**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

[1 Introdução *(retirado da proposta de projecto)* 6](#_Toc271787452)

[2 Exemplo de Utilização 7](#_Toc271787453)

[2.1 Análise do Problema 7](#_Toc271787454)

[2.2 Criação de Solução no Visual Studio 8](#_Toc271787455)

[2.3 Tipos do Dominio 9](#_Toc271787456)

[2.4 Entidades do Dominio 11](#_Toc271787457)

[2.5 Persistência em Base de Dados 12](#_Toc271787458)

[2.6 Processos de negócio 13](#_Toc271787459)

[2.7 Geração automática de código 14](#_Toc271787460)

[2.8 Geração automática do script de construção do modelo de dados 15](#_Toc271787461)

[3 Arquitectura da Solução 16](#_Toc271787462)

[4 Dicionário de Dados 17](#_Toc271787463)

[*4.1* Modelo de Tipos – elemento *<usertypes>* 17](#_Toc271787464)

[*4.1.1* Restrições 18](#_Toc271787465)

[4.1.2 Definição das Restrições 18](#_Toc271787466)

[4.1.2.1 length 18](#_Toc271787467)

[4.1.2.2 minLength 18](#_Toc271787468)

[4.1.2.3 maxLength 19](#_Toc271787469)

[4.1.2.4 pattern 19](#_Toc271787470)

[4.1.2.5 enumeration 19](#_Toc271787471)

[4.1.2.6 maxInclusive 19](#_Toc271787472)

[4.1.2.7 maxExclusive 19](#_Toc271787473)

[4.1.2.8 minInclusive 19](#_Toc271787474)

[4.1.2.9 minExclusive 19](#_Toc271787475)

[4.1.2.10 totalDigits 20](#_Toc271787476)

[4.1.2.11 fractionDigits 20](#_Toc271787477)

[*4.2* Entidades – elemento *<entities>* 20](#_Toc271787478)

[*4.2.1* Definição de uma entidade – Identificação 20](#_Toc271787479)

[*4.2.2* Definição de uma entidade – Herança 20](#_Toc271787480)

[*4.2.3* Atributos de uma entidade – elemento *<fields>* 21](#_Toc271787481)

[4.2.3.1 type 21](#_Toc271787482)

[4.2.3.2 unique 22](#_Toc271787483)

[4.2.3.3 visible 22](#_Toc271787484)

[4.2.3.4 nillable 22](#_Toc271787485)

[4.2.3.5 sequence 22](#_Toc271787486)

[4.2.3.6 searchable 22](#_Toc271787487)

[*4.2.4* Relações entre entidades – elemento *<relations>* 22](#_Toc271787488)

[*4.2.5* Relações 1-1 – elemento *<oneToOne>* 23](#_Toc271787489)

[*4.2.6* Relações 1-n – elemento *<oneToMany>* 23](#_Toc271787490)

[*4.2.7* Relações n-n – elemento *<ManyToMany>* 24](#_Toc271787491)

[*4.3* Serviços externos – elemento *<environments>* 24](#_Toc271787492)

[*4.3.1* Serviços de Base de Dados – elemento *<dataEnvironments>* 24](#_Toc271787493)

[*4.4* Processos de Negócio – elemento *<businessProcesses>* 25](#_Toc271787494)

[*4.4.1* Processo de Negócio – elemento *<businessProcess>* 25](#_Toc271787495)

[*4.4.1.1* Parâmetros – elemento *<input>* e *<param>* 26](#_Toc271787496)

[*4.4.1.2* Retorno – elemento *<output>* 26](#_Toc271787497)

[*4.5* Permissões 26](#_Toc271787498)

[Trabalhos Citados 28](#_Toc271787499)

## Índice de Figuras

[Figura 1 – Fases do desenvolvimento de software 6](#_Toc271787672)

[Figura 2 – Modelo Entidade Associação 7](#_Toc271787673)

[Figura 3 - Criação de solução EDM no Visual Studio 8](#_Toc271787674)

[Figura 4 - Formulário de identificação da solução EDM 9](#_Toc271787675)

[Figura 5 - Estrutura de um template EDM no Visual Studio 9](#_Toc271787676)

## Índice de Listagens

[Listagem 1 – Definição dos tipos do dominio 10](#_Toc271787677)

[Listagem 2 - Definição da entidade 'Editor' 11](#_Toc271787678)

[Listagem 3 - Definição da entidade 'Album' 11](#_Toc271787679)

[Listagem 4 - Definição da entidade 'LP' 12](#_Toc271787680)

[Listagem 5 - Definição da entidade 'EP' 12](#_Toc271787681)

[Listagem 7 - Definição das entidades ‘Interprete’ e ‘Faixa’ 12](#_Toc271787682)

[Listagem 8 - Parametrização de acesso a servidor de BD 13](#_Toc271787683)

[Listagem 9 - Definição de Processos de Negócio 14](#_Toc271787684)

[Listagem 10 - Schema do elemento field 21](#_Toc271787685)

[Listagem 11 - Schema do elemento oneToOne 23](#_Toc271787686)

[Listagem 12 - Schema do elemento oneToMany 23](#_Toc271787687)

[Listagem 13 - Schema do elemento manyToMany 24](#_Toc271787688)

[Listagem 14 - Declaração de processos de negócio 25](#_Toc271787689)

## Índice de Tabelas

[Tabela 1 - Tipos disponibilizados e respectivas correspondências 17](#_Toc271787690)

[Tabela 2 - Restrições aplicáveis a tipos 18](#_Toc271787691)

[Tabela 3 - Valores possíveis do enumerado type do elemento entity 21](#_Toc271787692)

[Tabela 4 - Atributos do elemento provider de dataEnvironments 25](#_Toc271787693)

[Tabela 5 - Atributos do elemento businessProcess 26](#_Toc271787694)

[Tabela 6 - Atributos do elemento param 26](#_Toc271787695)

[Tabela 7 - Atributos do elemento output 26](#_Toc271787696)

# 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 facilitar a compreensão de qual a forma de 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

- processos de negócio e respectivas permissões

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.

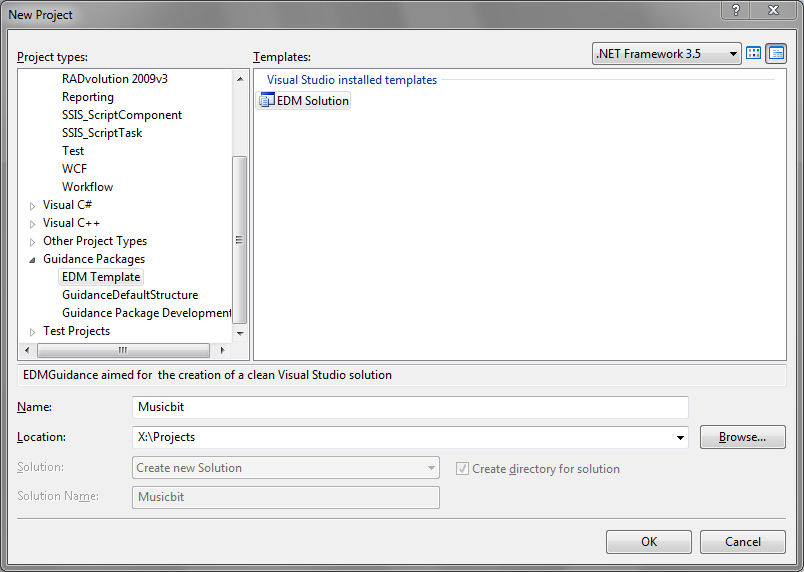


Figura - Criação de solução EDM no Visual Studio

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.

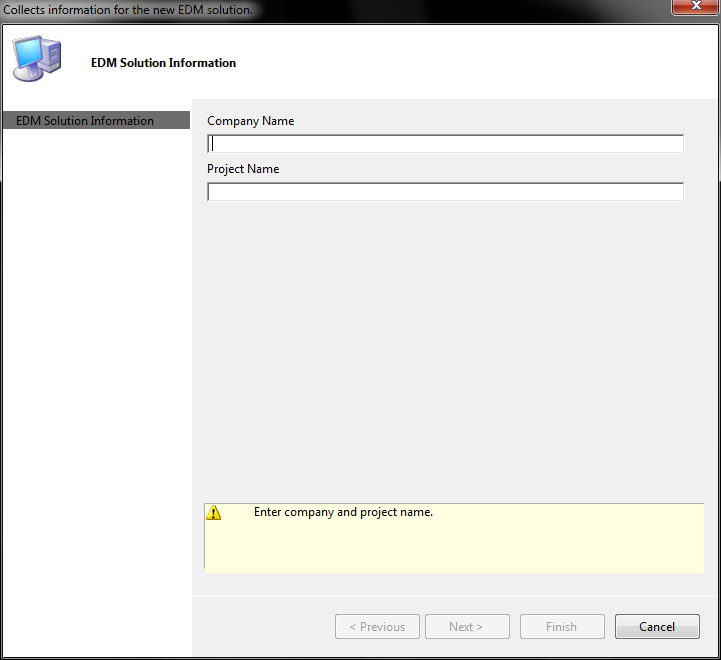


Figura - Formulário de identificação da solução EDM

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.

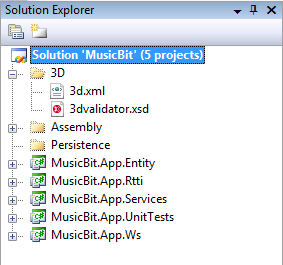


Figura - Estrutura de um template EDM no Visual Studio

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

# Tipos do Dominio

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 15).

# Entidades do Dominio

A fim de possiblitar a criação de objectos de dominio 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, dvidindo-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

# 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 sub-conjunto 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, indentificou-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 detalhes 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 incluido no intervalo de valores possíveis do dominio. 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 excluido do intervalo de valores possíveis do dominio. 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 incluido no intervalo de valores possíveis do dominio. 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 excluido do intervalo de valores possíveis do dominio. 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 digitos do tipo que representa.

# fractionDigits

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

# Entidades – elemento *<entities>*

A definição de entidades visa definir os objectos de dominio 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, do tipo de herança a aplicar e das relações com outros objectos.

# 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*. É igualmente exigida a definição do atributo *generatedFilename* que indicará ao gerador de código qual o nome do ficheiro gerado correspondente à entidade, sendo também este atributo único.

# 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* ou *dependent*. 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. A especificação do atributo *baseEntity* indicará que a deriva de outra. |
| dependent | A entidade gerada derivará da entidade com o nome igual ao valor do atributo *baseEntity*. |

Tabela - Valores possíveis do enumerado type do 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 atributo da entidade e o seu *schema* é o apresentado na listagem seguinte.

|  |
| --- |
| <xs:complexType name="t\_field"> |
| <xs:attribute name="name" type="xs:string" use="required"/> |
| <xs:attribute name="type" type="xs:string" use="required"/> |
| <xs:attribute name="unique" type="xs:boolean" use="optional" default="false"/> |
| <xs:attribute name="visible" type="xs:boolean" use="optional" default="false"/> |
| <xs:attribute name="nillable" type="xs:boolean" use="optional" default="false"/> |
| <xs:attribute name="sequence" type="xs:boolean" use="optional" default="false"/> |
| <xs:attribute name="searchable" type="xs:boolean" use="optional" default="false"/> |
| </xs:complexType> |

Listagem - Schema do elemento field

O atributo *name* identifica inequívocamente o campo de uma entidade e, como tal, é garantida a unicidade do mesmo no âmbito de cada uma. Detalham-se de seguida os restantes atributos que compõem um campo.

# 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.

# visible

Indicador de visibilidade do campo, a ser utilizado ao nível do *object model*.

# nillable

Indicador da possibilidade de serem usados valores *null* para afectar o campo.

# sequence

**QUAL O USO PARA O SEQUENCE?**

# searchable

Este atributo irá permitir a definição de índices no modelo relacional.

# Relações entre entidades – elemento *<relations>*

As relações possíveis entre entidades são *um-para-um*, *um-para-n* ou *n-para-n* e, a forma como cada uma se representa em *object model* e no modelo relacional é diferente.

As relações entre entidades deverão ser especificadas dentro do elemento *relations*. No sentido de permitir uma maior versatilidade na representação de relações, decidiu-se pela criação de elementos distintos para representar cada tipo de relação.

Para cada um dos tipos de relação é obrigatório indicar o nome do campo que a representará e o nome da entidade referenciada, bem como, a cardinalidade dessa relação. É neste último que surgem as diferenças entre os tipos de relações.

# Relações 1-1 – elemento *<oneToOne>*

Uma relação *1-para-1* irá materializar-se de igual forma tanto em *object model* como no modelo relacional. Devido à natureza de uma relação deste tipo, i.e., a partilha da mesma chave pelas entidades envolvidas, este é o tipo de relação que permite definir relações do tipo IS-A.

No que respeita à cardinalidade, é possível definir o número mínimo e máximo de ocorrências, contudo, este terá que assumir o valor ‘1’. A opção de manter estes dois atributos, ainda que assumam sempre o mesmo valor, tem a haver com a uniformização dos elementos que definem as relações.

|  |
| --- |
| <xs:complexType name="t\_oneToOne\_relation"> |
| <xs:attribute name="name" type="xs:string" use="required"/> |
| <xs:attribute name="entity" type="xs:string" use="required"/> |
| <xs:attribute name="minOccurs" type="t\_one" use="required"/> |
| <xs:attribute name="maxOccurs" type="t\_one" use="required"/> |
| </xs:complexType> |

Listagem - Schema do elemento oneToOne

# Relações 1-n – elemento *<oneToMany>*

Uma relação *1-para-n* tem materializações diferentes em *object model* e no modelo relacional. Quando na entidade de origem se define uma relação deste tipo, ao nível do *object model*, vai ser criado um contentor com referências para objectos do tipo de entidade de destino. Ao nível do modelo relacional, a tabela que representa a entidade de destino irá conter uma coluna para referir uma linha da tabela da entidade de origem.

|  |
| --- |
| <xs:complexType name="t\_oneToMany\_relation "> |
| <xs:attribute name="name" type="xs:string" use="required"/> |
| <xs:attribute name="entity" type="xs:string" use="required"/> |
| <xs:attribute name="minOccurs" type="t\_zeroOrOne" use="required"/> |
| <xs:attribute name="maxOccurs" type="t\_oneOrMany" use="required"/> |
| </xs:complexType> |

Listagem - Schema do elemento oneToMany

# Relações n-n – elemento *<ManyToMany>*

A declaração de uma relação *n-para-n* é onde se acentuam as diferenças entre a sua materialização em *object model* e no modelo relacional. Quando uma entidade de origem A define esta relação com a entidade de destino B, ao nível do *object model*, A terá um contentor de referências para B e vice-versa. Ao nível do modelo relacional terá que ser criada uma entidade associativa, no sentido de se conseguirem fazer as associações.

|  |
| --- |
| <xs:complexType name="t\_manyToMany\_relation "> |
| <xs:attribute name="name" type="xs:string" use="required"/> |
| <xs:attribute name="entity" type="xs:string" use="required"/> |
| <xs:attribute name="minOccurs" type="t\_zeroOrOne" use="required"/> |
| <xs:attribute name="maxOccurs" type="t\_oneOrMany" use="required"/> |
| </xs:complexType> |

Listagem - Schema do elemento manyToMany

# 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 conecção.

# 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 pela aplicação. 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 retornom 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 elemento *param* 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 um valor fixo de ‘1’. | X |
| maxOccurs | Número máximo de ocorrências do parâmetro, tomando o atributo valores entre ‘1’ inclusivé 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’ inclusivé e ‘*unbounded*’, à imagem do atributo *type* de *param*. | X |

Tabela - Atributos do elemento output

# Permissões

# 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