**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

Engenharia Informática e de Computadores

**Projecto e Seminário**

**Relatório do Projecto**

**Rapid Application Development**

**EDM Solution**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Membros do Grupo** | | | | | |
| **26657** | **Ricardo Neto** | **31923** | **Nuno Sousa** | **32223** | **Paulo Pires** |
| *26657@alunos.isel.pt* | | *31923@alunos.isel.pt* | | *32223@alunos.isel.pt* | |
| *Tlm. 91.420.02.53* | | *Tlm. 96.205.04.70* | | *Tlm. 92.757.77.35* | |

|  |  |
| --- | --- |
| **Orientador de Projecto** | |
| **Eng. Fernando Miguel Carvalho** | **mcarvalho@cc.isel.ipl.pt** |

*- Introdução (Descrição do problema e motivação para a solução apresentada)*

*- Exemplo de Utilização*

*- Descrever um exemplo e modela-lo no dicionário de dados*

*- Criar uma solução EDM no Visual Studio e promover a geração de código*

*- Utilizar o script SQL para criar o modelo físico*

*- Registar o webservice com os processos de negócio e utiliza-lo*

*- Criar um entry point e utilizar os objectos de dominio numa aplicação de consola*

*- Arquitectura da Solução*

*- Dicionário de Dados*

*- Framework EDM (Generator, FoundationClasses)*

*- Solução EDM (template VS)*

Índice

[Índice de Figuras 6](#_Toc272218812)

[Índice de Listagens 6](#_Toc272218813)

[Índice de Tabelas 6](#_Toc272218814)

[1 Introdução *(retirado da proposta de projecto)* 7](#_Toc272218815)

[2 Exemplo de Utilização 8](#_Toc272218816)

[2.1 Análise do Problema 8](#_Toc272218817)

[2.2 Criação de Solução no Visual Studio 9](#_Toc272218818)

[2.3 Tipos do Domínio 10](#_Toc272218819)

[2.4 Entidades do Domínio 12](#_Toc272218820)

[2.5 Persistência em Base de Dados 13](#_Toc272218821)

[2.6 Processos de negócio 14](#_Toc272218822)

[2.7 Geração automática de código 15](#_Toc272218823)

[2.8 Geração automática do script de construção do modelo de dados 16](#_Toc272218824)

[3 Arquitectura da Solução 17](#_Toc272218825)

[4 Dicionário de Dados 18](#_Toc272218826)

[*4.1* Modelo de Tipos – elemento *<usertypes>* 18](#_Toc272218827)

[*4.1.1* Restrições 19](#_Toc272218828)

[4.1.2 Definição das Restrições 19](#_Toc272218829)

[4.1.2.1 length 19](#_Toc272218830)

[4.1.2.2 minLength 19](#_Toc272218831)

[4.1.2.3 maxLength 20](#_Toc272218832)

[4.1.2.4 pattern 20](#_Toc272218833)

[4.1.2.5 enumeration 20](#_Toc272218834)

[4.1.2.6 maxInclusive 20](#_Toc272218835)

[4.1.2.7 maxExclusive 20](#_Toc272218836)

[4.1.2.8 minInclusive 20](#_Toc272218837)

[4.1.2.9 minExclusive 20](#_Toc272218838)

[4.1.2.10 totalDigits 21](#_Toc272218839)

[4.1.2.11 fractionDigits 21](#_Toc272218840)

[*4.2* Entidades – elemento *<entities>* 21](#_Toc272218841)

[*4.2.1* Definição de uma entidade – Identificação 21](#_Toc272218842)

[*4.2.2* Definição de uma entidade – Herança 21](#_Toc272218843)

[*4.2.3* Atributos de uma entidade – elemento *<fields>* 22](#_Toc272218844)

[*4.2.4* Relações entre entidades – elemento *<relations>* 23](#_Toc272218845)

[*4.2.5* Relações 1-1 – elemento *<oneToOne>* 23](#_Toc272218846)

[*4.2.6* Relações 1-n – elemento *<oneToMany>* 23](#_Toc272218847)

[*4.2.7* Relações n-n – elemento *<ManyToMany>* 24](#_Toc272218848)

[*4.3* Serviços externos – elemento *<environments>* 24](#_Toc272218849)

[*4.3.1* Serviços de Base de Dados – elemento *<dataEnvironments>* 24](#_Toc272218850)

[*4.4* Processos de Negócio – elemento *<businessProcesses>* 25](#_Toc272218851)

[*4.4.1* Processo de Negócio – elemento *<businessProcess>* 26](#_Toc272218852)

[*4.4.1.1* Parâmetros – elemento *<input>* e *<param>* 26](#_Toc272218853)

[*4.4.1.2* Retorno – elemento *<output>* 26](#_Toc272218854)

[5 Estrutura de uma Solução EDM no Visual Studio 28](#_Toc272218855)

[5.1 Metadados 28](#_Toc272218856)

[5.2 Biblioteca 28](#_Toc272218857)

[5.3 Projectos 28](#_Toc272218858)

[5.3.1 Projecto Rtti 29](#_Toc272218859)

[5.3.2 Projecto Entity 29](#_Toc272218860)

[5.3.3 Projecto Services 30](#_Toc272218861)

[5.3.4 Projecto Ws 31](#_Toc272218862)

[5.3.5 Projecto UnitTest 32](#_Toc272218863)

[6 Gerador de Código *(EDM.Generator)* 33](#_Toc272218864)

[6.1 Contexto de Geração *(EDM.Generator.Context)* 33](#_Toc272218865)

[6.2 Motor de Geração *(EDM.Generator.Engine)* 33](#_Toc272218866)

[6.2.1 Pipeline de Transformação (*GeneratorSteps)* 34](#_Toc272218867)

[6.2.2 Etapa *‘ThreeDPreExecuteStep’* 35](#_Toc272218868)

[6.2.2.1 Verificação de utilização de palavras reservadas 35](#_Toc272218869)

[6.2.2.2 Alteração de elementos <*field>* 35](#_Toc272218870)

[6.2.2.3 Alteração do elemento <*solution*> 36](#_Toc272218871)

[6.2.2.4 Alteração do elemento <*userTypes*> 36](#_Toc272218872)

[6.2.2.5 Alteração do elemento <*entities*> e elementos *<entity>* 36](#_Toc272218873)

[6.2.2.6 Alterações ao elemento *<relations>* 36](#_Toc272218874)

[6.2.2.7 Alterações ao *<businessProcesses>* 36](#_Toc272218875)

[6.2.3 Etapa *‘Step’* 37](#_Toc272218876)

[6.2.3.1 Transformações direccionadas ao projecto de tipos 37](#_Toc272218877)

[6.2.3.2 Transformações direccionadas ao projecto de entidades 37](#_Toc272218878)

[6.2.3.2.1 Campos de Entidades 38](#_Toc272218879)

[Trabalhos Citados 40](#_Toc272218880)

## Índice de Figuras

[Figura 1 – Fases do desenvolvimento de software 7](#_Toc272218881)

[Figura 2 – Modelo Entidade Associação 8](#_Toc272218882)

[Figura 3 - Criação de solução EDM no Visual Studio 9](#_Toc272218883)

[Figura 4 - Formulário de identificação da solução EDM 10](#_Toc272218884)

[Figura 5 - Estrutura de um template EDM no Visual Studio 10](#_Toc272218885)

[Figura 6 - Estrutura de uma Solução EDM 28](#_Toc272218886)

[Figura 7 - Estrutura do projecto 'Rtti' 29](#_Toc272218887)

[Figura 8 - Estrutura do projecto 'Entity' 30](#_Toc272218888)

[Figura 9 - Estrutura do projecto 'Services' 31](#_Toc272218889)

[Figura 10 - Estrutura do projecto 'Ws' 31](#_Toc272218890)

[Figura 11 - Estrutura do projecto 'UnitTest' 32](#_Toc272218891)

[Figura 12 - Diagrama UML de classes do contexto de geração 33](#_Toc272218892)

[Figura 13 - Diagrama UML de classes do motor de geração 34](#_Toc272218893)

[Figura 14 - Diagrama UML de classes da geração por etapas 35](#_Toc272218894)

## Índice de Listagens

[Listagem 1 – Definição dos tipos do dominio 11](#_Toc272218895)

[Listagem 2 - Definição da entidade 'Editor' 12](#_Toc272218896)

[Listagem 3 - Definição da entidade 'Album' 12](#_Toc272218897)

[Listagem 4 - Definição da entidade 'LP' 13](#_Toc272218898)

[Listagem 5 - Definição da entidade 'EP' 13](#_Toc272218899)

[Listagem 7 - Definição das entidades ‘Interprete’ e ‘Faixa’ 13](#_Toc272218900)

[Listagem 8 - Parametrização de acesso a servidor de BD 14](#_Toc272218901)

[Listagem 9 - Definição de Processos de Negócio 15](#_Toc272218902)

[Listagem 10 - Schema do elemento field 22](#_Toc272218903)

[Listagem 11 - Schema do elemento oneToOne 23](#_Toc272218904)

[Listagem 12 - Schema do elemento oneToMany 24](#_Toc272218905)

[Listagem 13 - Schema do elemento manyToMany 24](#_Toc272218906)

[Listagem 14 - Declaração de processos de negócio 25](#_Toc272218907)

## Índice de Tabelas

[Tabela 1 - Tipos disponibilizados e respectivas correspondências 18](#_Toc272218908)

[Tabela 2 - Restrições aplicáveis a tipos 19](#_Toc272218909)

[Tabela 3 – Relações de herança possíveis no elemento entity 22](#_Toc272218910)

[Tabela 4 - Atributos do elemento field 23](#_Toc272218911)

[Tabela 5 - Atributos do elemento provider de dataEnvironments 25](#_Toc272218912)

[Tabela 6 - Atributos do elemento businessProcess 26](#_Toc272218913)

[Tabela 7 - Atributos do elemento param 26](#_Toc272218914)

[Tabela 8 - Atributos do elemento output 27](#_Toc272218915)

# Introdução *(retirado da proposta de projecto)*

Com o aparecimento de software cada vez mais barato, torna-se imperativo para as *software houses* reduzirem os custos de desenvolvimento de produtos, no sentido de conseguirem operar num mercado global cada vez mais competitivo.

O meio para chegar a esse fim divide-se em diminuir o nível de remuneração dos seus recursos humanos ou diminuir o ciclo de desenvolvimento do produto.

Com o presente projecto pretende-se criar uma solução que permita o desenvolvimento de aplicações de forma mais rápida, diminuindo assim todo o ciclo de desenvolvimento incluído.

Considerando o ciclo de desenvolvimento de uma aplicação destacam-se as seguintes fases:

Figura – Fases do desenvolvimento de software

No caminho para a fase de desenvolvimento, são produzidos *outputs* suficientes para se conseguir descrever um problema com algum detalhe.

Pretende-se que, ao estruturar essa informação num formato específico, seja possível codificar estruturas sobre as quais a equipa de desenvolvimento iniciará o seu trabalho.

# Exemplo de Utilização

No sentido de se melhor compreender como se deve utilizar o presente projecto, bem como, quais os resultados esperados, optamos por descrever neste capítulo todos os passos envolvidos, recorrendo a um exemplo.

# Análise do Problema

A discoteca MusicBit pretende a criação de uma aplicação que permita a gestão dos artigos vendidos por si.

No sentido de melhor perceber quais as necessidades do Cliente, foram feitas várias reuniões que permitiram definir os seguintes pontos:

- entidades que deverão estar representadas na aplicação

- tipos de dados utilizados

- operações a serem expostas pela aplicação

Na sequência da análise das entidades envolvidas na aplicação, chegou-se ao modelo entidade associação da figura em baixo.

Figura – Modelo Entidade Associação

Foi também definido pela MusicBit que a aplicação deve permitir que os seus Clientes possam fazer encomendas de produtos, obter o estado das suas encomendas e cancela-las. Para tal, estabeleceu-se que será também necessário suportar o registo de Clientes na aplicação, protegendo o acesso à mesma com uma *password* que pode ser alterada ou mesmo recuperada.

Estabelecidos os requisitos da aplicação, inicia-se agora a fase de criação do dicionário de dados em formato XML onde é obrigatório que a sua especificação (ver pág. XXX) seja respeitada, a fim de garantir a correcta interpretação por parte do gerador de código.

# Criação de Solução no Visual Studio

O primeiro passo para o desenvolvimento de uma solução EDM é a criação da sua estrutura base no Visual Studio. Assim, após ter sido feita a instalação da ‘Guidance Package EDM.Template’, conforme descrito na Página XXX, é possível criar uma solução EDM no Visual Studio, seleccionando o tipo de projecto mostrado na figura em baixo.

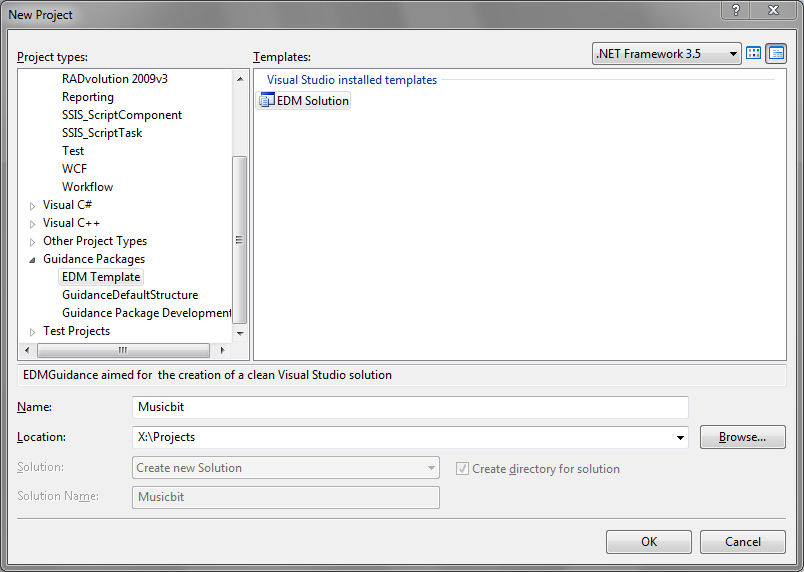


Figura - Criação de solução EDM no Visual Studio

Ao ser seleccionada a criação de uma solução com base no modelo EDM.Template é apresentado o formulário mostrado na figura em baixo, com vista à obtenção de informação relativa à identificação da aplicação, nomeadamente, o nome do projecto e da empresa Cliente.

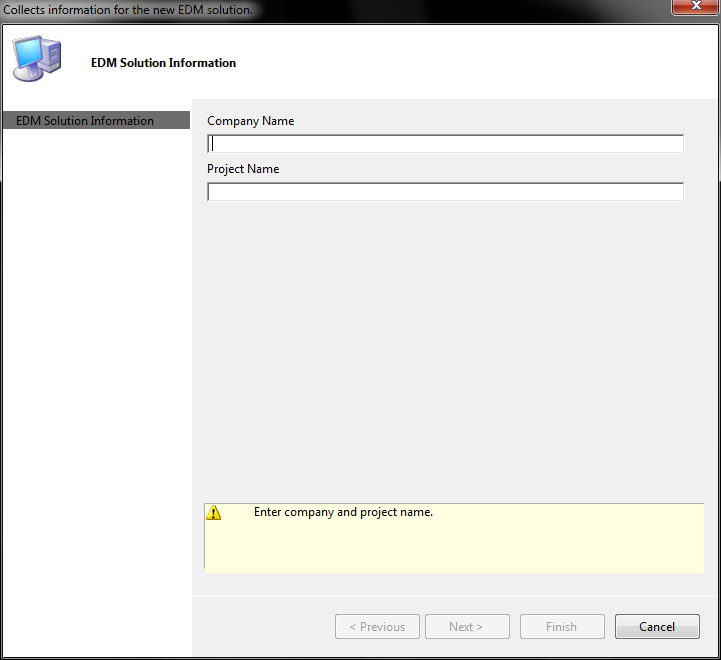


Figura - Formulário de identificação da solução EDM

A partir deste momento é automaticamente criada toda a estrutura da solução EDM, conforme figura em baixo, que servirá de base à aplicação a desenvolver.

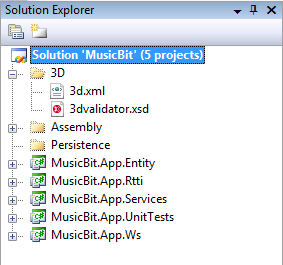


Figura - Estrutura de um template EDM no Visual Studio

Após a criação da solução importa agora codificar o resultado da análise do dicionário de dados.

# Tipos do Domínio

Durante a fase de análise foram determinados os tipos de dados a utilizar e, relativamente a estes, foram estabelecidas regras como é o caso de valores mínimos e máximos, intervalos de valores possíveis, dimensões permitidas, etc. Estas definições vão ser codificadas no elemento *userTypes* que por sua vez irá conter elementos que representam tipos concretos das linguagens de programação, como é o caso de *int* e *string*.

|  |
| --- |
| <userTypes generatedFileName="userTypes"> |
| <int name="identificador" maxInclusive="5000"/> |
| <string name="nomeArtista"/> |
| <string name="pais" maxLength="3" pattern="[A-Z]{3}"/> |
| <string name="tituloAlbum" maxLength="100"/> |
| <DateTime name="dataEdicaoLP"/> |
| <string name="generoMusical"> |
| <enumeration>Rock</enumeration> |
| <enumeration>Pop</enumeration> |
| <enumeration>Reggae</enumeration> |
| <enumeration>Blues</enumeration> |
| <enumeration>Jazz</enumeration> |
| <enumeration>Classica</enumeration> |
| </string> |
| <string name="nomeMusica"/> |
| <string name="tempoMusica" pattern="[0-9]{2}:[0-9]{2}"/> |
| <int name="anoColectanea"/> |
| <string name="patrocinadorColectanea"/> |
| <string name="nomeEditor" maxLength="50"/> |
| <string name="tipoAlbum"> |
| <enumeration>LP</enumeration> |
| <enumeration>EP</enumeration> |
| <enumeration>Colectanea</enumeration> |
| </string> |
| <int name="encomendaQtd" minInclusive="1" maxInclusive="100"></int> |
| <string name="canalVendas"> |
| <enumeration>LOJA</enumeration> |
| <enumeration>WEB</enumeration> |
| </string> |
| <int name="idCliente" minExclusive="0" maxInclusive="9999999"/> |
| <string name="idEncomenda" length="20"></string> |
| <string name="estadoEncomenda"> |
| <enumeration>Registada</enumeration> |
| <enumeration>Tramitada</enumeration> |
| <enumeration>Esgotada</enumeration> |
| <enumeration>Enviada</enumeration> |
| <enumeration>Anulada</enumeration> |
| <enumeration>Cancelada</enumeration> |
| <enumeration>Concluida</enumeration> |
| </string> |
| <string name="retornoCancelarEncomenda"> |
| <enumeration>Cancelada</enumeration> |
| <enumeration>EstadoInvalido</enumeration> |
| <enumeration>PendenteCancelamento</enumeration> |
| </string> |
| <string name="userName"></string> |
| <string name="password"></string> |
| <string name="nomeCompleto"></string> |
| <DateTime name="dtNascimento"></DateTime> |
| <string name="retornoAlteracaoPassword"> |
| <enumeration>Sucesso</enumeration> |
| <enumeration>UtilizadorInvalido</enumeration> |
| <enumeration>UtilizadorBloqueado</enumeration> |
| </string> |
| </userTypes> |

Listagem – Definição dos tipos do dominio

Conforme podemos ver na listagem em cima, os nomes dos tipos a serem utilizados na aplicação são definidos como atributos do elemento que define o seu tipo base. Para cada um deles é possível definir um conjunto de restrições, conforme descrito na especificação do dicionário de dados (Página 19).

# Entidades do Domínio

A fim de possibilitar a criação de objectos de domínio que estejam organizados de acordo com o modelo entidade associação da Figura 2, definem-se as entidades no elemento *entities*. Cada entidade será representada por um elemento *entity*.

|  |
| --- |
| <entities> |
| <entity type="base" name="Editor" generatedFileName="Editor"> |
| <fields> |
| <field type="nomeEditor" name="nome"/> |
| <field type="pais" name="pais"/> |
| </fields> |
| <relations> |
| <oneToMany entity="Album" name="Albuns" minOccurs="0" maxOccurs="unbounded"/> |
| </relations> |
| </entity> |

Listagem - Definição da entidade 'Editor'

Na listagem em cima define-se a entidade base ‘Editor’ que tem os campos ‘nome’ e ‘pais’. No que respeita a relações com outras entidades é definida uma relação *1-para-n* com Album podendo-se interpretar que: “Um Editor edita zero ou mais Albuns”.

|  |
| --- |
| <entity type="abstract" name="Album" generatedFileName="Album"> |
| <fields> |
| <field type="tituloAlbum" name="titulo"/> |
| </fields> |
| <relations> |
| <oneToOne entity="LP" name="LP" minOccurs="1" maxOccurs="1"/> |
| <oneToOne entity="EP" name="EP" minOccurs="1" maxOccurs="1"/> |
| </relations> |
| </entity> |

Listagem - Definição da entidade 'Album'

A definição da entidade abstracta ‘Album’, conforme listagem em cima, é feita declarando o campo ‘titulo’ e duas relações *1-para-1*. Estas relações permitem a criação da relação IS-A de ‘Album’ com ‘LP’ e ‘EP’, podendo-se interpretar que: “Um Album tem um e só um LP e um LP tem um e só Album”.

|  |
| --- |
| <entity type="dependent" name="LP" baseEntity="Album" generatedFileName="LP"> |
| <fields> |
| <field type="dataEdicaoLP" name="dtEdicao"/> |
| </fields> |
| <relations> |
| <oneToMany entity="Faixa" name="faixas" minOccurs="1" maxOccurs="unbounded"/> |
| </relations> |
| </entity> |

Listagem - Definição da entidade 'LP'

A definição de ‘LP’, mostrada na , caracteriza uma relação de herança, sendo ‘LP’’ dependente de ‘Album’. Ao nível da entidade está definido o campo ‘dataEdicaoLP’ e a relação *1-para-n* com ‘Faixa’.

|  |
| --- |
| <entity type="dependent" name="EP" baseEntity="Album" generatedFileName="EP"> |
| <fields/> |
| <relations> |
| <oneToMany entity="Faixa" name="faixa" minOccurs="1" maxOccurs="4"/> |
| </relations> |
| </entity> |

Listagem - Definição da entidade 'EP'

Um ‘EP’, pela sua natureza, contém um número reduzido de músicas. Assim, a definição em cima caracteriza a relação de herança entre ‘EP’ e ‘Album’ e a relação *1-para-n* entre ‘EP’ e ‘Faixa’, estabelecendo a cardinalidade dessa relação para um máximo de 4 ocorrências interpretando-se que: “Um EP tem uma ou quatro Faixas”.

|  |
| --- |
| <entity type="base" name="Interprete" generatedFileName="Artista"> |
| <fields> |
| <field type="nomeArtista" name="nome"/> |
| <field type="pais" name="nacionalidade"/> |
| </fields> |
| <relations> |
| <oneToMany entity="Album" name="album" minOccurs="1" maxOccurs="unbounded"/> |
| </relations> |
| </entity> |
| <entity type="base" name="Faixa" generatedFileName="Faixa"> |
| <fields> |
| <field type="nomeMusica" name="nome"/> |
| <field type="tempoMusica" name="duracao"/> |
| <field type="generoMusical" name="genero"/> |
| </fields> |
| </entity> |
| </entities> |

Listagem - Definição das entidades ‘Interprete’ e ‘Faixa’

A definição das entidades ‘Interprete’ e ‘Faixa’ não acrescenta sintaxe nova, em relação ao que já foi definido anteriormente, estando as mesmas de acordo com o diagrama da Figura 2.

Por cada entidade definida será gerada automaticamente a implementação de operações CRUD, não sendo necessário o seu desenvolvimento por parte do programador.

# Persistência em Base de Dados

Uma vez que se pretende que a persistência dos dados seja feita numa base de dados relacional, é necessário que sejam especificados determinados parâmetros que permitam a ligação ao serviço, como é o caso do nome do servidor que a disponibiliza, *username*, *password*, etc. Este tipo de especificação deverá ser feita dentro do elemento *dataEnvironments* através da definição do elemento *provider*. A secção de *environments* é apresentada com maior detalhe na página 24.

|  |
| --- |
| <environments> |
| <dataEnvironments> |
| <provider name="FirstBDServer" type="MsSql2008" |
| connectionString="Server=127.0.0.1;Initial Catalog=iselsample;User |
| Id=iselsample;Password=iselsample;" serverName="127.0.0.1" |
| instance="SQLSERVER1" username="iselsample" |
| password="iselsample" catalog="iselsample"/> |
| </dataEnvironments> |
| </environments> |

Listagem - Parametrização de acesso a servidor de BD

# Processos de negócio

A fim possibilitar definir quais as operações a serem disponibilizadas pela aplicação, terá que se definir o elemento *businessProcesses*, conforme listagem em baixo. Esta definição vai de encontro ao definido pelo Cliente, dividindo-se as operações em dois grupos: Venda e Registo.

|  |
| --- |
| <businessProcesses> |
|  |
| <component name="Venda"> |
| <businessProcess name="EncomendaCliente" description="Processo generico de encomenda"> |
| <input> |
| <param type="tipoAlbum" name="tipoAlbum" minOccurs="1" maxOccurs="1"></param> |
| <param type="identificador" name="idAlbum" minOccurs="1" maxOccurs="1"></param> |
| <param type="canalVendas" name="canal" minOccurs="1" maxOccurs="1"></param> |
| <param type="idCliente" name="idCliente" minOccurs="1" maxOccurs="1"></param> |
| <param type="encomendaQtd" name="encomendaQtd" minOccurs="1" maxOccurs="1"></param> |
| </input> <!-- unbounded --> |
| <output type="idEncomenda" minOccurs="1" maxOccurs="1"/> |
| </businessProcess> |
|  |
| <businessProcess name="ObterEstadoEncomenda" description="Verificação do estado da encomenda"> |
| <input> |
| <param type="idEncomenda" name="idEncomenda" minOccurs="1" maxOccurs="1"></param> |
| </input> |
| <!-- unbounded --> |
| <output type="estadoEncomenda" minOccurs="1" maxOccurs="1"/> |
| </businessProcess> |
|  |
| <businessProcess name="CancelarEncomenda" description="Cancelamento de encomenda"> |
| <input> |
| <param type="idEncomenda" name="idEncomenda" minOccurs="1" maxOccurs="1"></param> |
| </input> |
| <!-- unbounded --> |
| <output type="retornoCancelarEncomenda" minOccurs="1" maxOccurs="1"/> |
| </businessProcess> |
| </component> |
|  |
| <component name="Registo"> |
| <businessProcess name="RegistoCliente" description="Processo generico de registo de cliente"> |
| <input> |
| <param type="userName" name="userName" minOccurs="1" maxOccurs="1"></param> |
| <param type="password" name="password" minOccurs="1" maxOccurs="1"></param> |
| <param type="nomeCompleto" name="nomeCompleto" minOccurs="1" maxOccurs="1"></param> |
| <param type="dtNascimento" name="dtNascimento" minOccurs="1" maxOccurs="1"></param> |
| </input> |
| <!-- unbounded --> |
| <output type="idCliente" minOccurs="1" maxOccurs="1"/> |
| </businessProcess> |
|  |
| <businessProcess name="AlteracaoPassword" description="Processo de alteração de password"> |
| <input> |
| <param type="userName" name="userName" minOccurs="1" maxOccurs="1"></param> |
| <param type="password" name="passwordActual" minOccurs="1" maxOccurs="1"></param> |
| <param type="password" name="passwordFutura" minOccurs="1" maxOccurs="1"></param> |
| </input> |
| <!-- unbounded --> |
| <output type="retornoAlteracaoPassword" minOccurs="1" maxOccurs="1"/> |
| </businessProcess> |
| </component> |
|  |
| </businessProcesses> |

Listagem - Definição de Processos de Negócio

A definição de um processo de negócio caracteriza-se pela definição do protótipo da operação exposta, podendo ou não especificar argumentos ou retorno da operação. A divisão dos processos de negócio é conseguida pela agregação de processos da mesma área em elementos *component* comuns.

# Geração automática de código

A partir deste momento estão reunidas as condições para se conseguir fazer a geração automática de código.

# Geração automática do script de construção do modelo de dados

# Arquitectura da Solução

# Dicionário de Dados

O dicionário de dados é o nome dado ao ficheiro que agrega toda a informação relativa à aplicação a desenvolver. O formato escolhido para este ficheiro foi o XML pela sua simples manipulação e compreensão, por possibilitar a validação do seu conteúdo através de *XML Schema* e por suportar a definição da apresentação do seu conteúdo nas mais diversas formas através de XSLT.

Integrando-se numa aplicação EDM, é no dicionário de dados que o gerador de código vai encontrar a descrição dos elementos que compõem a aplicação, bem como, de quais as operações oferecidas por si. O dialecto criado permite a definição de tipos utilizados e respectivas restrições, entidades e suas relações, servidores de dados e processos de negócio.

# Modelo de Tipos – elemento *<usertypes>*

Devido à necessidade do modelo de tipos garantir correspondência com tipos disponibilizados em linguagens de programação e em motores de bases de dados relacionais, suportando a definição de restrições, optou-se por considerar o modelo presente na especificação XSD. Assim, chegou-se a um subconjunto de tipos, conforme tabela em baixo, que é suficiente para suportar o desenvolvimento de aplicações.

|  |  |  |
| --- | --- | --- |
| Tipo | Eq. C# | Eq. SQL |
| Datetime | Datetime | datetime |
| Long | Long | bigint |
| Int | Int | int |
| Short | Short | short |
| Byte | Byte | byte |
| Double | Double | real |
| Float | Float | float |
| Decimal | Decimal | decimal |
| Boolean | Boolean | bit |
| String | String | varchar |
| Binary | byte[] | binary |

Tabela - Tipos disponibilizados e respectivas correspondências

A definição de um tipo no dicionário de dados é feita através de elementos que têm o nome igual ao tipo que representam (e.g. tipo *int* representado pelo elemento <*int*>), devendo estes estar contidos no elemento *userTypes*. O elemento utilizado para definir um tipo no dicionário obriga à definição do atributo *name*, tendo este que ser único, a fim de identificar inequivocamente o tipo na aplicação, e permite a definição de restrições, dependente do tipo representado.

# Restrições

Tendo como base o modelo de tipos da especificação XSD, identificaram-se para cada tipo o conjunto de restrições aplicáveis, conforme tabela em baixo.

|  |  |
| --- | --- |
| Tipo | Restrições |
| String | length, minLength, maxLength, pattern, enumeration |
| decimal | totalDigits, fractionDigits, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive |
| float | pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive |
| double | pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive |
| datetime | pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive |
| binary | length, minLength, maxLength, pattern, enumeration |
| long | totalDigits, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive |
| int | totalDigits, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive |
| short | totalDigits, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive |
| byte | totalDigits, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive |

Tabela - Restrições aplicáveis a tipos

De notar que a utilização de um tipo, sem especificação de restrições ao mesmo, implica a adopção das características impostas pelas plataformas alvo (e.g. Java, Oracle, C#, SQL,...).

# Definição das Restrições

De seguida são apresentadas com maior detalhe todas as restrições possíveis.

# length

Número de unidades de comprimento. No caso de aplicação ao tipo *string* significa o número exacto de caracteres e, no tipo binário significa o número exacto de bytes. O seu valor deverá ser um número inteiro positivo e será sujeito a validação quando utilizado em conjunto com *minLength* e *maxLength*..

# minLength

Número mínimo de unidades de comprimento. No caso de aplicação ao tipo *string* significa o número mínimo de caracteres e, no tipo binário significa o número mínimo de bytes. O seu valor deverá ser um número inteiro positivo e será sujeito a validação quando utilizado em conjunto com *length* e *maxLength*.

# maxLength

Número máximo de unidades de comprimento. No caso de aplicação ao tipo *string* significa o número máximo de caracteres e, no tipo binário significa o número máximo de bytes. O seu valor deverá ser um número inteiro positivo e será sujeito a validação quando utilizado em conjunto com *length* e *minLength*.

# pattern

Permite caracterizar uma expressão regular que define a máscara de introdução. O seu valor será do tipo *string*.

# enumeration

Restringe os valores possíveis de um tipo aos indicados na restrição, todavia, não impondo relações de ordem no domínio.

# maxInclusive

Valor máximo incluído no intervalo de valores possíveis do domínio. O seu valor deverá ser do tipo a que a restrição diz respeito e será sujeito a validação quando utilizado em conjunto com *minExclusive* e *minInclusive*. A presença desta restrição invalida a utilização de *maxExclusive*.

# maxExclusive

Valor máximo excluído do intervalo de valores possíveis do domínio. O seu valor deverá ser do tipo a que a restrição diz respeito e será sujeito a validação quando utilizado em conjunto com *minExclusive* e *minInclusive*. A presença desta restrição invalida a utilização de *maxInclusive*.

# minInclusive

Valor mínimo incluído no intervalo de valores possíveis do domínio. O seu valor deverá ser do tipo a que a restrição diz respeito e será sujeito a validação quando utilizado em conjunto com *maxExclusive* e *maxInclusive*. A presença desta restrição invalida a utilização de *minExclusive*.

# minExclusive

Valor mínimo excluído do intervalo de valores possíveis do domínio. O seu valor deverá ser do tipo a que a restrição diz respeito e será sujeito a validação quando utilizado em conjunto com *maxExclusive* e *maxInclusive*. A presença desta restrição invalida a utilização de *minInclusive*.

# totalDigits

Define o número máximo de dígitos do tipo que representa.

# fractionDigits

Define o número máximo de dígitos à direita do ponto decimal do tipo que representa

# Entidades – elemento *<entities>*

A definição de entidades permite identificar caracterizar os objectos de domínio da aplicação, sendo o elemento *entities* o agregador de todas as entidades a serem consideradas.

O elemento *entity* caracteriza então uma entidade permitindo a sua utilização na aplicação, a definição dos seus atributos individuais, bem como o tipo de herança a aplicar.

# Definição de uma entidade – Identificação

A identificação de uma entidade na aplicação é feita pela definição obrigatória do atributo *name* do elemento *entiity*. O valor deste atributo é único, sendo tal garantido pelo *XML Schema*.

# Definição de uma entidade – Herança

Devido à necessidade de especificar relações de herança no dicionário de dados, incluiu-se o atributo *type* que indica qual o papel da entidade na hierarquia, podendo o mesmo tomar os valores *abstract*, *base*, *dependent* ou *abstractdependent*. O comportamento esperado para cada valor deste atributo, em conjunto com o atributo *baseEntity* é mostrado na tabela em baixo.

|  |  |
| --- | --- |
| Valor de *type* | Comportamento Esperado |
| base | As entidade que sejam deste tipo serão consideradas, em object model, classes base, podendo servir de ponto de extensibilidade para outras classes. Em modelo relacional constituirão a entidade base de uma associação IS-A. |
| abstract | A entidade gerada, ao nível da linguagem de programação alvo, não será instanciável e obrigará a que entidades derivadas implementem métodos definidos pelo programador. |
| dependent | A entidade gerada derivará da entidade com o nome igual ao valor do atributo *baseEntity*. |
| abstractdependent | Combina as propriedades dos tipos *abstract* e *dependent*. |

Tabela – Relações de herança possíveis no elemento entity

As relações de herança em *object model* são representadas através de uma hierarquia de classes. Em modelo relacional permitem associações do tipo *IS-A*.

A integridade referencial do atributo *baseEntity* é garantida pelo *XML Schema*.

# Atributos de uma entidade – elemento *<fields>*

Conforme mencionado, cada entidade será responsável por enumerar os seus atributos que, deverão estar contidos no elemento *fields*. O elemento *field* é assim o descritor de um campo da entidade em que está inserido. A tabela em baixo detalha quais os atributos que caracterizam o campo de uma entidade.

|  |  |
| --- | --- |
| Atributo | Descrição |
| name | O atributo *name* identifica inequivocamente o campo de uma entidade e, como tal, é garantida a unicidade do mesmo no âmbito de cada uma entidade. |
| type | Este atributo indica qual o tipo de campo, sendo que terá que ser referenciado o atributo *name* de um elemento filho de *userTypes* garantindo assim que o tipo de um campo só pode ser o nome de um tipo definido no âmbito da solução. |
| unique | Este atributo indica quais os campos que identificam inequivocamente uma entidade, permitindo assim, ao nível do *object model*, identificar quais os campos que permitem obter uma instância para entidade específica e, ao nível do modelo relacional, qual o tuplo que deverá ser único na tabela que materializa a entidade. |
| nillable | Indicador da possibilidade de serem usados valores *null* para afectar o campo. |

Tabela - Atributos do elemento field

# Relações entre entidades – elemento *<relation>*

As relações entre entidades são especificadas caracterizando as entidades de origem e destino, a cardinalidade, a existência da relação inversa, e o tipo de relação. Os tipos de relação suportados pelo dicionário de dados são *1-para-1*, *1-para-n, n-para-1* e *n-para-n*, sendo os mesmos detalhados de seguida.

# Relações *1-para-n* e *n-para-1 (OneToMany e ManyToOne)*

Este tipo de relações evidencia diferenças de representação de uma relação em *object model* e em modelo relacional. No sentido de melhor perceber a forma como se declaram relações deste tipo e quais as consequências dessa declaração, considere-se o seguinte exemplo:

|  |
| --- |
| <relation type="OneToMany" name="Comments" oneEntity="Post" manyEntity="Comment" |
| nillable="false" minOccurs="1" maxOccurs="unbounded" inverse="true"/> |

Listagem - Declaração de relação 1-n

A leitura que pode ser feita é a seguinte: *“um ‘Post’ tem vários ‘Comment’ obrigando à existência de pelo menos 1”*. Com base neste requisito, a representação possível desta relação em *object model* poderia ser feita da seguinte forma:

|  |
| --- |
| public class Comment { |
| //Comment fields.... |
| } |
|  |
| public class Post { |
| //Post fields.... |
| IList<Comment> \_Comments = new List<Comment>(); |
| } |
|  |

Listagem - Relação 1-n em object model

No que respeita ao modelo relacional, a modelação apropriada seria a apresentada na figura em baixo.



Figura - Modelação de relação 1-n

A principal diferença entre ambas as representações tem a haver com qual a entidade que guarda referência para a outra entidade da relação. No caso do *object model* a entidade do lado ‘*one’* guarda uma colecção de referências para a entidade do lado ‘*many’* e, no caso do modelo relacional, a entidade do lado ‘*many’* guarda uma referência para a entidade *‘one’*.

De acordo com o exemplo anterior, em *object model*, é então possível obter uma lista de objectos *‘Comment’* associados a um *‘Post’*. Todavia, podem haver cenários em que seja útil que um objecto *‘Comment’* guarde a referência para o objecto *‘Post’* em que aparece, i.e., que seja referencie a entidade no outro lado da relação. Nesse sentido são disponibilizadas as relações *ManyToOne* que assumem uma declaração idêntica, conforme listagem em baixo.

|  |
| --- |
| <relation type="ManyToOne" name="Comments" manyEntity="Comment" oneEntity="Post" |
| nillable="false" minOccurs="1" maxOccurs="1" inverse="true"/> |
|  |

Listagem - Declaração de relação n-1

O atributo *‘inverse’* visa facilitar a declaração de relações em que se pretende a criação de atributos no lado oposto à entidade a que se refere. Assim, uma relação *OneToMany* com o atributo *inverse* a *true* equivale à declaração de duas relações com o atributo *inverse* a *false*: uma relação *OneToMany* e uma outra *ManyToOne*.

# Relações *n-para-n (ManyToMany)*

As relações *n-para-n* implicam que no modelo relacional seja utilizada uma tabela associativa para as representar. Nesse sentido, a declaração de uma relação deste tipo pressupõe a existência de uma entidade criada para esse efeito. A entidade associativa pode assim conter os seus próprios campos sendo que irá incluir também campos relativos às entidades com que se relaciona.

# Relações *1-para-1*

As declaração de relações *1-para-1* não é feita de forma explícita no elemento *relations*, mas sim através da especificação de relações de herança entre entidades (Página 21) definindo uma relação IS-A entre entidades. Assim, ao nível do *objecto model*, a relação irá definir a herança típica do paradigma orientado a objectos e, ao nível do modelo relacional, a entidade base agregará todos os seus campos, bem como, os campos de entidades derivadas.

# Serviços externos – elemento *<environments>*

O elemento *environments* tem o propósito de possibilitar a descrição de serviços externos passíveis de serem utilizados no âmbito da aplicação. Este será responsável por agregar os elementos que irão descrever a forma como se pode aceder a determinado serviço, tipicamente pela via de uma ligação.

# Serviços de Base de Dados – elemento *<dataEnvironments>*

As fontes de dados, como parte integrante da maioria das aplicações, terão que ser referenciadas, não só para a obtenção/persistência de informação, mas também para permitir que o gerador de código consiga criar automaticamente o *script* de construção do modelo de dados.

Um *dataEnvironment* deverá conter um elemento *provider* que descreve o servidor de base de dados através dos atributos presentes na tabela em baixo.

|  |  |  |
| --- | --- | --- |
| Atributo | Observações | Obrigatório |
| name | Nome que identifica inequivocamente o servidor. | X |
| type | Este é o atributo responsável por indicar à camada de acesso a dados qual o tipo de servidor de dados. Corresponde a um enumerado que pode tomar os valores: *MsSql2000*, *MsSql2005*, *MsSqlCe*, *MySQL*, *MySQL5*, *PostgreSQL*, *Informix* e *Ingres.* | X |
| connectionString | String de conecção ao servidor | X |
| serverName | Nome ou IP do servidor | X |
| instance | Nome da instância do serviço | X |
| catalog | Nome da base de dados da instância | X |
| port | Porto no servidor |  |
| username | Nome de utilizador | X |
| password | Password | X |
| security | Informação relativa à segurança na ligação com o servidor |  |
| timeout | Timeout associado à ligação |  |

Tabela - Atributos do elemento provider de dataEnvironments

# Processos de Negócio – elemento *<businessProcesses>*

Os processos de negócio foram incluídos no dicionário de dados com o objectivo de possibilitar a declaração de operações a serem expostas. A declaração dessas operações é feita com recurso a elementos *businessProcess* que devem estar contidos em elementos *component*, conforme listagem em baixo. A função deste último é a de agregar processos de negócio que dizem respeito a áreas semelhantes (e.g. processos de negócio expostos pelos mesmos departamentos).

|  |
| --- |
| <businessProcesses> |
| <component name="Departamento\_A"> |
| <businessProcess name="Op\_A" description="......">....</businessProcess> |
| ....... |
| <businessProcess name="Op\_n" description=".......">....</businessProcess> |
| </component> |
| ....... |
| <component name="Departamento\_n"> |
| ....... |
| </component> |
| </businessProcesses> |

Listagem - Declaração de processos de negócio

# Processo de Negócio – elemento *<businessProcess>*

Um processo de negócio será materializado num protótipo de um método pertencente à classe representada pelo componente que o agrega. Assim, fazendo a analogia do processo de negócio ao método, é possível definir parâmetros de entrada e tipo de retorno do processo, bem como, a ausência de um ou ambos, sendo que, neste último cenário se está perante um método que não aceita parâmetros e que retorna *void*.

Além de permitir a definição de parâmetros de entrada e tipo de retorno, o elemento *businessProcess* é caracterizado pelos seguintes atributos:

|  |  |  |
| --- | --- | --- |
| Atributo | Observações | Obrigatório |
| name | Nome que identifica inequivocamente o processo de negócio, no âmbito do elemento *component* em que se insere. | X |
| description | Descrição do processo de negócio. | X |

Tabela - Atributos do elemento businessProcess

# Parâmetros – elemento *<input>* e *<param>*

Os parâmetros de um processo de negócio devem ser especificados no elemento *input* através de elementos *param*. Cada parâmetro é caracterizado pelos seguintes atributos:

|  |  |  |
| --- | --- | --- |
| Atributo | Observações | Obrigatório |
| type | Tipo do parâmetro sendo garantida a integridade referencial com os nomes de tipos definidos no elemento *userTypes*. Desta forma, os parâmetros de processos de negócio apenas podem ser de tipos definidos no dicionário. | X |
| name | Nome que identifica inequivocamente o parâmetro, no âmbito do processo de negócio a que se refere. | X |
| minOccurs | Número mínimo de ocorrências do parâmetro, tomando o atributo o valor fixo de ‘1’. | X |
| maxOccurs | Número máximo de ocorrências do parâmetro, tomando o atributo valores entre ‘1’ e ‘*unbounded*’. Na geração automática de código, quando o valor deste atributo for maior que ‘1’, o resultado irá traduzir-se na utilização de um contentor para representar o parâmetro. | X |

Tabela - Atributos do elemento param

# Retorno – elemento *<output>*

Este elemento será responsável por identificar o tipo de retorno do processo de negócio, sendo caracterizado pelos seguintes atributos:

|  |  |  |
| --- | --- | --- |
| Atributo | Observações | Obrigatório |
| type | Tipo de retorno sendo também garantida a integridade referencial, à imagem do atributo *type* de *param.* | X |
| minOccurs | Número mínimo de ocorrências do parâmetro, tomando o atributo um valor fixo de ‘1’. | X |
| maxOccurs | Número máximo de ocorrências do parâmetro, tomando o atributo valores entre ‘1’ e ‘*unbounded*’, à imagem do atributo *type* de *param*. | X |

Tabela - Atributos do elemento output

# Estrutura de uma Solução EDM no Visual Studio

Conforme se verificou anteriormente (Página 18), no dicionário de dados é possível definir o modelo de tipos da aplicação, entidades que nela participam, serviços externos e processos de negócio. Assim, uma Solução EDM está estruturada para conseguir, de forma organizada, representar a materialização dessas definições em código fonte, conforme figura em baixo.

Figura - Estrutura de uma Solução EDM

# Metadados

Os metadados da Solução EDM dizem respeito ao dicionário de dados, disponibilizado através do ficheiro ‘3D.xml’, e ao seu *XML Schema*, disponibilizado através do ficheiro ‘3dValidator.xsd’. Ambos estão contidos num *Solution Folder* com o nome ‘3D’. Este será o local de onde o gerador de código irá obter o conteúdo do dicionário.

# Biblioteca

A biblioteca da Solução EDM é o local onde estão armazenados os *assemblies* referenciados pelos projectos, nomeadamente o *assembly* ‘EDM.FoundationClasses.dll’ que será detalhado mais à frente (Página XXX). Todas estes ficheiros estão contidos .no *Solution Folder* com o nome ‘Assembly’.

# Projectos

Os projectos presentes na Solução EDM visam separar as implementações das várias secções do dicionário de dados, havendo dependências de uns para os outros, conforme se verá mais à frente. O nome dos projectos irá depender do nome da empresa Cliente e do nome da aplicação, definidos na fase de criação da solução (Página XXX), sendo a máscara para os mesmos: <*companyName>.<projectName>.*<nome\_do\_projecto>

A estrutura dos projectos foi desenhada de forma a possibilitar geração automática de código sem destruir código já desenvolvido pelo programador. Nesse sentido, adoptou-se o conceito de *proxy* sendo que todos os ficheiros susceptíveis de serem gerados várias vezes são guardados em pastas específicas (tipicamente com o nome ‘*Base*’) e, os ficheiros a serem alterados pelo programador, os *proxies*, disponibilizados na raiz do projecto.

# Projecto *Rtti*

O projecto Rtti é onde estão materializados os tipos definidos no elemento *userTypes* dicionário de dados (Página 18), bem como, as restrições definidas para estes. A sua estrutura é a apresentada na figura em baixo.

Figura - Estrutura do projecto 'Rtti'

A pasta ‘Base’ contém a classe ‘*BaseUserTypeMetadata’* que contém o código gerado automaticamente responsável por criar instâncias de *IUserType<T>* (Página XXX). Na raiz do projecto encontra-se o proxy para a classe referenciada onde o programador pode proceder à criação de novos tipos além dos definidos no dicionário de dados.

# Projecto *Entity*

O projecto *Entity* contém a implementação dos objectos de domínio definidos no elemento *entities* do dicionário de dados, bem como, a implementação de objectos DAO que viabilizam o relacionamento entre *object model* e modelo relacional. A sua estrutura é a apresentada na figura em baixo.

Figura - Estrutura do projecto 'Entity'

Os objectos de domínio gerados automaticamente estão presentes na pasta ‘*Domain’* e os *proxies* para os mesmos na raíz do projecto.

Relativamente ao objecto *factory* de DAOs, a interface do mesmo está definida em ‘*IDAOFactoryBase’*, sendo implementado pela classe ‘*NHibernateDaoFactoryBase’*. O *proxy* para este objecto é disponibilizado pela classe *‘NHibernateDaoFactory’*.

A interface base do DAO de cada entidade está definida em *‘IDaoInterfaceBase’*, sendo este ficheiro gerado por cada sincronização com o dicionário de dados. Por forma a permitir alterações à interface, é disponibilizado o *proxy* para a mesma em *‘IDaoInterface’*. Todavia, devido à possibilidade de serem incluídas novas interfaces DAO após sincronização com o dicionário de dados, optou-se por marcar a *proxy* para cada interface com o atributo *partial*. Assim, por cada sincronização com o dicionário de dados os ficheiros, além de serem geradas as interfaces base DAO para cada entidade, é também gerada a interface parcial em *‘IDaoIntefacePartial’*, garantindo assim que o comportamento adicionado pelo programador para o DAO de determinada entidade se mantém inalterado.

A concretização das interfaces DAO assenta sobre o mesmo principio de gerar as classes DAO de cada entidade por cada sincronização em *‘DAOImplementationPartial’* e disponibilizar o *proxy* em *‘DAOImplementation’*.

# Projecto *Services*

O projecto Services agrega a implementação de operações CRUD (*Create, Read, Update, Delete*) sobre as entidades, bem como, a definição de protótipos relativos aos processos de negócio definidos no dicionário de dados. A sua estrutura é a apresentada na figura em baixo.

Figura - Estrutura do projecto 'Services'

A pasta ‘*Base*’ contém as classes geradas automaticamente para as operações CRUD sobre cada entidade definida no dicionário e, na raiz do projecto estão disponíveis os *proxies* para as mesmas. Na pasta ‘*Base*’ são também geradas classes com o nome definido no atributo *name* do elemento *component* do dicionário de dados (Página 25), contendo métodos com a assinatura especificada no processo de negócio relativo, sendo igualmente disponibilizados os *proxies* para as mesmas na raiz do projecto.

É no projecto Services que está implementado o mecanismo de segurança no acesso às operações CRUD de cada entidade, bem como, no acesso aos processos de negócio expostos (Página XXX).

# Projecto *Ws*

O projecto Ws é o responsável por expor as operações disponibilizadas no projecto Services através de *webservices*. Após a geração automática de código, este é o projecto que garante de imediato acesso às entidades definidas no dicionário de dados sem ser necessário a escrita de uma linha de código. A sua estrutura é a apresentada na figura em baixo.

Figura - Estrutura do projecto 'Ws'

A pasta ‘*Base*’ contém os *webservices* gerados automaticamente, constituídos pelas operações CRUD, no caso das entidades, e pelos processos de negócio, no caso de componentes. Na raiz do projecto são disponibilizados os *proxies* para os mesmos.

Sendo um projecto *web*, é também incluído o ficheiro *web.config* que possibilita especificar a existência de um tipo que extende o mecanismo de permissões adoptado (Página XXX). É também aqui que está registado o módulo responsável por estabelecer o início e fim de uma transacção com o servidor de base de dados e, nesse sentido, dado ter sido adoptada a *framework NHibernate*, é disponibilizado o ficheiro *‘hibernate.cfg.xml’* que contém as configurações necessárias ao mesmo.

# Projecto *UnitTest*

O projecto *UnitTest* contém testes unitários elementares paraa cada entidade definida no dicionário de dados. A sua estrutura é a apresentada na figura em baixo.

Figura - Estrutura do projecto 'UnitTest'

À imagem do definido para os restantes projectos, a pasta *‘Base’* contém as implementações de testes unitários geradas automaticamente e, a raiz do projecto contém os *proxies* para os mesmos.

# Gerador de Código *(EDM.Generator)*

# Contexto de Geração *(EDM.Generator.Context)*

O contexto de geração é a entidade responsável por identificar quais os intervenientes no processo de geração de código para a aplicação.



Figura - Diagrama UML de classes do contexto de geração

Assim, para o tratamento do *input* do processo de geração é utilizada a classe *ThreeD* que encapsula o dicionário de dados e consegue navegar na sua estrutura, auxiliada pela classe *ThreeDXPath*. No decorrer da geração serão aplicadas transformações XSLT, sendo estas manipuladas pela classe *Transform*. O resultado da geração é conhecido pela classe *Output*, sendo capaz de identificar o caminho físico absoluto dos vários constituintes da Solução EDM utilizada.

# Motor de Geração *(EDM.Generator.Engine)*

O motor de geração é a entidade que, utilizando as ferramentas presentes no contexto de geração, promove a criação automática de código.



Figura - Diagrama UML de classes do motor de geração

Uma vez que um dos objectivos do projecto é a possibilidade de geração de código para várias linguagens, e.g. Java, optou-se por não comprometer a classe abstracta *Engine* com uma estratégia de geração específica. Dessa forma, foi criada a classe *CSharpEngine* que implementa a estratégia de geração de código por passos.

# Pipeline de Transformação (*GeneratorSteps)*

A estratégia de geração de código adoptada consiste no conceito de um *pipeline* de etapas que desempenham funções específicas. A divisão por etapas visa não só a separação da lógica de geração associada a diferentes elementos, mas também a facilidade de inclusão de novas etapas eexclusão de etapas em actividade. A Figura 15 mostra a estrutura desenhada.



Figura - Diagrama UML de classes da geração por etapas

A classe *CSharpEngine* agrega objectos do tipo *AbstractStep* que representam etapas na geração de código. A classe *GeneratorSteps* é um *decorator* de *AbstractStep*, sendo esta a classe responsável por materializar o conceito de *pipeline* de transformação, agregando instâncias de especializações de *AbstractStep*.

Os dois tipos de etapas utilizadas (*ThreeDPreExecuteStep* e *Step*) são detalhados de seguida.

# Etapa *‘ThreeDPreExecuteStep’*

Esta etapa é o ponto inicial de geração. Aqui são feitas verificações ao dicionário de dados, bem como, adição de informação que visa facilitar a execução de etapas subsequentes.

# Verificação de utilização de palavras reservadas

Devido a restrições impostas pelas linguagens, é necessário garantir que não estão a ser utilizadas palavras reservadas, neste caso do C#, que possam impedir a compilação da aplicação. Assim, são verificados nomes de tipos, entidades, campos de entidades, componentes de processos de negócio, nomes de processos de negócio e os seus parâmetros. Caso algum destes elementos utilize uma palavra reservada (e.g. *stackalloc*) é lançada a excepção ‘*KeyWordUsageException’* com informação acerca do elemento causador do erro.

Após o primeiro passo de verificação inicia a fase de adição de informação ao dicionário. Em baixo são detalhados quais os elementos alterados neste passo, quais as alterações feitas e qual o seu objectivo.

# Alteração de elementos <*field>*

Neste elemento é incluído o atributo ‘*nillable’*, em elementos *‘field’* que não o tenham definido, tomando este o valor ‘*false’*. Esta alteração visa dar a informação acerca da possibilidade de um campo assumir valores *null* no modelo relacional, informação esta necessária à criação dos ficheiros de *mapping* do *NHibernate* (Página XXX).

É também incluído o atributo ‘*edmType’* que terá o nome do elemento que define o tipo do campo em tratamento especificado no atributo ‘*type’* de ‘*field’*. O objectivo desta alteração é o de facilitar a geração do código relativo à declaração dos campos das entidades.

# Alteração do elemento <*solution*>

Inclusão do atributo ‘*namespace*’ recorrendo aos valores dos atributos ‘*companyName*’ e ‘*projectName*’. O valor do atributo final é então determinado da seguinte forma: *‘<companyName>.<projectName>’*. Esta alteração visa facilitar a referências ao *namespace* da solução nos blocos *using* de várias classes geradas.

# Alteração do elemento <*userTypes*>

Inclusão dos atributos ‘*namespace*’ e ‘*assemblyName*’, identificando assim o elemento com o *namespace* da aplicação e o nome do *assembly* do projecto de tipos (Página 29). Desta forma, ao ser gerado código para o projecto de tipos da aplicação a transformação XSLT apenas precisa de percorrer o elemento *userTypes*.

# Alteração do elemento <*entities*> e elementos *<entity>*

Inclusão do atributo ‘*namespace’* ao elemento *entities*, identificando o *namespace* do projecto de entidades (Página 29). No que respeita aos elementos *entity*, são feitas alterações no sentido de identificar os *namespaces* dos projectos onde participam: projecto de entidades, serviços, *webservices* e testes unitários, sendo acrescentados atributos específicos a cada caso. No sentido de facilitar a escrita em disco dos ficheiros gerados é também acrescentado o atributo *‘targetDomainPath’* a cada elemento *entity*, identificado o caminho absoluto do projecto de entidades.

# Alterações ao elemento *<relations>*

# Alterações ao *<businessProcesses>*

Os elementos *component* de *businessProcesses*, conforme visto na Estrutura de uma Solução EDM (Página 30), materializam-se em ficheiros nos projectos de serviços e de *webservices*. Assim são adicionados os *namespaces* desses projectos a cada elemento *component* de *businessProcesses*, bem como, atributos relativos aos *namespaces* dos projectos de entidades e de tipos.

Em relação aos parâmetros de cada *businessProcess*, bem como, ao seu retorno, é também incluído o atributo ‘*edmType’* que, à imagem do que acontece com os elementos *fields* (Página 35), terá o nome do elemento que define o tipo do campo em tratamento especificado no atributo ‘*type’* de ‘*field’*.

# Etapa *‘Step’*

Esta etapa representa o passo no *pipeline* de transformação, responsável por interpretar parte do dicionário de dados e gerar o respectivo código. Para tal, uma instância de *Step* precisa de determinar qual o fragmento do dicionário que irá transformar, recorrendo para isso a uma expressão XPATH, saber o nome da transformação XSLT que deverá aplicar, o caminho físico de destino do ficheiro gerado e a *flag mandatory,* indicando se o ficheiro a gerar deverá ser reescrito. De acordo com o conceito de *proxy* utilizado na geração de código (Página 28), todas as classes passíveis de serem alteradas pelo programador são geradas com a *flag mandatory* a *false*, impedindo assim a destruição das implementações das mesmas.

Considerando o resultado pretendido, recorre-se a várias instâncias de *Step* para popular os projectos que constituem a estrutura da solução, detalhando-se de seguida os tipos de transformações aplicadas por projecto.

# Transformações direccionadas ao projecto de tipos

A geração de código para o projecto de tipos envolve a criação das classes *‘BaseUserTypeMetadata’* e *‘UserTypeMetada’*, de acordo com o definido na estrutura do projecto de tipos (Página 29). Para tal são utilizados os elementos filhos de *userTypes* que especificam os nomes das instâncias de *IUserType*, bem como os valores das restrições aplicadas.

# Transformações direccionadas ao projecto de entidades

A geração de código para este projecto envolve a criação não só dos objectos de domínio, mas também das interfaces DAO que permitem a relação entre o *object model* e o modelo relacional.

|  |  |
| --- | --- |
| Membro | Observações |
| Campos de instância | São criadas propriedades *getters* e *setters* referentes aos campos definidos no elemento *fields*. |
| Campos de relações *oneTomany* | Os campos deste tipo de relações são materializados sob a forma de uma colecção do tipo *IList<*T*>*, onde o parâmetro genérico T |
| maxOccurs | Número máximo de ocorrências do parâmetro, tomando o atributo valores entre ‘1’ inclusivé e ‘*unbounded*’, à imagem do atributo *type* de *param*. |

# Campos de Entidades

Para cada entidade são criadas propriedades *getters* e *setters* relativas aos seus campos, definidos no elemento *fields*.

são também criados membros relativos às relações que existem entre outras entidades.

# Trabalhos Citados

Biron, P. V., & Malhotra, A. (2 de May de 2001). *XML Schema Part 2: Datatypes*. Obtido de W3C Recommendation: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#rf-facets