|  |  |
| --- | --- |
| DSE – Departamento de Sistemas Educacionais | |
| Logo AccSys Abr15 | Logo MIT Abr15 |

**ESPECIFICAÇÃO TÉCNICA**

**Projecto ISUTC Fénix -Primavera**

**Versão 0.1**

ÍNDICE

[Apresentação da equipe 4](#_Toc428263437)

[Controlo de versões 5](#_Toc428263438)

[Introdução 6](#_Toc428263439)

[Objectivos 7](#_Toc428263440)

[Âmbito 8](#_Toc428263441)

[Pressupostos 9](#_Toc428263442)

[Arquitectura da Integração 10](#_Toc428263443)

[Descrição 10](#_Toc428263444)

[Componentes 10](#_Toc428263445)

[WCF Service 11](#_Toc428263446)

[Operação 1 – Criar Aluno 11](#_Toc428263447)

[Funcionalidades da operação 11](#_Toc428263448)

[Service Layer 12](#_Toc428263449)

[Operação 2 – Actualizar Aluno 13](#_Toc428263450)

[Funcionalidades da operação 13](#_Toc428263451)

[Service Layer 14](#_Toc428263452)

[Operação 3 – Criar Factura 15](#_Toc428263453)

[Funcionalidades do Método 16](#_Toc428263454)

[Service Layer 17](#_Toc428263455)

[Operação 4 – Criar ou Actualizar Contrato 18](#_Toc428263456)

[Funcionalidade do Método 18](#_Toc428263457)

[Service Layer 19](#_Toc428263458)

[Operação 4 – Consultar Conta 19](#_Toc428263459)

[Funcionalidade do Método 19](#_Toc428263460)

[Service Layer 20](#_Toc428263461)

[Possíveis erros - PrimaveraResultStructure 20](#_Toc428263462)

[Diagrama de pacotes e de classes 23](#_Toc428263463)

[ERP Primavera 24](#_Toc428263464)

[Campos de Utilizador 24](#_Toc428263465)

[Tabelas do Utilizador 24](#_Toc428263466)

[TDU\_Semestre 24](#_Toc428263467)

[TDU\_ParametrosISUTC 25](#_Toc428263468)

[Explicação dos Conceitos 26](#_Toc428263469)

[Webservice 26](#_Toc428263470)

[WCF – Windows Communication Foundation 26](#_Toc428263471)

[Self-Hosting 27](#_Toc428263472)

[Self-Hosting Windows Communication Foundation Services (WCF) 27](#_Toc428263473)

[Serviço ISS - Internet information service 27](#_Toc428263474)

[SOAP (Simple Object Access Protocol) 27](#_Toc428263475)

[Tecnologia de Integração PRIMAVERA 27](#_Toc428263476)

[Motores das Aplicações 27](#_Toc428263477)

[Tecnologia COM e Arquitectura Windows DNA 28](#_Toc428263478)

[Objectos de Negócio (BE) 28](#_Toc428263479)

[Serviços de Negócios (BS) 28](#_Toc428263480)

[Serviços de Dados (DS) 29](#_Toc428263481)

[Validação do Documento 29](#_Toc428263482)

# 

# Apresentação da equipe

|  |  |  |
| --- | --- | --- |
| **Nome** | **Organização** | **Responsabilidade** |
| **Cameron Smith** | Transcom | Gestor do Projecto |
| **Carlos Mahumane** | Transcom | Programador |
| **Danilo Cassy** | Transcom | Participante |
| **Gerson Zandamela** | Transcom | Programador |
| **Guimarães Mahota** | Accsys | Programador |
| **Rogério Melo** | Accsys | Gestor do Projecto |
| **Ted Fernandes** | Transcom | Programador |

# 

# Controlo de versões

|  |  |  |
| --- | --- | --- |
| Versão | Data | Detalhes |
| 0.1 | 13/08/2015 | Criação do documento com base na especificação funcional V 0.2 |

# Introdução

O ISUTC Fénix-Primavera é um projecto que consiste na ligação fixa e integrada de duas plataformas nomeadamente o FénixEDU e o Primavera ERP.

O sistema FénixEDU é uma plataforma integrada que funciona em todos os níveis do processo académico. Enquanto, o primavera ERP é uma solução global de gestão que fortalece a performance das organizações. É de vital importância pois permite o acesso directo a um fluxo de informação relativo a todos os aspectos do negócio da organização, dividindo por várias áreas.

# Objectivos

Este projecto tem como objectivos:

* Descrever os procedimentos técnicos usados para interligar o Fénix e o Primavera;
* Detalhar o funcionamento interno dos métodos aplicados na construção do *WebService*;
* Parametrizar os tipos de dados usados pelas duas plataformas;
* Detalhar com clareza possíveis situações de erro e a forma como os mesmos serão tratados;
* Ilustrar como estes processos estão dispostos com base em diagramas.

# Âmbito

Criação de um *Webservice* para comunicação entre o Fénix e o ERP Primavera que permita a implementação de todos os processos, que interferem com o referido ERP e que estão referenciados no documento “Especificação Funcional”.

Assim, os dados existentes no Fénix, quando necessário, serão automaticamente enviados para o *Webservice*. Este deverá ser colocado, preferencialmente, no mesmo servidor onde esta instalado o ERP Primavera.

De uma forma sucinta, pretende-se que os dados circulem do Fénix para o ERP Primavera através do *Webservice* a criar, garantindo-se uma solução compartimentada em que o Fénix comunica com o *Webservice* sobre o protocolo HTTP ou HTTPS e o *Webservice* comunica com o ERP Primavera através da utilização dos Motores Primavera.

# Pressupostos

O Sistema precisa dos seguintes elementos como base de suporte para a sua execução:

* Versão 8.10 do primavera ERP;
* IIS 7.5 ou superior instalado;
* A porta definida no sistema deve estar aberta no Servidor onde estiver instalado o *Webservice*;
* Todas as entidades financeiras criadas na Fénix devem estar criadas no ERP Primavera;

# Arquitectura da Integração

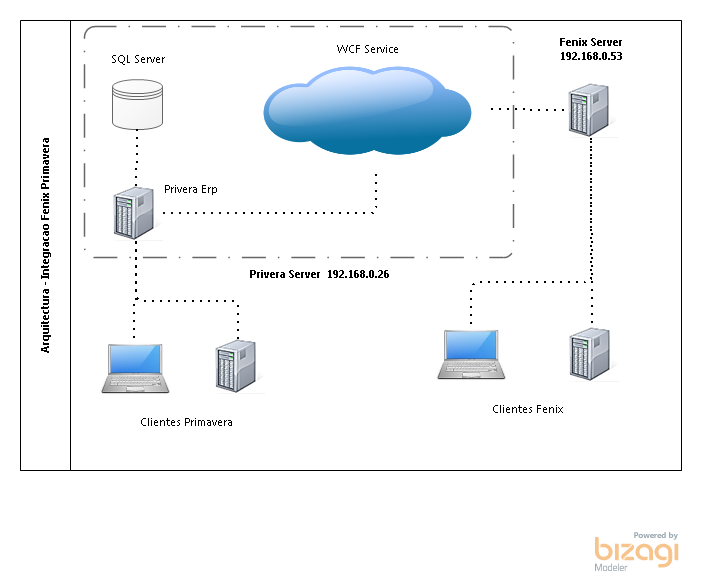


Figure 1: Arquitectura - Integração Fenix Primavera

## Descrição

1. Um Cliente Fénix comunica via HTTP ou HTTPS com o servidor Fénix.
2. Fénix por sua vez, comunica com o Webservice.
3. O Webservice comunica com o ERP primavera através dos motores primavera.
4. O Primavera lê ou escreve na Base de dados SQL Server a partir dos Motores Primavera.

## Componentes

* **Servidor Primavera**
  + Primavera ERP V.8.10
  + SQL 2012 Standard
  + IIS V.7.5
  + Serviço WCF .NET 4.0
  + Framework .NET 4.0
* **Servidor Fénix**
  + Fénix-web-app 3.14.x
  + MySql 5.5
  + Java 8
  + Tomcat7

## WCF Service

O Webservice será uma composto por métodos privados e públicos. Somente as operações/métodos públicos vão estar disponíveis no service layer.

Os métodos/operações públicas serão:

* CriarAluno
* ActualizarAluno;
* CriarFactura;
* CriarOuActualizarContrato;
* ConsultarConta.

### Operação 1 – Criar Aluno

Esta operação será responsável por criar um determinado aluno “**cliente**” (candidato da estudante) através do método ***criarAluno(<inputsListados>)***

#### Funcionalidades da operação

1. **Cria o cliente no ERP Primavera**

* Inicia a transacção na empresa Primavera “**\_erp.IniciaTransaccao()”.**
* Valida-se se o código do cliente existe no Erp Primavera. **\_erp.Comercial.Clientes.Existe (cliente)**
  + Caso sim

Erro = sim, motivo = alunoExiste

* + Caso não exista será referenciada um novo cliente “**\_cliente = new GcpBeCliente()**”.
    - Actualiza-se o cliente “**\_cliente**” com os campos definidos no ***“service layer”.***
* Actualiza-se o cliente “**\_cliente**” os seguintes campos, obedecendo as regras de negócio no Erp Primavera definidas pela Transcom:
* ***Moradas***
  + - **Tipo Terceiro -** este campo esta mapeado com o código da licenciatura
* ***Dados Fiscais***
  + - **Nome Fiscal -** por defeito é igual ao Nome
    - **Espaço Fiscal** - por defeito é Nacional
    - **Regime do IVA -** por édefeito Normal
    - **Pessoa –** por defeito é Singular
* ***Dados Comercias***
  + - **Tipo Preço –** por defeito é P.V.P. 1
    - **Desconto –** por defeito é 0,00%
    - **Condição de Pagamento –** será actualizada consoante o Vendedor (ISU – Próximo dia 10; ITC – Próximo dia 5, EMU – Próximo dia 10, Outros – Factura 30 dias)
    - **Modo Pagamento –** por defeito é REP24 – Recebimento Ponto 24
    - **Moeda –** por defeito é MT – Metical
* Grava-se o cliente a partir do método \_**erp.comercial.clientes.Actualiza (\_cliente, strAvisos)**
* No caso de erro faz ***“roolback”*** das operações “**\_erp.DesfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

1. **Adicionar ou Editar os contactos do cliente**

Este método privado “**insert\_update\_Client\_Contacts(List<Contacts>lista\_contactos)**” será invocado somente após a gravação/edição com sucesso do cliente. O método irá fazer percorrer a lista dos contactos “**foreach( var contactoTemp in lista\_contactos)**” e seguira os seguintes acções:

* Valida se existe o contacto “**\_contacto =\_erp.CRM.Contactos.Existe( contactoTemp.contacto)**”
  + - Caso sim será referenciado o contacto ­“**\_contacto = \_erp.CRM.Contactos.Edita (contactoTemp.contacto)”**;
    - Caso não exista será referenciada um novo contacto “**\_contacto = new CRMBeContacto()**”.
* Actualiza o contacto “**\_contacto**” com os campos definidos no contacto “**contactoTemp**”
* Caso o contacto seja novo associa-se o a Entidade “**\_contactoEntidade**” com base na informação do contacto e do cliente e insere lista de contactos da entidade **“\_contacto.get\_LinhasEntidade().Insere (\_contactoEntidade)**”
* Grava-se o contacto “**\_erp.CRM.Contactos.Actualiza(\_contacto)**”
* No caso de erro faz ***“roolback”*** da operação “**\_erp.DesfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

1. **Criar a respectiva referência bancária**

Este método privado “**create\_update\_Cliente\_Reference(entidadeBancaria,nrCodigo)**” será invocado somente após a criação do cliente ou na mudança de código do estudante.

- Actualizar todos a referência bancária nova e a referência bancária antiga no cliente

- Actualizará no cliente em todos os documentos pendentes a nova referência e a antiga

- No caso de erro faz ***“roolback”*** da operação “**\_erp.DesfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

1. **Fim do método “*ActualizarCliente”***

* No caso de não haver ocorrido erros - Termina a transacção “**\_erp.TerminaTransaccao()**”
* Devolve como resultado uma classe do tipo “***PrimaveraResultStruture()”*** nos seguintes cenários:
  + **Erro** **–** que representa o respectivo erro;
  + **Sucesso –** devolve a referência bancária do estudante dentro mascara #XXXXXXXXX# e A mensagem que representa se foi sucesso completo ou sucesso com avisos;

#### Service Layer

##### Public PrimaveraResultStruture actualizarCliente

|  |  |  |  |
| --- | --- | --- | --- |
| Nome do campo | Descrição | Tipo de Dados/ Tamanho | Valores Permitidos |
| tipoPlataforma(\*) | Tipo de Plataforma Instalada no Servidor | Integer | 0 - Executive  **1 - Profissional** |
| codEmpresa (\*) | Código da Empresa Primavera | String(10) |  |
| codUtilizador (\*) | Código do Utilizador Primavera | String(10) |  |
| Password (\*) | Password do Utilizador Primavera | String(10) |  |
| cliente (\*) | Código do Cliente | String(12) | Alfanuméricos |
| nome (\*) | Nome completo do cliente | String(50) | No numbers |
| nomeFiscal | Nome fiscal do cliente | String(50) | No numbers |
| Desconto | Desconto do cliente | Double | Positive Numbers |
| facMorada (\*) | Morada do cliente na Factura | String(50) |  |
| facLocalidade | Localidade do cliente na Factura | String(20) |  |
| facNrTelefone | Número do Telefone do cliente na Factura | String(20) | (82) 1234567 |
| nuit | NUIT do cliente | String(20) |  |
| vendedor (\*) | Instituto do cliente | String(3) | ISU |
| Observação | Observações sobre o cliente | String(250) |  |
| CDU\_bolsa | Se o cliente tem bolsa | Boolean |  |
| CDU\_geraMulta | Se o cliente vai gerar multas | Boolean |  |
| CDU\_tipoIngresso | O tipo de Ingresso do cliente | String(100) |  |
| CDU\_codLic | O código da licenciatura do cliente | String(50) |  |
| CDU\_turma | O código da turma do cliente | String(50) |  |
| listaContactos | Contactos do cliente | List<Contacto> |  |

### Operação 2 – Actualizar Aluno

Esta operação será responsável por criar ou editar um determinado cliente (candidato, estudante ou instituto) através do método ***actualizarAluno(<inputsListados>)****.*

O Fénix será responsável por gerir a lógica do uso durante nas operações de candidatura, matricula e actualização a nível dos campos obrigatórios na candidatura, matricula e actualização.

#### Funcionalidades da operação

1. **Cria ou Actualiza o cliente no ERP Primavera**

* Inicia a transacção na empresa Primavera “**\_erp.IniciaTransaccao()”.**
* Valida-se se o código do cliente existe no Erp Primavera. **\_erp.Comercial.Clientes.Existe (cliente)**
  + Caso sim será editado o cliente ­“**\_cliente = \_erp.Comercial.Clientes.Edita (cliente)”**;
  + Caso não exista será referenciada um novo cliente “**\_cliente = new GcpBeCliente()**”.
* Actualiza-se o cliente “**\_cliente**” com os campos definidos no ***“service layer”.***
* Actualiza-se o cliente “**\_cliente**” os seguintes campos, obedecendo as regras de negócio no Erp Primavera definidas pela Transcom:
* ***Moradas***
  + - **Tipo Terceiro -** este campo esta mapeado com o código da licenciatura
* ***Dados Fiscais***
  + - **Nome Fiscal -** por defeito é igual ao Nome
    - **Espaço Fiscal** - por defeito é Nacional
    - **Regime do IVA -** por édefeito Normal
    - **Pessoa –** por defeito é Singular
* ***Dados Comercias***
  + - **Tipo Preço –** por defeito é P.V.P. 1
    - **Desconto –** por defeito é 0,00%
    - **Condição de Pagamento –** será actualizada consoante o Vendedor (ISU – Próximo dia 10; ITC – Próximo dia 5, EMU – Próximo dia 10, Outros – Factura 30 dias)
    - **Modo Pagamento –** por defeito é REP24 – Recebimento Ponto 24
    - **Moeda –** por defeito é MT – Metical
* Grava-se o cliente a partir do método \_**erp.comercial.clientes.Actualiza (\_cliente, strAvisos)**
* No caso de erro faz ***“roolback”*** das operações “**\_erp.DesfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

1. **Adicionar ou Editar os contactos do cliente**

Este método privado “**insert\_update\_Client\_Contacts(List<Contacts>lista\_contactos)**” será invocado somente após a gravação/edição com sucesso do cliente. O método irá fazer percorrer a lista dos contactos “**foreach( var contactoTemp in lista\_contactos)**” e seguira os seguintes acções:

* Valida se existe o contacto “**\_contacto =\_erp.CRM.Contactos.Existe( contactoTemp.contacto)**”
  + - Caso sim será referenciado o contacto ­“**\_contacto = \_erp.CRM.Contactos.Edita (contactoTemp.contacto)”**;
    - Caso não exista será referenciada um novo contacto “**\_contacto = new CRMBeContacto()**”.
* Actualiza o contacto “**\_contacto**” com os campos definidos no contacto “**contactoTemp**”
* Caso o contacto seja novo associa-se o a Entidade “**\_contactoEntidade**” com base na informação do contacto e do cliente e insere lista de contactos da entidade **“\_contacto.get\_LinhasEntidade().Insere (\_contactoEntidade)**”
* Grava-se o contacto “**\_erp.CRM.Contactos.Actualiza(\_contacto)**”
* No caso de erro faz ***“roolback”*** da operação “**\_erp.DesfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

1. **Criar a respectiva referência bancária**

Este método privado “**create\_update\_Cliente\_Reference(entidadeBancaria,nrCodigo)**” será invocado somente após a criação do cliente ou na mudança de código do estudante.

- Actualizar todos a referência bancária nova e a referência bancária antiga no cliente

- Actualizará no cliente em todos os documentos pendentes a nova referência e a antiga

- No caso de erro faz ***“roolback”*** da operação “**\_erp.DesfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

1. **Fim do método “*ActualizarCliente”***

* No caso de não haver ocorrido erros - Termina a transacção “**\_erp.TerminaTransaccao()**”
* Devolve como resultado uma classe do tipo “***PrimaveraResultStruture()”*** nos seguintes cenários:
  + **Erro** **–** que representa o respectivo erro;
  + **Sucesso –** devolve a referência bancária do estudante dentro mascara #XXXXXXXXX# e A mensagem que representa se foi sucesso completo ou sucesso com avisos;

#### Service Layer

##### Public PrimaveraResultStruture actualizarCliente

|  |  |  |  |
| --- | --- | --- | --- |
| Nome do campo | Descrição | Tipo de Dados/ Tamanho | Valores Permitidos |
| tipoPlataforma(\*) | Tipo de Plataforma Instalada no Servidor | Integer | 0 - Executive  **1 - Profissional** |
| codEmpresa (\*) | Código da Empresa Primavera | String(10) |  |
| codUtilizador (\*) | Código do Utilizador Primavera | String(10) |  |
| Password (\*) | Password do Utilizador Primavera | String(10) |  |
| cliente (\*) | Código do Cliente | String(12) | Alfanuméricos |
| novoNrCliente | Novo código do cliente | String(12) | Alfanuméricos |
| nome (\*) | Nome completo do cliente | String(50) | No numbers |
| nomeFiscal | Nome fiscal do cliente | String(50) | No numbers |
| Desconto | Desconto do cliente | Double | Positive Numbers |
| facMorada (\*) | Morada do cliente na Factura | String(50) |  |
| facLocalidade | Localidade do cliente na Factura | String(20) |  |
| facNrTelefone | Número do Telefone do cliente na Factura | String(20) | (82) 1234567 |
| nuit | NUIT do cliente | String(20) |  |
| vendedor (\*) | Instituto do cliente | String(3) | ISU |
| Observação | Observações sobre o cliente | String(250) |  |
| CDU\_bolsa | Se o cliente tem bolsa | Boolean |  |
| CDU\_geraMulta | Se o cliente vai gerar multas | Boolean |  |
| CDU\_tipoIngresso | O tipo de Ingresso do cliente | String(100) |  |
| CDU\_codLic | O código da licenciatura do cliente | String(50) |  |
| CDU\_turma | O código da turma do cliente | String(50) |  |
| listaContactos | Contactos do cliente | List<Contacto> |  |

##### Classe Contacto

|  |  |  |  |
| --- | --- | --- | --- |
| Nome do campo | Descrição | Tipo de Dados/ Tamanho | Valores Permitidos |
| Código | Código do cliente | String(12) | Alfanuméricos |
| primeiroNome | Primeiro nome do cliente | String(12) |  |
| ultimoNome | Último nome do cliente | String (12) |  |
| nomeCompleto | Nome completo do cliente | String(12) |  |
| tipoContacto | Tipo de Contacto do cliente | String(12) | Aluno |
| emailPrincipal | Email ISUTC do cliente | String(100) |  |
| emailAlternativo | Email Alternativo do cliente | String(100) |  |
| nrTelefone | Número de telefone do cliente | String(20) |  |
| nrTelefoneAlternativo | Número de telefone alternativo do cliente | String(20) |  |
| Morada | Morada do cliente | String(50) |  |
| Localidade | Localidade do cliente | String(20) |  |

### Operação 3 – Criar Factura

Esta operação será responsável por gravar facturas para um determinado cliente (candidato, estudante ou instituto) através do método ***gravarFactura(<inputsListados>)****.*

O Fénix será responsável por gerir a lógica do uso durante as operações na candidatura, matrícula, nível dos artigos e as suas respectivas quantidades sem os preços que serão definidos no ERP pelo utilizador para cada tipo de terceiro (curso) no inicio de cada ano lectivo preferencialmente.

#### Funcionalidades do Método

1. **Inicia a Transacção “\_erp.iniciaTransaccao()”**
2. **Cria o Cabeçalho do Documento de Venda**

Este método privado “**PreencheCabecalho\_DocumentoVenda(…)**” é chamado para preencher o cabeçalho do documento de venda “**\_documentoVenda**”, com os seguintes:

* **dataDoc** será igual a data do sistema
* **dataGravacao** será igual a data do sistema
* **tipoEntindade** será igual á “C”
* **tipoDoc** para o caso da Factura – **FA**
* **Serie** será calculada no ERP a série por defeito para o respectivo documento de venda
* **CDU\_semestre** será vazio
* **CDU\_refBancaria** este campo irá ser preenchido pela referência do estudante

O resto dos campos necessários para preencher ao documento de venda factura será invocada partir do método do para preencher todos os dados relacionados “**\_erp.Comercial.Vendas. preencheDadosRelacionados(\_documentoVenda)**”.

- No caso de erro faz ***“roolback”*** da operação “**\_erp.DesfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

1. **Adiciona Linha ao Documento de Venda**

Este método privado “**adiciona\_Linhas\_DocumentoVenda(List<Artigo>lista\_artigos)**” será invocado somente depois de ter sido criado o cabeçalho com sucesso. O método irá fazer percorrer a lista dos artigos “**foreach( var artigoTemp in lista\_artigos)**” e seguira os seguintes acções:

- Adiciona a linha no documento de venda “**objmotor.Comercial.Vendas.AdicionaLinha( \_documentoVenda , artigoTemp .artigo, artigoTemp .quantidade,…)**”

- No caso de erro faz ***“roolback”*** da operação “**\_erp.DesfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

**NB:** O motor ERP irá preencher automaticamente o valor do preço do artigo definido no inicio do ano académico pelo utilizador no ERP.

1. **Grava o Documento de Venda**

Este método privado “**grava\_DocumentoVenda()**” será responsável por gravar o documento no ERP primavera a partir do método “**\_erp.Comercial.Vendas.Actualiza(\_documentoVenda, str\_Avisos)**”

* - No caso de erro faz ***“roolback”*** da operação “**\_erp.desfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

1. **Fim do método “GravarFactura**”

No caso de não haver ocorrido erros - Termina a transacção “**\_erp.TerminaTransaccao()**”

Devolve como resultado uma classe do tipo “***PrimaveraResultStruture***”nos seguintes cenários:

* **Erro** – que representa o respectivo erro;
* **Sucesso** – Código que representa o documento #**\_documentoVenda .tipodoc +” ”+ \_documentoVenda .numdoc + “/”+\_documentoVenda .Serie** # + a mensagem de sucesso completo ou o sucesso com avisos.

#### Service Layer

##### Método Gravar Factura

|  |  |  |  |
| --- | --- | --- | --- |
| Nome do campo | Descrição | Tipo de Dados/ Tamanho | Valores Permitidos |
| tipoPlataforma | Tipo de Plataforma Instalada no Servidor | Integer | 0- Executive  **1- Profissional** |
| codEmpresa | Código da Empresa Primavera | String(10) |  |
| codUtilizador | Código do Utilizador Primavera | String(10) |  |
| password | Password do Utilizador Primavera | String(10) |  |
| nomeUtilizador | Nome do Utilizador Fénix | String(100) | Alfanuméricos |
| cliente (\*) | Código do Cliente | String(12) | Alfanuméricos |
| tipoDoc(\*) | Tipo de documento (FA) | String (5) |  |
| listaArtigos | Lista de Artigos a Serem Facturados | List<Artigo> |  |

##### ClasseArtigo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome do campo | Descrição | Tipo de Dados/ Tamanho | Valores Permitidos | |
| artigo(\*) | Código do Artigo | String(12) | | Alfanuméricos |
| quantidade | Quantidade dos artigos | Double | |  |
| desconto | Desconto sobre o artigo | Double | |  |

### Operação 4 – Criar ou Actualizar Contrato

Esta operação será responsável por gravar contratos para um determinado cliente (estudante) através do método ***actualizaContrato(<inputsListados>)****.*

O Fénix será responsável por gerir a lógica do uso durante as operações de matrícula, renovação, nível dos artigos e as suas respectivas quantidades sem os preços que serão definidos no ERP para cada tipo de terceiro (curso).

#### Funcionalidade do Método

1. **Inicia a Transacção “\_erp.iniciaTransaccao()”**
2. **Cria o Cabeçalho do Documento de Venda**

Este método privado “**PreencheCabecalho\_DocumentoVenda(…)**” é chamado para preencher o cabeçalho do documento de venda “**\_documentoVenda**”, com os seguintes:

* **dataDoc** será igual a data do sistema
* **dataGravacao** será igual a data do sistema
* **tipoEntindade** será igual á “C”
* **tipoDoc** para o caso da Factura – **CTR**
* **Serie** será calculada no ERP a série por defeito para o respectivo documento de venda
* **CDU\_semestre** será vazio
* **CDU\_refBancaria** este campo irá ser preenchido pela referência do estudante

O resto dos campos necessários para preencher ao documento de venda factura será invocada partir do método do para preencher todos os dados relacionados “**\_erp.Comercial.Vendas. preencheDadosRelacionados(\_documentoVenda)**”.

1. **Adiciona Linha ao Documento de Venda**

Este método privado “**adiciona\_Linhas\_DocumentoVenda(List<Artigo>lista\_artigos)**” será invocado somente depois de ter sido criado o cabeçalho com sucesso. O método irá fazer percorrer a lista dos artigos “**foreach( var artigoTemp in lista\_artigos)**” e seguira os seguintes acções:

- Adiciona a linha no documento de venda “**objmotor.Comercial.Vendas.AdicionaLinha(documentoVenda , artigoTemp .artigo, artigoTemp .quantidade,…)**”

- No caso de erro faz ***“roolback”*** da operação “**\_erp.DesfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

**NB:** O motor ERP irá preencher automaticamente o valor do preço do artigo definido no inicio do ano académico pelo utilizador no ERP.

1. **Grava o Documento de Venda *(publicPrimaveraResultStructureGravar\_DocumentoVenda())***

Este método privado “**grava\_DocumentoVenda()**” será responsável por gravar o documento no ERP primavera a partir do método “**\_erp.Comercial.Vendas.Actualiza(\_documentoVenda, str\_Avisos)**”

- No caso de erro faz **“roolback”** da operação “**\_erp.desfazTransaccao()”** e devolve o respectivo erro “**primaveraStrutureResult**”.

1. **Grava a Avença com base no Documento de Venda**

Este método privado “**Grava\_DocumentoAvenca()**” irá gravar a avença “**\_documentoAvenca**” com base no documento de contrato “**\_documentoVenda**” e na informação do semestre para o início e fim da avença a partir do método “**\_erp.Comercial.Avencas.Actualiza(\_documentoAvenca)**”.

1. **Fim do método “*actualizaContrato”***

- No caso de não haver ocorrido erros - Termina a transacção “**\_erp.TerminaTransaccao()**”

- Devolve como resultado uma classe do tipo “***PrimaveraResultStruture***”nos seguintes cenários:

* **Erro** – que representa o respectivo erro;
* **Sucesso** – Código que representa o documento contrato #**\_documentoVenda .tipodoc +” ”+ \_documentoVenda .numdoc + “/”+\_documentoVenda .Serie** # + a mensagem de sucesso completo ou o sucesso com avisos

#### Service Layer

##### Gravar Contracto

|  |  |  |  |
| --- | --- | --- | --- |
| Nome do campo | Descrição | Tipo de Dados/ Tamanho | Valores Permitidos |
| tipoPlataforma | Tipo de Plataforma Instalada no Servidor | Integer | 0- Executive  **1- Profissional** |
| codEmpresa | Código da Empresa Primavera | String(10) |  |
| codUtilizador | Código do Utilizador Primavera | String (10) |  |
| password | Password do Utilizador Primavera | String (10) |  |
| nomeUtilizador | Nome do Utilizador Fénix | String (100) | Alfanuméricos |
| cliente (\*) | Código do Cliente | String(12) | Alfanuméricos |
| tipoDoc | Tipo de documento (CTR) | String (5) |  |
| semestreId | Id do semestre | String (10) |  |
| lista\_Artigos | Lista de Artigos a Serem Facturados | List<Artigo> |  |

##### ClasseArtigo

|  |  |  |  |
| --- | --- | --- | --- |
| Nome do campo | Descrição | Tipo de Dados/ Tamanho | Valores Permitidos |
| artigo(\*) | Código do Artigo | String(12) | Alfanuméricos |
| quantidade | Quantidade dos artigos | Double |  |
| desconto | Desconto sobre o artigo | Double |  |

### Operação 4 – Consultar Conta

Esta operação será responsável por consultar os pendentes do cliente que se encontram fora do prazo. *List<Pendentes>* ***consultaConta (string codigoCliente)****.*

#### Funcionalidade do Método

1. **Devolver a lista de Pendentes**

Este método publico **“consutaConta(string codigoCliente)”** irá devolver uma lista com pendentes do respectivo cliente dos documentos que se encontram fora do prazo.

Esses documentos pendentes podem estar entre Facturas, Adiantamentos e outros documentos financeiros que forem a ser usados.

#### Service Layer

##### Consulta Conta

|  |  |  |  |
| --- | --- | --- | --- |
| Nome do campo | Descrição | Tipo de Dados/ Tamanho | Valores Permitidos |
| tipoPlataforma | Tipo de Plataforma Instalada no Servidor | Integer | 0- Executive  **1- Profissional** |
| codEmpresa | Código da Empresa Primavera | String(10) |  |
| codUtilizador | Código do Utilizador Primavera | String (10) |  |
| password | Password do Utilizador Primavera | String (10) |  |
| cliente (\*) | Código do Cliente | String(12) | Alfanuméricos |

##### ClassePendente

|  |  |  |  |
| --- | --- | --- | --- |
| Nome do campo | Descrição | Tipo de Dados/ Tamanho | Valores Permitidos |
| cliente | Código do cliente | String(12) | Alfanuméricos |
| tipoDoc | Tipo de Documento | String(5) | FA, ADC |
| numDoc | Número do Documento | String(50) |  |
| serie | Série do Documento | String(10) |  |
| valorTotal | Valor Total do Documento | Double |  |
| valorPendente | Valor Pendente do Documento | Double |  |
| moeda | Moeda | String(10) |  |
| dataCriacao | Data de Criação do Documento | DateTime |  |
| dataVencimento | Data de Vencimento do Documento | DateTime |  |

### Possíveis erros - PrimaveraResultStructure

Os possíveis erros vão representados pela classe *”****PrimaveraResultStructure****”.* Esta Classe vai conter toda a estrutura de erro, avisos, rejeições e resposta dependendo da operação. Os atributos **“*estadoConta*”** e **“*refBancaria***” deverão ser preenchidos quando forem chamadas as operações **“consultaConta*()”*** e “***criarAluno()”*** respectivamente.

A descrição dos erros, o tipo de problema, rejeições, avisos assim como os níveis, podem se encontrar na tabela abaixo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo de Problema | Código | Código do Nível – Descrição | Código/Subnível – Descrição | Procedimento após o erro |
| Códigos de Sucesso | **0** | **00** – Sucesso completo | **000** – Gravado com sucesso | **-** |
| **01** – Sucesso com aviso | **010** – Aluno gravado, mas existe outro com nome idêntico | **-** |
| Erro de Marshalling de dados | **1** | **10** – Validação dos campos | **100** - Primavera encontrou um campo com nome inválido | Insere um nome válido sem números e caracteres especiais |
| **101** - Primavera encontrou um campo com data de registo impossível (ex. data no futuro) | Verificar a data inserida no campo |
| **102 –** Primavera encontrou um campo com nr° de telefone inválido | Insere correctamente o nr° de telefone |
| **103**– Primavera encontrou um campo com Email inválido | Inserir correctamente o endereço de mail |
| **104** – Nr° de caracteres maior que “X” | Verificar a quantidade de caracteres inseridos no campo “Y” |
| Erro de sistema no Primavera | **2** | **20** – Erros com a DB | **200** - Primavera não consegue conectar a Base de Dados | Consultar os técnicos do projecto |
| **201** – Primavera não consegue conectar a Tabela “X” | Consultar os técnicos do projecto |
| **21** – Erros com ficheiros DLL | **210** - Primavera não consegue carregar um DLL | Consultar os técnicos do projecto |
| **22** – Outros erros | **220** - Erro não especificado | Consultar os técnicos do projecto |
| Erro lógico no Primavera | **3** | **30** – Erros de inscrição | **300** - Aluno já existe com o mesmo código | Consultar os técnicos do projecto |
| **301** – Já existe um aluno com o mesmo nr° de BI | Verificar bem os dados, pode ser que esteja a introduzir um aluno existente |
| **302** – O tipo de Ingresso não foi especificado | Especificar o tipo de ingresso |
| **31** – Erros na definição de *itens* | **310** - *item* referenciado, tem PVP 0 |  |
| Erros de rede | **4** | **40 –** Erros comuns | **400 –** *Time Out* | Verificar a conectividade com a rede |

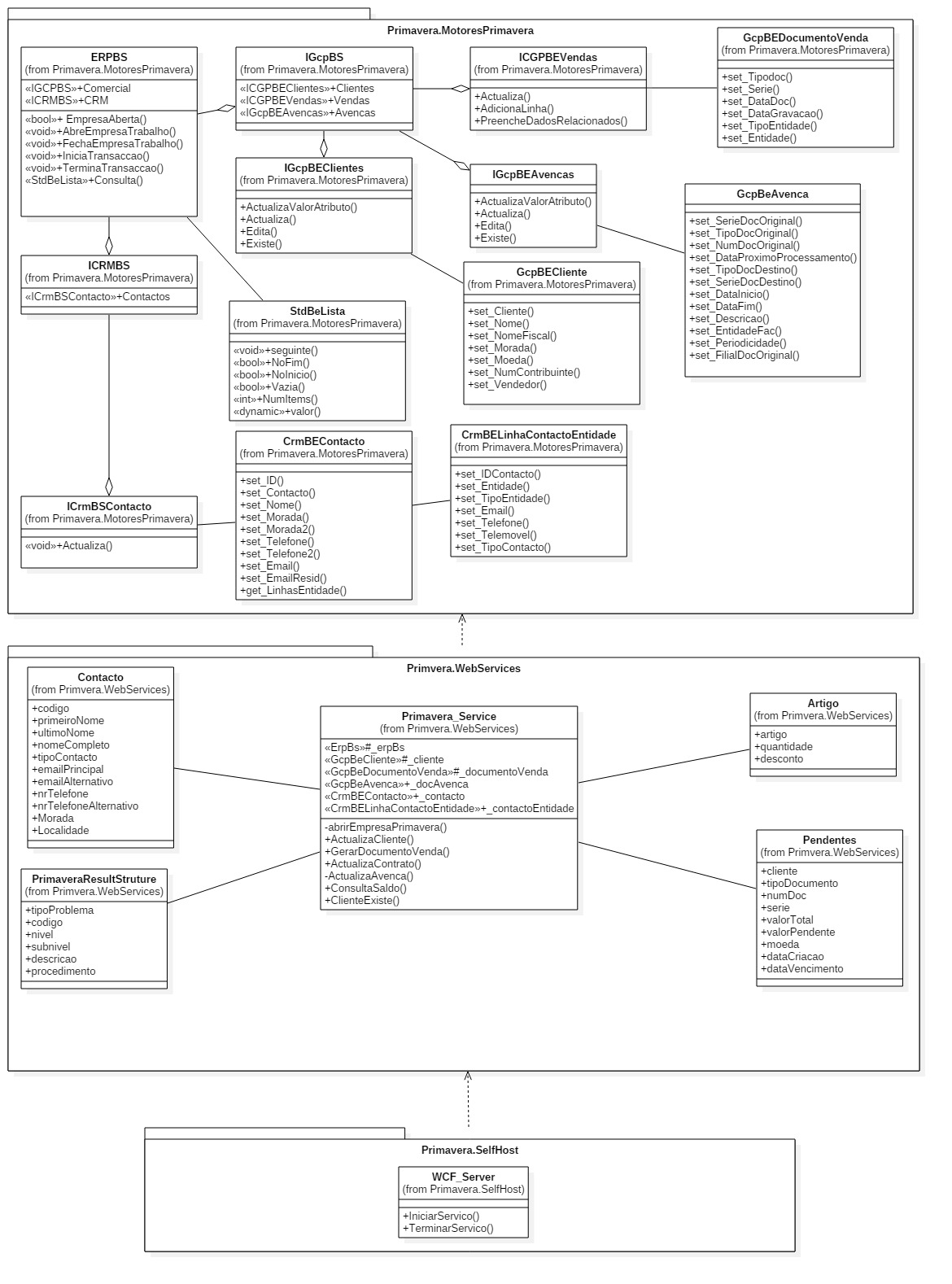
**Definição de Rejeições e Avisos pela validação do Primavera no *PrimaveraResultStructure***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo de Problema | Código | Código do Nível – Descrição | Código/Subnível – Descrição | Procedimento após Rejeição |
| Rejeições pela validação do Primavera | **5** | **50 –** Rejeições Comuns | **500** – Descrição da rejeição. | - Analisar e agir segundo a descrição da rejeição. |
| Avisos pela Validação do Primavera | **6** | **60 –** Avisos Comuns | **600** – Descrição do Aviso. | - Analisar e agir segundo a descrição do aviso. |

A classe **PrimaveraResultStructure** será definida como mostra a tabela abaixo:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Classe | Atributos | | | Métodos |
| PrimaveraResultStructure | **Nome dos campo/atributos** | **Tipo de Dados/ Tamanho** | **Valores Permitidos** | Um método **toString()** que vai retornar todos o valores dos atributos desta classe, assim como a descrição do problema. |
| codigo (\*) | Integer (50) |  |
| descricao (\*) | String (50) |  |
| subnivel (\*) | String (50) |  |
| tipoProblema (\*) | String (50) |  |

# Diagrama de pacotes e de classes

****

# ERP Primavera

## Campos de Utilizador

Campos adicionados as tabelas do sistema

|  |  |  |  |
| --- | --- | --- | --- |
| Campo | Tabela | Tipo | Descrição |
| CDU\_GeraMulta | Clientes | bit | Campo que irá definir se o aluno/cliente esta sujeita a multas |
| CDU\_Bolsa | Clientes | bit | Campo que irá definir se o aluno/cliente é bolseiro ou não |
| CDU\_CodlLic | Clientes | varchar(50) | Campo que irá conter o código da licenciatura do aluno |
| CDU\_Turma | Clientes | varchar(50) | Campo que irá conter a turma do aluno |
| CDU\_REFBNC | Clientes | nvarchar(11) | Campo que irá conter o código da referência bancaria do aluno |
| CDU\_TipoIngresso | Clientes | nvarchar(110) | Campo que irá conter o tipo de ingresso do aluno |
| CDU\_ANO | Clientes | nvarchar(10) | Campo que irá conter o ano do aluno |
| CDU\_Semestre | Clientes | nvarchar(10) | Campo que ira conter o semestre do aluno |
| CDU\_NumeroDisciplinas | Clientes | Integer | Campo que irá conter o número de disciplinas do Aluno |
| CDU\_NOME\_UTILIZADOR | CabecDoc | nvarchar(150) | Campo que irá conter o nome do utilizador que emitiu a factura |

## Tabelas do Utilizador

Tabelas adicionadas as tabelas por defeito do sistema.

### TDU\_Semestre

|  |  |  |  |
| --- | --- | --- | --- |
| Campo | Tipo | Descrição | Visível |
| CDU\_IDSemestre | nvarchar(50) | Campo que irá conter o id do semestre | Yes |
| CDU\_Codigo | nvarchar(10) | Campo que irá conter a descrição do Semestre | Yes |
| CDU\_AnoLectivo | nvarchar(50) | Campo que irá conter o ano lectivo | Yes |
| CDU\_DataInicio | Smalldatetime | Campo que irá a data do inicio do semestre | Yes |
| CDU\_DataPrimeiraPropina | Smalldatetime | Campo que irá conter a data da primeira propina | Yes |
| CDU\_DataFim | Smalldatetime | Campo que irá conter a data de fim do semestre | Yes |
| CDU\_DataUltimaPropina | Smalldatetime | Campo que irá conter a data da ultima propina | Yes |
| CDU\_Activo | bit | Campo que ira se o semestre se encontra activo ou não | Yes |

### TDU\_ParametrosISUTC

|  |  |  |  |
| --- | --- | --- | --- |
| Campo | Tipo | Descrição |  |
| Chave | Varchar(1) | Campo que servir de chave primária da tabela | No |
| Instituto | Varchar(10) | Campo que irá armazenar o código do instituto atribuído pela entidade bancária. | Yes |
| DocumentoLiquidacao | Varchar(5) | Campo que irá armazenar o código do documento de liquidação, na importação do ficheiro do banco. | Yes |
| DocumentoAdiantamento | Varchar(5) | Campo que irá armazenar o código do documento de adiantamento, no caso das liquições serem superiores aos valores pendentes, no processo de importação do ficheiro do banco. | Yes |
| DocumentoTesouraria | Varchar(5) | Campo que irá armazenar o código do documento de Tesouraria, onde serão lançadas as comissões (taxas), no processo de importação do ficheiro do banco. | Yes |
| ContaBancaria | Varchar(5) | Campo que irá armazenar a conta bancária de onde serão debitadas as comissões (taxas), no processo de importação do ficheiro do banco. | Yes |
| MovimentoBancario | Varchar(5) | Campo que irá armazenar a rubrica bancária onde serão debitadas as comissões (taxas), no processo de importação do ficheiro do banco. | Yes |
| DocumentoMulta | Varchar(5) | Campo que irá armazenar o código da nota de débito com a multa gerada por atraso no pagamento. | Yes |
| DiasParaMulta | Int | Campo que irá armazenar o número de dias que após a data de vencimento serão geradas multas. | Yes |
| Percentagem | Decimal(10,2) | Percentagem do valor Total que será cobrado de multa pelo atraso. | Yes |
| EmailResponsavelVD | Varchar(100) | Email do Responsável das Vendas. | Yes |
| EmailResponsavelLQ | Varchar(100) | Email do Responsável das Liquidações. | Yes |
| FolderFilesPlatao | Varchar(500) | Pasta onde devem ser descarregadados os ficheiros a importar do Platão. | Yes |
| FolderFilesBanco | Varchar(500) | Pasta onde devem ser descarregadados os ficheiros a importar do Banco. | Yes |

# Explicação dos Conceitos

## Webservice

**Webservice** é uma solução utilizada na integração de sistemas e na comunicação entre [aplicações](https://pt.wikipedia.org/wiki/Aplica%C3%A7%C3%A3o) diferentes. Com esta [tecnologia](https://pt.wikipedia.org/wiki/Tecnologia) é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em [plataformas](https://pt.wikipedia.org/wiki/Plataforma) diferentes sejam compatíveis. Os *Webservices* são componentes que permitem às aplicações enviar e receber dados em formato ***“***[***XML***](https://pt.wikipedia.org/wiki/XML)***”.*** Cada aplicação pode ter a sua própria linguagem, que é traduzida para uma linguagem universal, um formato intermediário como *XML, Json, CSV, etc*.

Para as empresas, os *Webservices* podem trazer agilidade para os processos e eficiência na comunicação entre cadeias de produção ou de [logística](https://pt.wikipedia.org/wiki/Log%C3%ADstica). Toda e qualquer comunicação entre sistemas passam a ser dinâmica e principalmente segura, pois não há intervenção humana.

Essencialmente, o Webservice faz com que os recursos da aplicação do software estejam disponíveis sobre a rede de uma forma normalizada. Outras tecnologias fazem a mesma coisa, como por exemplo, os browsers da Internet acessam às páginas Web disponíveis usando por norma as tecnologias da Internet, HTTP e HTML. No entanto, estas tecnologias não são bem sucedidas na comunicação e integração de aplicações. Existe uma grande motivação sobre a tecnologia Webservice pois possibilita que diferentes aplicações comuniquem-se entre si e utilizem recursos diferentes.

Utilizando a tecnologia Webservice, uma aplicação pode invocar outra para efectuar tarefas simples ou complexas mesmo que as duas aplicações estejam em diferentes sistemas e escritas em linguagens diferentes. Por outras palavras, os Webservices fazem com que os seus recursos estejam disponíveis para que qualquer aplicação cliente possa operar e extrair os recursos fornecidos pelo Webservice.

Os Webservices são identificados por um URI (***Uniform Resource Identifier***), descritos e definidos usando XML (***Extensible Markup Language***). Um dos motivos que tornam os Webservices atractivos é o facto deste modelo ser baseado em tecnologias standards, em particular XML e HTTP (***Hypertext Transfer Protocol***). Os Webservices são utilizados para disponibilizar serviços interactivos na Web, podendo ser acessados por outras aplicações usando, por exemplo, o protocolo [SOAP](https://pt.wikipedia.org/wiki/SOAP) (***Simple Object Access Protocol***).

O **objectivo dos Web Services** é a comunicação de aplicações através da Internet. Esta comunicação é realizada com intuito de facilitar a [EAI](https://pt.wikipedia.org/wiki/EAI) (***Enterprise Application Integration***) que significa a integração das aplicações de uma empresa, ou seja, interoperabilidade entre a informação que circula numa organização nas diferentes aplicações como, por exemplo, o comércio electrónico com os seus clientes e seus fornecedores. Esta interacção constitui o sistema de informação de uma empresa. E para além da interoperabilidade entre as aplicações, a EAI permite definir um “***workflow***” entre as aplicações e pode constituir uma alternativa aos [ERP](https://pt.wikipedia.org/wiki/ERP) (***Enterprise Resource Planning***). Com um “***[workflow](https://pt.wikipedia.org/wiki/Workflow" \o "Workflow)***” é possível optimizar e controlar processos e tarefas de uma determinada organização.

### WCF – Windows Communication Foundation

O **WCF** é uma parte da ***“.NET”*** Framework que fornece um modelo unificado de programação para construir de forma rápida aplicações distribuídas orientadas a serviço (**SOA** – ***Service Oriented Architecture***), permite enviar mensagens assíncronas de um serviço a outro, as vantagens que o WCF oferece são:

* Pode ser parte de um serviço hospedado no ISS;
* Pode ser um serviço hospedado em uma aplicação;
* Pode ser um cliente de um outro serviço que busca dados de outro serviço;
* As mensagens podem ser simples caracteres ou palavras mandadas em XML ou dados complexos, como dados binários.

Em cenários comuns para hospedar Serviços WCF é usado o IIS, WAS, Self-Hosted, Managed Windows Service.

### Self-Hosting

Define-se como a capacidade que uma aplicação tem de desenvolver rotinas compiladas ou interpretadas por si mesma. Este processo pode ser comparado a uma auto-compilação ou auto-suficiência de recursos.

### Self-Hosting Windows Communication Foundation Services (WCF)

Para chamar um serviço WCF, deve-se hospedar o serviço em um ambiente de tempo de execução. O papel do aplicativo “***host”*** é:

* Iniciar e parar o serviço;
* Escutar as solicitações de clientes;
* Direccionar essas solicitações para o serviço e enviar respostas de volta para os clientes.

### Serviço ISS - Internet information service

“***Internet Information Services***” é um servidor Web extensível criado pela Microsoft para uso com a família Windows NT. IIS suporta HTTP, HTTPS, FTP, FTPS, SMTP e NNTP, com essas características é ideal para participar no processo de comunicação entre softwares diferentes com a possibilidade de partilhar recursos a nível local ou internet.

### SOAP (Simple Object Access Protocol)

É uma especificação de protocolo para troca de informações estruturadas na implementação de serviços Web em redes de computadores. Ele usa “***XML Information Set***” para seu formato de mensagem, e se baseia em outros protocolos da camada de aplicação, mais notavelmente “***Hypertext Transfer Protocol***” (**HTTP**) ou “***Simple Mail Transfer Protocol***” (**SMTP**), para a negociação de mensagem e transmissão.

Este protocolo tem principais características:

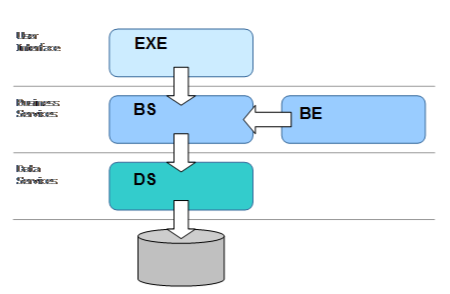
* Independência - permite qualquer modelo de programação e plataforma;
* Pode operar sobre qualquer protocolo de transporte como HTTP, SMTP, TCP, UDP
* Serve para comunicação entre aplicações;
* Baseado em XML;
* Simples e extensível.

## Tecnologia de Integração PRIMAVERA

### Motores das Aplicações

Conceptualmente, a arquitectura do motor de cada aplicação PRIMAVERA, estruturada em três camadas, isola o acesso aos dados, potenciando a sua reutilização por aplicações externas sem colocar em causa a integridade dos dados.

O motor de uma aplicação é composto por um conjunto de DLLs COM que podem ser referenciadas e utilizadas a partir de aplicações externas, bastando para tal que estejam registadas no sistema e que sejam incluídas no respectivo projecto.



O diagrama anterior apresenta a arquitectura e organização do motor de uma aplicação que será composto pelas seguintes DLLs:

|  |  |
| --- | --- |
| xxxBE800.dll | Objectos de negócio.  Representam as entidades geridas pelo motor e pela aplicação. |
| xxxBS800.dll | **Serviços de Negócio.**  Disponibilizam todas as regras de negócio. |
| xxxDS800.dll | **Serviços de Dados.**  Disponibilizam todos os serviços de carregamento e gravação de dados a partir da base de dados. |

### Tecnologia COM e Arquitectura Windows DNA

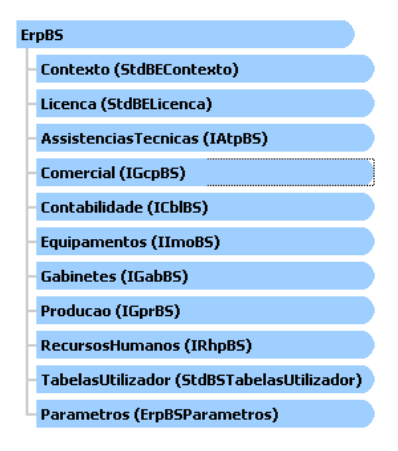
* Todas as aplicações PRIMAVERA desenvolvidas com base nesta tecnologia
* Permite disponibilizar o acesso à camada intermédia de uma forma que não permite subverter a lógica inerente ao funcionamento das aplicações
* Garante o cumprimento das regras de negócio estabelecidas no ERP
* Independência do modelo de dados
* Partilha de objectos entre aplicações
* Utilização dos objectos em aplicações externas, ex: Microsoft Excel ou aplicação desenvolvida pelo utilizador.

### Objectos de Negócio (BE)

* Objectos de negócio (ou entidades) geridos por cada aplicação.
* Nomenclatura: <Apl>BE<nome> ,exemplo: GcpBEDocumentoVenda, GcpBEClientes, GcpBEAvenças, GcpBEPendentes, CrmBEContactos.
* Cada objecto terá um conjunto de propriedades.

### Serviços de Negócios (BS)

* Disponibiliza todos os serviços .
* Implementa todas as regras de negócio.
* Nomenclatura: <Apl>BS<nome> , exemplo: GCPBSClientes, CrmBsContactos
* Ponto de entrada no motor do ERP – Classe ErpBS



### Serviços de Dados (DS)

* Disponibiliza o serviço de acesso a dados
* Nunca deve ser referenciado a partir de aplicações externas
* Os serviços devem ser sempre executados a partir da camada de negócio (BS)

# Validação do Documento

|  |  |
| --- | --- |
| **Responsável do DSE do ISUTC** |  |
| **Responsável da DSI do ISUTC** |  |
| **Gestor de projecto Accsys / Director Accsys** |  |