Object Access Protocol
- Contacto directo entre as entidades dispensando terceiros (intermediários)…
- Interface das aplicações normalizam via WSDL – Web Services Definition Language
- Web Services dedicados (para um determinado parceiro) ou universais
- A natureza XML dos WS promove uma chave fundamental de integração: metadados. XML é ideal para descrever conteúdos.

- Um conjunto de normas que definem como descrever, publicar e utilizar determinado tipo de componentes
- Principais normas
  - **SOAP** – um mecanismo (tipo RPC) para invocar métodos sincronamente em componentes remotos
  - **WSDL** – uma linguagem (tipo IDL) para descrever formalmente a interface de um "serviço" (conjunto de métodos oferecidos por um componente)
  - **UDDI** – um serviço (implementado por um componente) que armazena e disponibiliza interfaces de outros serviços
- SOAP é 99% dos Web Services
  - WSDL é invisível e UDDI não é utilizado

- Nível do Negócio
  - Maravilhoso mundo novo que permitirá criar novas aplicações utilizando funcionalidades residentes em aplicações legadas e componentes remotos
  - O modelo ASP (*Application Servive Provider*) prometia aplicações remotas que seriam alugadas
    - Mas falhou redondamente
  - Perguntas
    - Serão os Web Services uma "reinvenção" dos ASP?

      Também os ASP eram baseados em software remoto e alugado
    - Qual o "modelo de negócio" para os Web Services ?

      Também o CORBA prometia um "mercado" de componentes
    - Fará sentido "alugar" componentes remotos ?

Informação Base

- Nível Técnico
  - Os Web Services são apenas uma tecnologia de integração "tipo RPC" baseada em XML
    - Provavelmente os Web Services são apenas outra tecnologia de componentes remotos como o CORBA, EJB e DCOM
    - Não trazem nada de fundamentalmente novo

      Bem pelo contrário, até podem ser considerados um retrocesso tecnológico pois recuperam as tecnologias que deram origem ao falhanço do CORBA
  - Grande Mentira
    - Web – não têm nada a ver com HTML
    - Services – são componentes vulgares com métodos que podem ser invocados remotamente
  - Mas afinal o que é um serviço ??

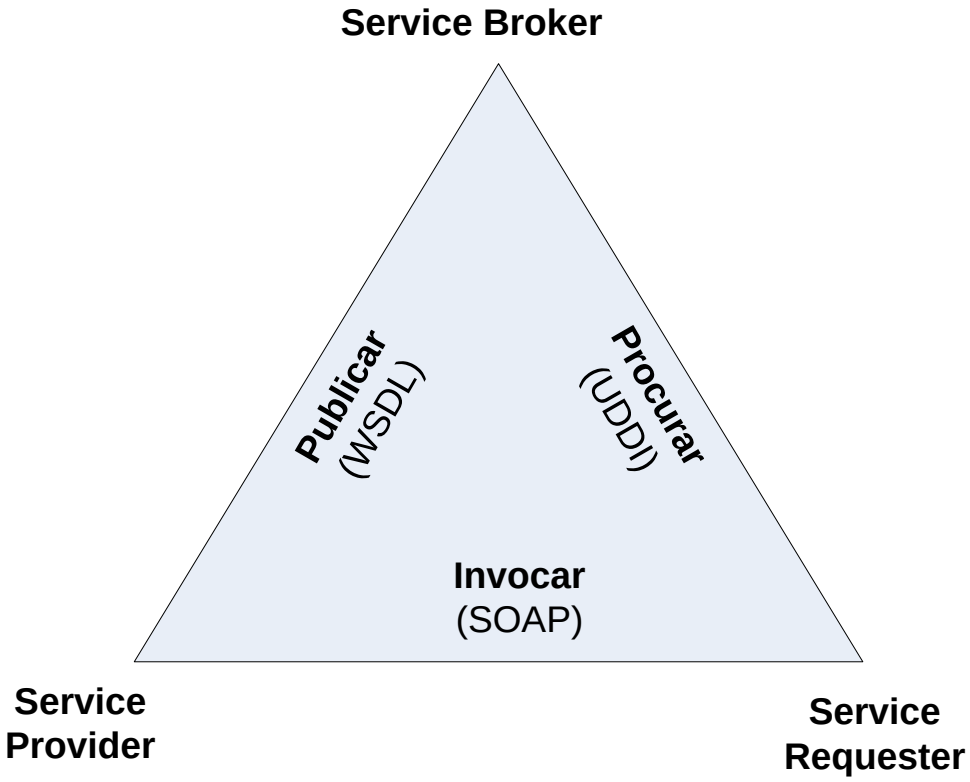
Informação Base

- WS apresentam-se como aplicações modulares, auto-descritivas que:
  - São acessíveis, como se fossem componentes, a partir de qualquer local na internet;
  - recorrem a protocolos standard tais como HTTP, XML, SOAP, WSDL, UDDI;
  - Conseguem funcionar mesmo na presença de firewalls e servidores proxy, uma vez que sendo representados por documentos XML todo o tráfego passa pela porta 80;
  - Podem tirar partido da autenticação do HTTP, assim como das capacidades de integração existentes no Secure Socket Layer (SSL);

Informação Base

- WS apresentam-se como aplicações modulares, auto-descritivas que:
  - combinam as melhores características da poo com a programação para a web;
  - são independentes das plataformas de desenvolvimento;
  - apresentam como resultado um documento XML;
  - permitem a interacção entre aplicações sem intervenção humana;
  - diminuem a complexidade e os custos associados à integração de aplicações.

Informação Base



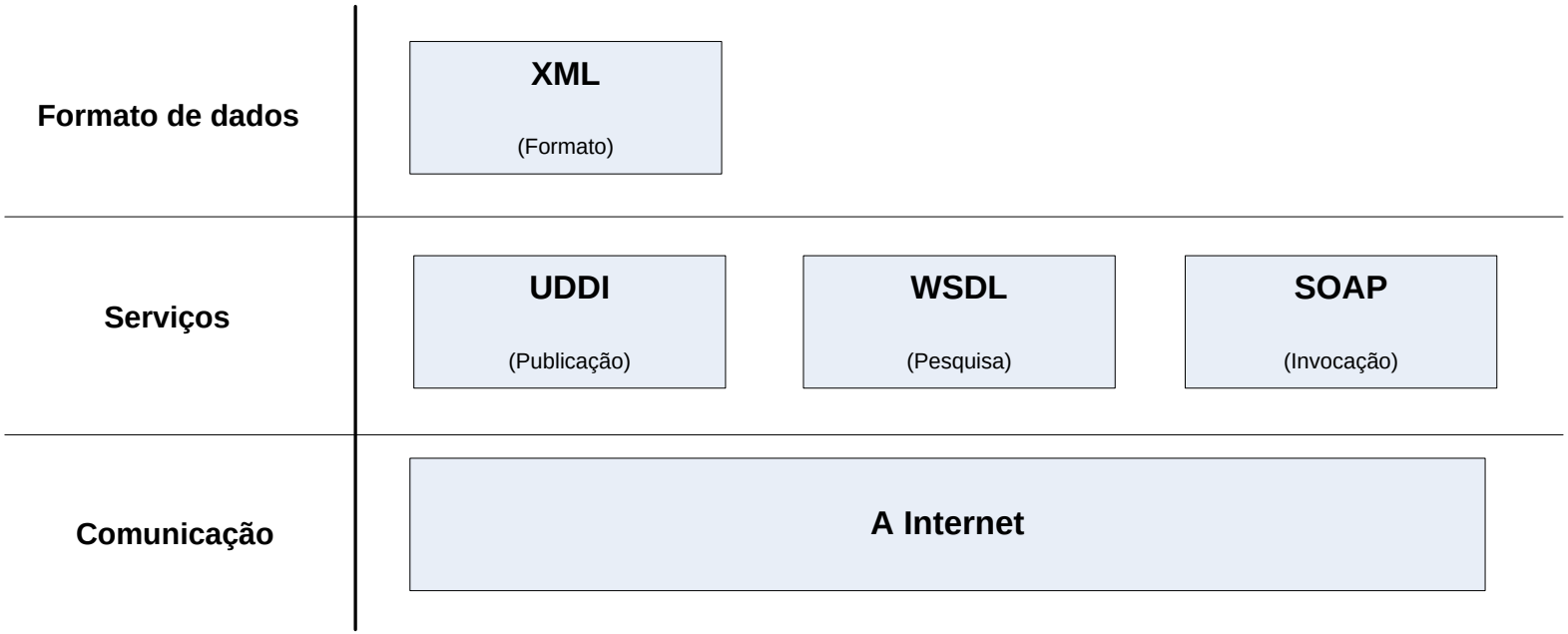
- **Publicação**: processo, opcional, através do qual o fornecedor do WS dá a conhecer a existência do seu serviço, efectuando o registo do mesmo no reservatório de WS (UDDI).

- **Descoberta**: processo, opcional, através do qual uma aplicação cliente faz uma pesquisa sobre o reservatório de WS e toma conhecimento da existência do WS pretendido (UDDI).

- **Descrição**: a aplicação cliente tem a acesso a toda a interface do WS, onde se encontram descritas todas as funcionalidades por ele disponibilizadas, assim como os tipos de mensagens que permitem aceder às ditas funcionalidades (WSDL).

- **Invocação**: processo pelo qual cliente e servidor interagem, através da troca de mensagens (SOAP).

Informação Base



Informação Base

- **Ciclo de vida**

- O fornecedor constrói o serviço utilizando a linguagem de programação que entender;

- De seguida, especifica a interface/assinatura do serviço que definiu em WSDL;

- Após a conclusão dos dois primeiros passos, o fornecedor regista o serviço no UDDI;

- O utilizador (aplicação cliente) encontra o serviço procurando no UDDI;

- A aplicação cliente estabelece a ligação com o WS e estabelece um diálogo com este, via mensagens SOAP.

• Simple Object Access Protocol (SOAP)

Informação Base

- **O SOAP foi criado para ser um protocolo simples (pouco exigente), baseado em XML e destinado à troca de informação tipada e estruturada entre aplicações distribuídas e descentralizadas.**

- O SOAP não define um protocolo de transporte para o envio de mensagens, apesar de possuir uma descrição de como deve ser utilizado sobre HTTP (protocolo standard para a Internet).

Informação Base

Mensagem SOAP

- O intercâmbio de informação é efectuado através de mensagens em formato XML, estando a cargo do SOAP a definição desse mesmo formato.

- Uma mensagem é essencialmente uma transmissão unidireccional no sentido

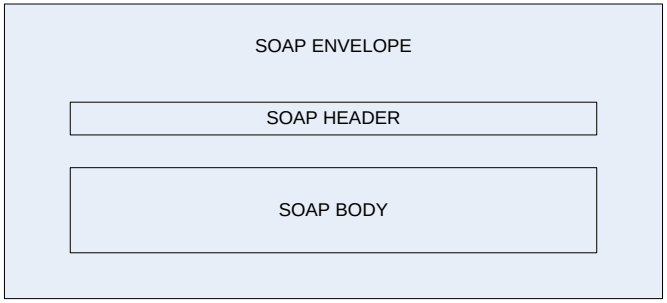
emissor → receptor.

Informação Base

Mensagem SOAP

- Analogia com uma carta de correio.

Mensagem SOAP



- O envelope envolve toda a mensagem sendo constituído por um cabeçalho e um corpo.

- Dentro do **corpo** encontramos o conteúdo (informação da mensagem);

- No **cabeçalho**, a identificação sobre a natureza desse mesmo conteúdo.

• Simple Object Access Protocol (SOAP)

Informação Base

SOAP 1.2

The following is a sample SOAP 1.2 request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /webservices/Math.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XM
  <soap12:Body>
    <soma xmlns="http://tempuri.org/">
      <a>float</a>
      <b>float</b>
    </soma>
  </soap12:Body>
</soap12:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XM
  <soap12:Body>
    <somaResponse xmlns="http://tempuri.org/">
      <somaResult>float</somaResult>
    </somaResponse>
  </soap12:Body>
</soap12:Envelope>
```

• Simple Object Access Protocol (SOAP)

Informação Base

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

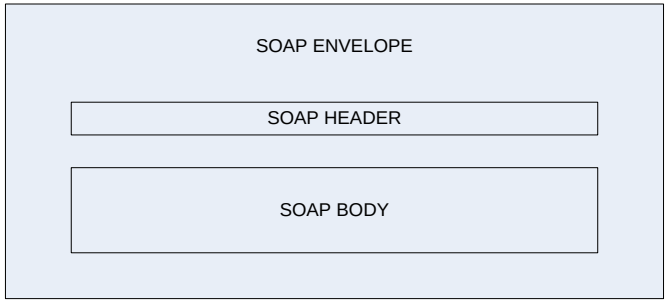
```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

Mensagem SOAP



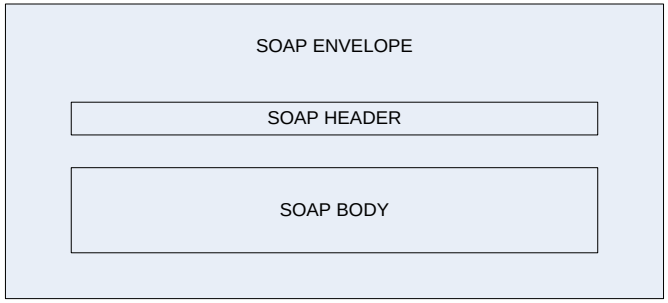
- Duas partes: emissor e receptor;

- A cada **receptor** que executa algum processamento sobre a mensagem, chamamos **endpoint** ou **ponto terminal**. É da responsabilidade do ponto terminal a análise da mensagem e a remoção da parte que lhe é endereçada.

- Cada ponto terminal pode ser simultaneamento receptor e emissor, sendo então designado por **intermediário**.

• Simple Object Access Protocol (SOAP)

- **Mensagem SOAP**



- Cada ***endpoint*** executa três operações:

- Análise da mensagem e verificação se alguma parte da mesma lhe é endereçada;

- Verificação se alguma das partes que lhe são dirigidas é de execução obrigatório. Se conseguir executar as operações processa a mensagem, caso contrário rejeita-a.

- Se se trata de um intermediário, então é necessário retirar todas as partes identificadas na 1.ª operação e só depois enviar a mensagem ao próximo ponto terminal.

• Simple Object Access Protocol (SOAP)

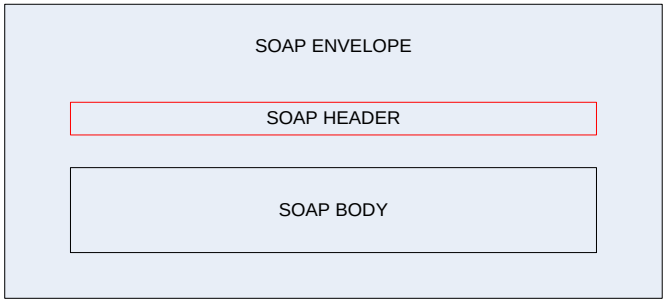
Mensagem SOAP

- Envelope

- Elemento raíz de uma mensagem SOAP, correspondendo à descrição da mensagem e do que deve ser processado.

• Simple Object Access Protocol (SOAP)

- **Mensagem SOAP**



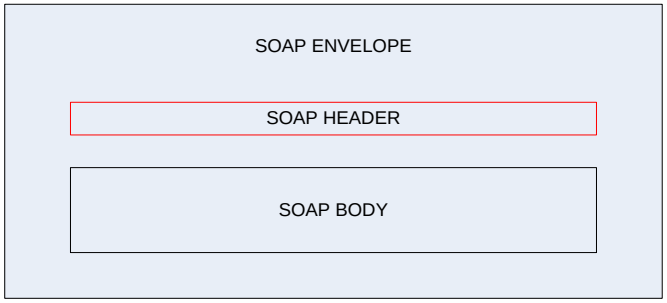
- **Header**

- Apesar de todas as potencialidades do SOAP (algumas a ver com independência da plataforma de desenvolvimento) apresenta algumas limitações a nível de segurança, reencaminhamento e transacções.

- É possível adicionar extensões que permitem adicionar funcionalidades não implementadas pelo SOAP

• Simple Object Access Protocol (SOAP)

- **Mensagem SOAP**



- **Header**

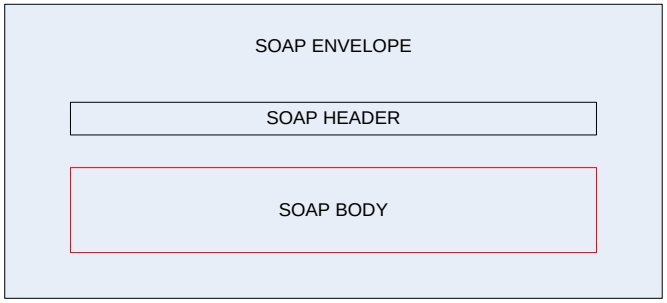
- O elemento Header é opcional e tem como função estender as funcionalidades das mensagens SOAP, sem necessidade de alteração das especificações do SOAP.

- Autenticação; transacções; encriptação.

- A existir tem de ser o primeiro filho do elemento envelope.

• Simple Object Access Protocol (SOAP)

- **Mensagem SOAP**



- **BODY**

- Elemento obrigatório.

- Dentro do corpo de uma mensagem podemos encontrar desde simples chamadas a métodos, contendo informação em formato XML que se pretende transferir através de uma mensagem.

• Simple Object Access Protocol (SOAP)

- **SOAP sobre HTTP**

- Existe uma clara separação entre a definição da mensagem e o transporte da mesma.

- Cada pedido SOAP sobre HTTP é enviado num pedido HTTP (HTTP POST),

```
POST /webservices/Math.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/soma"
```

- **SOAP sobre HTTP**

```
POST /webservices/Math.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/soma"
```

- Indica o método HTTP a ser utilizado, assim como URI onde se encontra o ponto terminal em questão, e a versão HTTP que está a ser utilizada.

`Content-Type` indica qual o tipo de conteúdo da mensagem, o que no caso de uma mensagem SOAP é sempre um conteúdo do tipo “text/xml”, uma vez que cada mensagem SOAP é um documento XML;

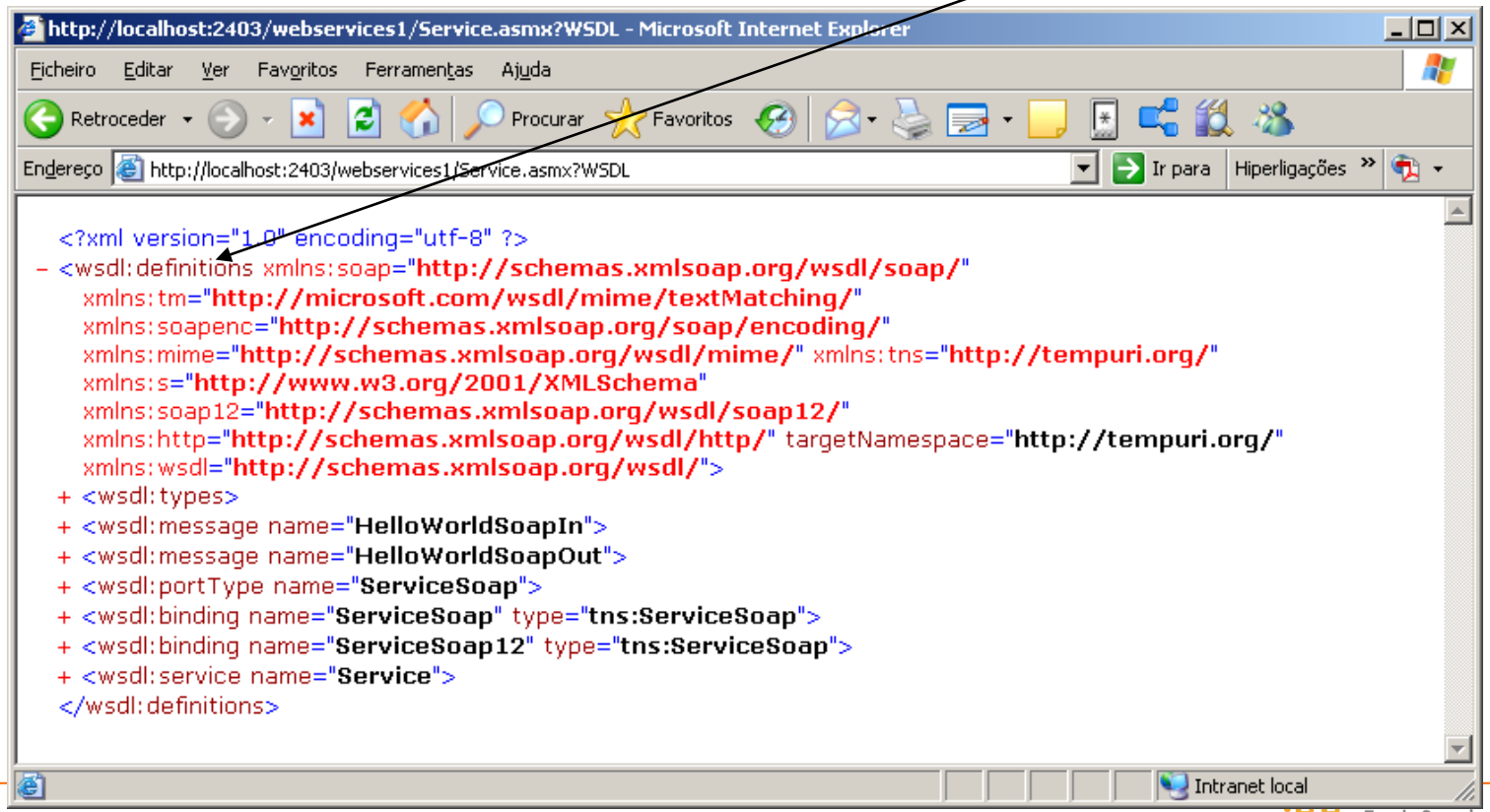
`Content-Length` indica o tamanho de bytes do conteúdo da mensagem;

`SOAPAction` é um URI que indica o objectivo da mensagem.

Necessidade dos WS serem aplicações auto-descritivas.

> Formato XML para descrever WS e a forma de os invocar.

Elemento raíz



• Web Services Description Language (WSDL)

■ Elementos

definitions (elemento raíz do documento)

types (identificação dos tipos de dados dentro das mensagens

message (assinatura de pedido ou de resposta para cada método)

portType (colecção de operações (métodos) expostos pelo WS)

operation (uma operação é um conjunto composto por uma mensagem de *input*, uma de *output* e uma outra opcional de erro; corresponde a cada um dos métodos a serem invocados por uma aplicação cliente)

binding (indicação de qual o protocolo de transporte a utilizar)

service (identifica os diferentes pontos de acesso do WS a invocar)

port (identificação do endereço do WS)

• Web Services Description Language (WSDL)

■ Elementos

definitions (elemento raíz do documento)

```
- <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://tempuri.org/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://tempuri.org/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
```

• Web Services Description Language (WSDL)

■ Elementos

types (elemento utilizado para descrever os tipos de dados contidos nas mensagens)

```
- <wsdl:types>
  - <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
    - <s:element name="HelloWorld">
        <s:complexType />
      </s:element>
    - <s:element name="HelloWorldResponse">
      - <s:complexType>
        - <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="HelloWorldResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
```

• Web Services Description Language (WSDL)

■ Elementos

message (uma mensagem é constituída por um conjunto de parâmetros ou resultados representados pela notação *part*. Cada parâmetro encontra-se associado a um dos *types* previamente definidos, ou a qualquer tipo de dados pré-definidos na especificação da linguagem de *schemas* adoptada.)

```
- <wsdl:message name="HelloWorldSoapIn">
    <wsdl:part name="parameters" element="tns:HelloWorld" />
  </wsdl:message>
- <wsdl:message name="HelloWorldSoapOut">
    <wsdl:part name="parameters" element="tns:HelloWorldResponse" />
  </wsdl:message>
```

• Web Services Description Language (WSDL)

■ Elementos

operation (não é possível associar mensagens de forma a obter-se um par Pedido/Resposta. Na WSDL esta associação é feita através do elemento *operation*, o qual indica qual a mensagem correspondente à resposta)

```
- <wsdl:operation name="HelloWorld">
    <wsdl:input message="tns:HelloWorldSoapIn" />
    <wsdl:output message="tns:HelloWorldSoapOut" />
  </wsdl:operation>
```

• Web Services Description Language (WSDL)

■ Elementos

portType (consiste num conjunto de mensagens agrupadas por operações, i.e., é um conjunto de operações (métodos) disponibilizadas pelo WS)

```
- <wsdl:portType name="ServiceSoap">
  - <wsdl:operation name="HelloWorld">
      <wsdl:input message="tns:HelloWorldSoapIn" />
      <wsdl:output message="tns:HelloWorldSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
```

• Web Services Description Language (WSDL)

■ Elementos

binding (independência do protocolo de transporte, no entanto torna-se necessário associar a cada operação a identificação do protocolo de transporte que será utilizado. O elemento *binding* define a forma como é estabelecida a ligação entre as operações e os respectivos protocolos de transporte)

portType

```
- <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  - <wsdl:operation name="HelloWorld">
      <soap:operation soapAction="http://tempuri.org/HelloWorld" style="document" />
    - <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    - <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
```

Elementos

soap:binding (indica que o *binding* será feito através de SOAP. O atributo *transport* indica qual o protocolo de transporte a ser utilizado pelas mensagens SOAP. O único valor definido é o http://schemas.xmlsoap.org/soap/http, o qual indica que o HTTP é o protocolo de transporte a utilizar.

O atributo opcional *style* indica o tipo de mensagem que a operação representa. O valor *document* (valor por defeito) indica que as mensagens de pedido/resposta serão documentos XML, enquanto que o valor rpc especifica um formato RPC para as mensagens.)

• Web Services Description Language (WSDL)

■ Elementos

soap:operation (indica a forma como é feito o *binding* de determinada operação, com determinada implementação SOAP.)

soap:body (este elemento permite especificar os detalhes das mensagens de *input e output*, a serem incluídas no corpo da mensagem SOAP. O valor *literal* no atributo *use* significa que o corpo da mensagem está em texto de formato arbitrário, ou seja , é apenas XML)

• Web Services Description Language (WSDL)

■ Elementos

port e service (um *port* identifica o endereço do WS ao qual pretendemos aceder, consistindo num ponto de acesso ao WS em causa. Cada *port* tem um atributo *binding* previamente definido. Um *service* é um conjunto de *port*, que representa o WS que se pretende disponibilizar)

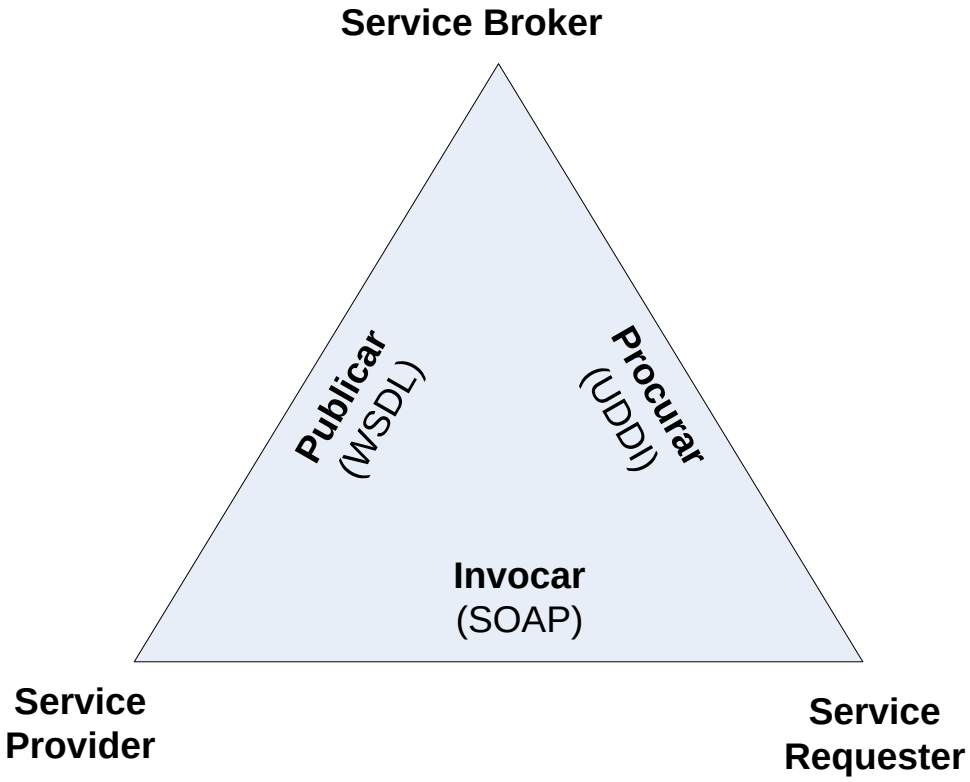
```
- <wsdl:service name="Service">
  - <wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
      <soap:address location="http://localhost:2403/webservices1/Service.asmx" />
    </wsdl:port>
  - <wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
      <soap12:address location="http://localhost:2403/webservices1/Service.asmx" />
    </wsdl:port>
  </wsdl:service>
```

• Web Services Description Language (WSDL)

■ Primitivas de transmissão

- **Operação de Pedido** (o servidor recebe um pedido, mas não envia resposta)
- **Operação de Pedido/Resposta** (o servidor recebe uma mensagem e envia uma resposta)
- **Operação de Solicitação/Resposta** (o servidor envia uma mensagem à aplicação cliente e recebe desta uma mensagem de resposta – ex.º pedido periódico de identidade por parte de um servidor a um cliente que com ele esteja ou pretenda comunicar)
- **Operação de Notificação** (uma notificação consiste numa mensagem enviada pelo servidor, e para a qual não é esperada nenhuma resposta – ex.º envio de uma mensagem relativa a uma recuperação de uma *password* esquecida))

- Universal Description, Discovery and Integration (UDDI)



- **IBM, Microsoft e Ariba**

- Método Standard capaz de divulgar e descobrir os WS, independente das plataformas de desenvolvimento.

- UDDI – base de dados replicada de informação sobre WS -> várias empresas a fazerem o hosting de um UDDI público.

- **Funções**

- Divulgação: debruça-se sobre a forma como a entidade fornecedora de serviços se regista e regista os seus WS;

- Procura: preocupa-se com a maneira pela qual uma aplicação cliente encontra a especificação do WS e/ou do seu fornecedor;

- Mapeamento (Bind): trata da forma como uma aplicação cliente invoca e interage com um WS após a sua localização.

- **Categorias**

- White Pages: contêm a identificação básica da empresa que fornece o WS, tal como nome, morada, telefone, descrição da empresa, etc. É possível também encontrar uma lista de identificadores alternativos pelos quais a empresa pode ser reconhecida;

- Yellow Pages: contêm a informação quer do WS quer do seu fornecedor registada por categorias em que estes se inserem, como por exemplo, tipo de serviço e localização geográfica da entidade fornecedora do serviço;

- Green Pages: contêm a informação técnica que permite descrever o comportamento e funções disponibilizadas pelo serviço. Nestas páginas encontram-se também informações relativas à localização do WS.

- **Repositórios**

- Repositórios UDDI de teste

http://test.uddi.microsoft .com

https://uudi.ibm.com/testregistry/registry.html

http://udditest.sap.com

- **Modelo de dados do UDDI**

- Registo organizado por categorias, que vão desde a informação relativa à entidade fornecedora até à informação técnica do serviço.

- 4 tipos principais de informação:

`businessEntity` (descrever a informação relativa ao fornecedor do serviço – White Pages)

`businessService` (contém a informação relativa a um determinado WS de entre os diversos WS disponibilizados pela entidade em questão – Yellow Pages)

`bindingTemplate` (informação técnica sobre o WS – Green Pages)

`tModel` (indicar a localização da informação técnica do WS em causa)

- **UDDI API**

- Registo e invocação ao UDDI podem ser feitos através de uma interface programática.
- API do UDDI está dividida em 2 tipos: API de registo e API de invocação.

- API de Registo
- Autenticação dos utilizadores (necessária à manipulação dos dados de registo);
- Manipulação dos dados.

- **Implementações UDDI**

- Microsoft UDDI SDK

http://msdn.microsoft.com/library/en-us/uudi/uudi/about_the_microsoft_uddi_sdk.asp

- IBM UDDI4J (biblioteca de classes JAVA que disponibilizam uma API para interagir com os diversos repositórios UDDI)

http://oos.software.ibm.com/developerworks/projects/uuddi4j