INSTITUTO SUPERIOR DE CIÊNCIAS DO TRABALHO E DA EMPRESA

DEPARTAMENTO DE CIÊNCIAS E TECNOLOGIAS DA INFORMAÇÃO

DEFINIÇÃO DO MODELO CONCEPTUAL DE DADOS

ATRAVÉS DAS LINGUAGENS UML E ORM

Isabel Maria Cruz Valentim

Tese submetida como requisito parcial para obtenção do grau de

Mestre em Gestão de Sistemas de Informação

Orientador:

Prof. Doutor Pedro Nogueira Ramos

Setembro de 2008

Resumo

Esta dissertação tem por objectivo a comparação e avaliação de duas linguagens de modelação de dados, elas são o diagrama de classes do UML (Unified Modeling Language) e o ORM (Object-Role Modeling), o diagrama de classes por ser um diagrama vocacionado para a modelação de dados que suportam os sistemas de informação e o ORM por ser um método que pretende simplificar o processo no planeamento da construção e da estruturação do modelo de dados de um sistema de informação.

Ambas as linguagens utilizam uma notação própria que se pretende comparar e avaliar, de forma a verificar qual das linguagens representa e completa mais adequadamente o modelo conceptual de dados, no levantamento de todos os detalhes relevantes do domínio da aplicação que se pretende implementar.

Para melhor compreendermos estas linguagens é apresentada a notação e a semântica utilizada por cada uma delas, procedendo-se à sua aplicação através de um caso prático sobre um sistema de informação para a gestão de "Leilões On-line", onde é feita a definição do modelo conceptual de dados em ambas as linguagens e, posteriormente, é efectuada a sua comparação.

Com base na comparação é efectuada uma avaliação à qualidade semântica e à qualidade sintáctica das linguagens, de acordo com os critérios, métricas, valores e ponderações aplicáveis à análise da qualidade das linguagens de modelação de dados.

Palavras-chave: Modelação conceptual de dados, Linguagens de modelação de dados, UML, ORM, Qualidade semântica, Qualidade sintáctica

Abstract

This dissertation is aimed at the comparison and evaluation of two languages used in modeling data, they are the class diagram of the UML (Unified Modeling Language) and ORM (Object-Role Modeling), the diagram of classes to be a diagram designed for the modeling of data that support information systems and ORM to be a method that aims to simplify the process in planning the construction and structure of the data model of an information system.

Both languages use a notation itself to be compared and evaluated in order to check which language is most appropriate and complete the conceptual model of data in the capture of all relevant details in the domain of application to implement.

To better understand these languages is presented graphically the notation used in each one, making up for its implementation through a case study on an information system for the management of "On-line Auctions" which is made in the shaping of data both languages and, later, it made its comparison.

Based on a comparison is made to assessment of the quality semantic and the quality syntactic of the languages, according to the criteria, metrics, values and weights applied to the analysis of the quality of language modeling data.

Keywords: conceptual data modeling, languages of data modeling, UML, ORM, semantic quality, syntactic quality.

Agradecimentos

Agradeço a todos aqueles que me ajudaram à realização deste trabalho, em especial: Ao Professor Doutor Pedro Ramos, pelas suas sugestões e orientação;

À minha família, em especial aos meus pais e irmã, pelo apoio que sempre me deram;

Aos meus amigos e colegas pelo incentivo à execução deste trabalho.

Abreviaturas

CASE – Computer Assisted Software Engineering

DDL – Data Definition Language

ISO - International Standards Organization

ORM – Object Role Modeling

OCL – Object Constraint Language

OMG - Object Management Group

UML – Unified Modeling Language

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

A utilização de meios informáticos nas organizações exige, cada vez mais, a necessidade de descrever com rigor o modo como estas funcionam para que os sistemas de informação possam satisfazer plenamente as suas necessidades. Este é um requisito importante quer se venha a optar pela aquisição de uma aplicação informática existente no mercado ou por um desenvolvimento específico.

Os sistemas de informação tendem a ser cada vez mais flexíveis, mas muitos não estão preparados para satisfazer todas as necessidades de informação dos seus utilizadores, o que requer constantes definições do que se pretende de um sistema de informação, de forma a avaliar se este é capaz de responder a essas necessidades ou se requer adaptações.

Desde modo, torna-se necessário recorrer a linguagens que facilitem a comunicação entre aqueles que têm de lidar com os sistemas de informação: os actuais e os potenciais utilizadores, os analistas que avaliam se os sistemas de informação satisfazem essas necessidades e os programadores que desenvolvem as funcionalidades pretendidas.

Uma definição correcta do que se pretende de um sistema de informação passa, principalmente, pela qualidade do desenho do modelo conceptual de dados, onde é feita uma descrição a um alto nível de abstracção de como é guardada e estruturada a informação, tendo em consideração os requisitos do domínio a que está sujeita, possibilitando uma melhor compreensão daquilo que se pretende do sistema. É aqui que as linguagens de modelação são importantes pois permitem aos analistas representar a informação a um nível de abstracção, acima dos detalhes de implementação, de modo a dar enfoque ao modelo de negócio que está a ser representado. Estas linguagens têm também a vantagem de permitirem reduzir o tempo e os custos na fase de análise dos projectos.

1.1. Contexto

Actualmente, na indústria de informática existem linguagens próprias para representar os modelos conceptuais de dados, como o UML (Unified Modeling Language) e o ORM (Object Role Modeling), que ligadas a ferramentas CASE (Computer Aided Software Engineering) possibilitam a criação de automatismos no processo de construção dos

sistemas de informação, quer apoiando a modelação das várias constituintes no desenvolvimento dos sistemas de informação, como sejam os fluxos de informação e a caracterização das entidades envolvidas, quer gerando código de acordo com as linguagens de programação nelas incluídas.

Um dos factores de sucesso destas ferramentas está na notação e capacidade semântica das linguagens nelas incorporadas, que quanto mais válidas e completas, mais facilmente traduzem a realidade do domínio da aplicação que se pretende implementar.

O UML é uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objectos, adoptada pela OMG (Object Management Group) e tem-se tornado num padrão na indústria do desenvolvimento dos sistemas de informação. Esta linguagem recorre a um conjunto de diagramas que possibilitam a representação dos sistemas de informação sob diversas perspectivas de visualização. No contexto deste trabalho, apenas o diagrama de classes será abordado, uma vez que é aquele que está vocacionado para a modelação de dados que suporta os sistemas de informação.

A linguagem ORM não é tão usual como o UML na indústria do desenvolvimento de sistemas de informação, mas que se define como um método que pretende simplificar o processo de planeamento da construção e da estruturação do modelo de dados de um sistema de informação, através de uma linguagem gráfica e textual. Sobre ORM já existem alguns estudos, nomeadamente livros e sites dedicados à sua metodologia, assim como artigos que comparam ORM com UML. O ORM está também disponível na ferramenta "Visio for Enterprise Architects" da Microsoft.

1.2. Objectivo

As linguagens UML e ORM utilizam uma notação gráfica e textual na modelação de dados, de um qualquer sistema de informação, tendo cada uma as suas próprias características. Na modelação de dados é importante escolher uma linguagem que permita o levantamento de todos os detalhes relevantes do domínio da aplicação, e que permita uma interpretação simplificada do mesmo. Por esta razão torna-se necessário a escolha de

uma linguagem com notação e capacidade semântica capaz de gerar um modelo conceptual de dados completo, válido e simples.

Este trabalho de dissertação tem por objectivo comparar e avaliar as principais características as linguagens UML e ORM, por forma a analisar qual delas representa e completa mais adequadamente o modelo conceptual de dados, no levantamento de todos os detalhes relevantes do domínio da aplicação, de acordo com a semântica e a notação de cada uma das linguagens.

1.3. Metodologia

De modo a compreendermos estas linguagens é apresentado um caso prático sobre um sistema de informação para a gestão de "Leilões On-line" onde é feita a modelação de dados de acordo a notação e semântica de ambas as linguagens para, posteriormente, efectuarmos a sua comparação.

Com base na comparação é efectuada uma avaliação das linguagens onde se determina a qualidade semântica e a qualidade sintáctica das linguagens de acordo com os critérios, métricas, valores e ponderações aplicáveis à análise da qualidade das linguagens de modelação de dados.

Os critérios aqui utilizados para a avaliação das linguagens foram os definidos por Patrício (2003), baseados nos autores Lindland, Sindre e Solvberg (1994). Para a construção da grelha de avaliação foi aplicado o método de valoração proposto, por Sousa (1998), no modelo de avaliação PHIMA, que será aplicado às ponderações atribuídas.

Com esta avaliação pretendemos analisar a qualidade semântica e sintáctica das linguagens de modelação de dados, de modo a apuráramos qual das linguagens apresenta uma notação gráfica mais completa e expressiva na representação na definição do modelo conceptual de dados, para que possam ser gerados modelos com o máximo de automatismo e rigor sobre o domínio da aplicação.

1.4. Estrutura da Dissertação

Esta dissertação estrutura-se, por um lado, na apresentação dos aspectos teóricos relevantes para a compreensão da modelação de dados e, por outro, nos aspectos práticos onde se aplicam os aspectos teóricos a um caso prático de modelação de dados, através das linguagens abordadas, procedendo-se à comparação da notação e semântica das linguagens consoante os seus formalismos e, no final, efectua-se a avaliação das linguagens.

Assim sendo, o capítulo que se segue começa por introduzir os conceitos relevantes sobre a modelação de dados e a sua importância no desenvolvimento dos sistemas de informação, assim, como a importância de existirem linguagens e ferramentas para definirem os modelos conceptuais de dados.

O terceiro capítulo é dedicado à linguagem ORM, descrevendo-se as suas principais características, assim como a sua notação e semântica utilizada na definição do modelo conceptual de dados.

O quarto capítulo é dedicado à linguagem UML, nomeadamente, ao Diagrama de Classes que é aquele onde recai o âmbito deste trabalho, descrevendo-se a sua notação e semântica utilizada na definição do modelo conceptual de dados.

No quinto capítulo é apresentado um caso prático sobre a definição do modelo conceptual de dados em ambas as linguagens, seguindo-se o capítulo sexto relativo à comparação da notação e da semântica de acordo com os formalismos gráficos apresentados pelas linguagens.

No capítulo sétimo efectua-se a avaliação das linguagens de acordo com o quadro de ponderações e valorações definido.

Por fim, no último capítulo, apresentam-se as conclusões e trabalhos futuros.

2. Modelação de dados

2.1. Modelo e modelação

Um modelo consiste na interpretação de um dado domínio do problema, ou seja, de um fragmento do mundo real sobre o qual as tarefas de modelação vão recair, segundo uma determinada estrutura de conceitos.

A representação de um modelo é feita através da utilização de uma determinada linguagem, a qual pode ser formal ou informal, textual ou gráfica. Quando a representação é gráfica, designa-se por diagrama.

De acordo com Silva (2001), um modelo é sempre uma interpretação simplificada da realidade. Desde sempre que a ciência em geral procura representar "a realidade" através de modelos mais ou menos correctos, mais ou menos abrangentes, mais ou menos detalhados, sendo preferível ter um mau modelo que nenhum modelo na descrição de qualquer sistema.

Ainda de acordo com o autor supra citado, a modelação é a arte e ciência de criar modelos de uma determinada realidade. É uma técnica bem aceite e adoptada pela generalidade das disciplinas científicas, que permite a partilha de conhecimento entre diferentes grupos de intervenientes (técnicos e não técnicos), facilitando e promovendo a comunicação entre todos. A modelação permite ter uma gestão mais eficaz e eficiente das equipas de projecto, ao dar uma visão mais adequada sobre os vários produtos a desenvolver, permitindo ainda que as previsões de custos e prazos sejam efectuadas de acordo com critérios mais realistas, o que contribui para a minimização dos riscos associados.

Segundo Booch (1999), a engenharia informática necessita de adoptar notações e linguagens de representação dos seus modelos, de modo a obter benefícios através da modelação, entre os quais se destacam:

- Os modelos ajudam a visualizar um sistema, quer seja a sua situação no passado, no presente ou no futuro;
- Os modelos permitem especificar a estrutura ou o comportamento de um sistema;

— Os modelos permitem controlar e guiar o processo de construção do sistema;

— Os modelos documentam as decisões tomadas.

2.2. Importância da modelação de dados

A modelação de dados é uma técnica que permite organizar e documentar os dados de um sistema de informação. Os dados são utilizados por um sistema que os transforma em informação, sendo esta informação vital para as organizações que a partilham pelos vários sectores. Como tal, os dados devem estar organizados, de modo a serem flexíveis, de maneira a que se possam adaptar para responder a necessidades, muitas vezes, imprevisíveis – obter dados com estas características é o principal objectivo da modelação de dados. Além disso, a modelação de dados é importante porque permite aos analistas de um sistema chegar rapidamente a um consenso com os utilizadores, sobre a terminologia e as regras do negócio.

A construção de um modelo de dados passa por um processo de abstracção, onde apenas são identificados os aspectos importantes de um sistema ignorando todos os outros. No desenho de bases de dados significa que se deve concentrar primeiro nas entidades ou objectos e nas suas características e relações, antes de se decidir sobre a forma como estes devem ser implementados, obtendo-se um modelo que é independente do sistema sobre o qual a base de dados vai ser implementada.

Um modelo de dados é uma representação dos dados de um sistema assim como uma planta é uma representação de uma casa. Tal como é pouco provável que se possa construir uma boa casa sem ter sido desenhada uma boa planta, também é pouco provável que se possa implementar um bom sistema de bases de dados sem realizar primeiro um bom modelo de dados. Um bom modelo de dados deve ser simples, não redundante e flexível.

2.3. Níveis de modelação de dados

Um modelo de dados pretende descrever os dados, as relações entre os dados, a semântica dos dados e as restrições dos dados, através de uma representação abstracta e simplificada, que permite a validação e explicação das características do mundo real. Por esta razão, um modelo de dados assume um papel muito importante na compreensão de um sistema de informação durante a fase de percepção.

Durante o ciclo de desenvolvimento de um sistema de informação, os modelos de dados passam por três níveis distintos:

1. Modelo conceptual de dados;
2. Modelo lógico de dados;
3. Modelo físico de dados.

O primeiro nível é o modelo conceptual de dados, onde os objectos e as suas características e relacionamentos têm uma representação real do domínio da aplicação, independentemente das limitações impostas pelas tecnologias, técnicas de implementação ou dispositivos físicos. É utilizado ao nível da conversação, entendimento e validação de conceitos do mundo real.

Algumas das vantagens do modelo conceptual de dados são:

- O mesmo modelo permite derivar diferentes estruturas de implementação;
- O processo de modelação não fica restringindo aos recursos disponibilizados por cada tecnologia de implementação;
- Representa um esforço inferior, simplificando a tarefa de modelação;
- Aumenta a aplicabilidade da modelação de dados a outras finalidades, como instrumento de comunicação de conceitos, especificações e regras.

O segundo nível é o modelo lógico de dados, onde os objectos e as suas características e relacionamentos têm uma representação de acordo com as regras de implementação e limitações impostas por algum tipo de tecnologia, como por exemplo o modelo relacional e

o modelo orientado a objectos. A sua representação é independente dos dispositivos ou meios de armazenamento físico das estruturas de dados, resulta da aplicação de regras de derivação sobre o modelo conceptual e envolve a representação dos objectos observados e dos conceitos necessários à implementação.

O terceiro nível é o modelo físico de dados, onde a representação dos objectos é feita de acordo com o armazenamento físico dos dados e com a implementação física das ocorrências das entidades e seus relacionamentos, em que o conhecimento do modo físico de implementação das estruturas de dados passa a ser um ponto básico para o domínio deste tipo de modelo, que varia consoante o modo de implementação física das características e recursos necessários para o armazenamento e manipulação das estruturas de dados.

2.4. Linguagens e ferramentas de modelação de dados

Uma das primeiras linguagens de modelação de dados a aparecer designava-se como o modelo E-R (Entidade – Relacionamento), que se tornou dominante como ferramenta para o desenho de bases de dados, por ser uma linguagem simples e ao mesmo tempo poderosa. No entanto, sendo suficiente para o desenho da maioria dos sistemas de bases de dados, apresentava algumas limitações quando utilizada no desenho de sistemas mais complexos. Assim sendo, têm vindo a ser desenvolvidos novos conceitos, envolvendo aspectos semânticos, incorporados no modelo E-R original com sucesso.

Surgem, assim, as linguagens como o UML e o ORM que incorporadas em ferramentas, designadas por ferramentas CASE, facilitam a geração dos modelos de dados com o máximo de automatismos e rigor.

Ferramentas CASE são ferramentas que auxiliam as actividades desenvolvimento de software, desde análise de requisitos e modelação de dados até programação e testes. Podem ser consideradas como ferramentas automatizadas, que têm como objectivo auxiliar os analistas e programadores nas várias fases do ciclo de desenvolvimento de software.

A título de exemplo, apresentam-se as seguintes ferramentas CASE:

Ferramenta	Empresa
Visio	Microsoft
Rational Rose	Rational
Paradigm Plus, ErWin	Computer Associates
PowerDesigner	Sybase
Designer, Developer	Oracle
Visual UML	Visual Object Modelers

Tabela 1 – Ferramentas CASE

No âmbito deste trabalho, foram utilizadas as ferramentas Microsoft PowerDesigner 11 da Sybase eVisio Entreprise Arcthitects da para o estudo da definição do modelo conceptual de dados, assim como para geração automática da linguagem de definição de dados (DDL), apresentada nos Anexos IV e V, através das linguagens UML e ORM respectivamente.

3. Object-Role Modeling (ORM)

Segundo Halpin (2001), ORM é um método que pretende simplificar o processo no planeamento da construção e da estruturação do modelo de dados de um sistema de informação utilizando uma linguagem gráfica e textual, que recorre a diagramas e esquemas que podem ser preenchidos e verbalizados com exemplos.

De acordo com Halpin (2006), o ORM teve origem na Europa, nos anos 70, onde era conhecido como NIAM (Natural-Language Information Analysis Method) ou por FCO-IM (Fully Communication Oriented Information Modeling). Actualmente, trata-se de uma extensão destas linguagens, que são orientadas aos factos e, apenas, consideram os factos relevantes para serem expressos em factos simples ou elementares.

O ORM pretende proporcionar um entendimento claro e inequívoco de um sistema de informação ao seu nível mais abstracto, aos técnicos e não técnicos, através de um modelo que seja facilmente compreendido e validado. ORM é assim chamado porque representa o mundo real em termos de objectos (entidades e valores) que desempenham papéis entre si (relacionamentos).

3.1. Método ORM

O ORM apresenta um método para a definição do modelo conceptual de dados de um sistema de informação, de forma a expressar todos os exemplos de informação, em factos elementares, e a identificar todos os objectos e relacionamentos a serem considerados na definição do modelo.

Parte do pressuposto que cada modelo deve ter uma dimensão suficiente para ser tratado como uma unidade. No caso de se tratar de um grande sistema de informação, este deve ser dividido em várias subsecções e dada prioridade a cada uma delas. A cada subsecção deve-se aplicar o método "Conceptual Schema Design Procedure" (CSDP). Este método está dividido em sete etapas, que são as seguintes:

1. Transformar e validar os exemplos de informação relevantes em factos elementares;
2. Desenhar os factos elementares e efectuar validações utilizando verbalizações e exemplos;
3. Validar cada entidade que possa vir a ser relacionada;
4. Adicionar restrições de unicidade e validar as suas relações;
5. Adicionar restrições de obrigatoriedade e verificar a existência de derivações lógicas;
6. Adicionar restrições de valor, de igualdade e de subconjuntos;
7. Adicionar outras restrições e efectuar as últimas validações.

No final de aplicado a todas as subsecções do modelo, estas serão integradas num modelo conceptual global.

Conforme referido, o método ORM utiliza uma linguagem gráfica e textual para expressar e representar o modelo conceptual de dados. No âmbito deste trabalho, pretende-se conhecer a sua notação e semântica utilizada por esta linguagem, mas para dar um melhor entendimento é necessário ter presente alguns conceitos utilizados pelo ORM, tais como: facto elementar, tipos de objectos e predicados.

3.2. Facto elementar

Um facto elementar é uma afirmação ou proposição acerca do domínio da aplicação. No ORM devem ser identificados todos os factos com interesse e relevantes para o sistema de informação. Os factos não são mais do que relatos que podem ser vistos como um conjunto de objectos que desempenham papéis entre si.

O adjectivo “elementar” significa que o facto não pode ser dividido em pequenas unidades de informação porque, no conjunto, não fornecem a mesma informação. Um facto elementar não faz uso de operadores lógicos (e, ou, negação, se) nem de quantificadores lógicos (todos, alguns, etc.). Um exemplo de facto elementar seria a seguinte afirmação: “A pessoa com o nome Ana trabalha para o departamento de Informática”.

3.3. Tipos de Objectos: Entidade e Valor

Os factos elementares são afirmações sobre objectos que desempenham papéis ou funções num determinado universo e esses objectos são especificados por entidades e valores.

Uma entidade é referenciada, conceptualmente, como sendo uma descrição decisiva do objecto e um valor como sendo uma referência de uma entidade, que pode ser do tipo alfanumérico ou numérico. Considerando sempre que as entidades e os valores existem mesmo no mundo real.

Uma entidade é ainda composta pelo modo de referência que representa o valor que identifica, univocamente, cada uma das suas instâncias. Assim, a designação de uma entidade envolve três componentes: o nome da entidade, o modo de referência e os valores.

Por exemplo, no facto elementar anteriormente referido, "A pessoa com o nome Ana trabalha para o departamento de Informática", são identificadas duas entidades e dois valores, a entidade "*Pessoa*" com o valor "*Nome* " e a entidade "*Departamento*" com o valor "*Nome do Departamento*".

3.4. Predicados

Para especificar os papéis que os objectos desempenham são utilizados os predicados. O predicado indica o papel que o objecto desempenha, isolado ou relacionado com outros objectos, ou seja, um predicado pode envolver um ou mais objectos, tratando-se de predicados unários, binários, ternários, …, *n*-ários.

Em geral, um predicado *n*-ário é um facto com *n* objectos. O valor de *n* é a ariedade ou o grau do predicado. Enquanto, as entidades e o modo de referência são representados pelo nome (substantivo), o predicado é representado pelo verbo (acção). Um predicado pode ser lido da esquerda para a direita como da direita para esquerda, sendo neste último caso necessária a sua leitura invertida.

Por exemplo, no facto elementar anteriormente referido, "A pessoa com o nome Ana trabalha para o departamento de Informática", o predicado é o verbo "…*trabalha*…" que

também pode ser lido como de forma invertida "No departamento de Informática *trabalha* a pessoa com o nome Ana".

Resumindo, a definição do modelo conceptual de dados, através do ORM, passa pela identificação dos exemplos de informação relevante, descrevê-los em factos elementares e identificar os objectos e os predicados para que possam ser representados graficamente pela linguagem.

3.5. Notação e semântica

3.5.1. Entidade e Valor

Uma entidade é representada por um eclipse com uma linha sólida (Figura 1). O nome da entidade é colocado dentro do eclipse e a identificação de como é referenciada a entidade – o modo de referência – fica abaixo do nome da entidade.

Figura 1 – ORM – exemplo de Entidade

Um valor é equivalente a ter um atributo relacionado a uma entidade. É representado por um eclipse com uma linha a tracejado e no seu interior é mencionado o nome do objecto do tipo valor (Figura 2).

Figura 2 - ORM exemplo de Valor

3.5.2. Subtipo

O subtipo indica que uma entidade herda propriedades e características de outra entidade, a sua representação é feita com a ligação entre as duas entidades através de uma seta (⟶) apontada para a entidade que herda as propriedades, conforme ilustrado na Figura 3.

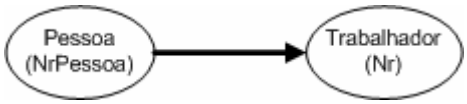


Figura 3 – ORM exemplo de Subtipo

No exemplo da Figura 3, a entidade Trabalhador vai herdar propriedades da entidade Pessoa, ou seja, a entidade Trabalhador não necessita de valores para os dados pessoais do trabalhador pois herda-os da entidade Pessoa.

3.5.3. Predicados

Os predicados são representados por uma caixa ou por uma sequência de caixas, consoante o grau dos predicados, ou seja, depende do número de relações ou regras entre os objectos. Segundo Halpin (2001), os predicados mais comuns são os binários que indicam a relação entre dois objectos (Figura 4).

Figura 4 – ORM exemplo de predicado binário

No exemplo da Figura 4, a entidade Trabalhador tem uma relação com a entidade Departamento, em que as duas entidades se relacionam indicando que um trabalhador trabalha para um departamento.

Na tabela 2 estão representados os vários tipos de relações entre os objectos, que vão desde a relação unária até às relações *n*-árias consoante o número de objectos que se está a relacionar.

Grau/várias	Notação	Exemplo
Unário	…	"A pessoa com o nome Ana trabalha"
Binário	… …	"A pessoa com o nome Ana trabalha no departamento de Informática"
Ternário	… … … ..	"A pessoa com o nome Ana trabalha no departamento de Informática com a categoria de programadora"
Quaternário	… … … …	"A pessoa com o nome Ana trabalha no departamento de Informática com a categoria de programadora no horário da manhã"
Quinquenário	…	…

Tabela 2 – Predicados

3.5.4. Objecto aninhado (Nested object)

Um objecto aninhado ocorre quando um objecto participa numa relação entre dois ou mais objectos. A sua representação faz-se colocando um eclipse em volta do predicado, conforme apresentado na Figura 5.

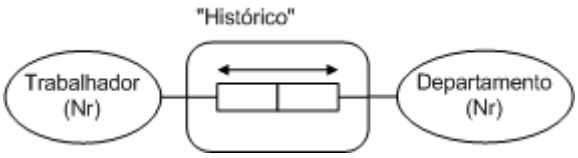


Figura 5 – ORM exemplo Objecto aninhado

No exemplo da Figura 5, na relação entre as entidades Trabalhador e Departamento existe a participação de uma outra entidade, a entidade Histórico, que permite ter uma relação onde é possível manter um histórico do trabalhador pelos vários departamentos, onde é colocado ao longo do tempo.

3.5.5. Restrições

Uma restrição é uma regra, à qual os objectos devem obedecer de acordo com os factos elementares identificados. No ORM, as restrições são consideradas internas ou externas –

internas quando aplicadas a um só predicado (Figura 6) e externas quando aplicadas a dois ou mais predicados (Figura 7).

Figura 6 - Restrição interna (regra num predicado)

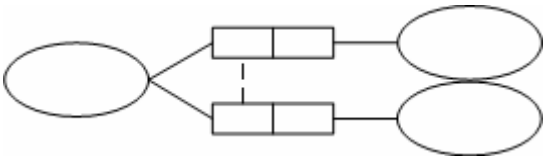


Figura 7 - Restrição externa (regra entre dois predicados)

3.5.5.1. Restrições de Unicidade (Uniqueness constraints)

A restrição de unicidade garante que uma instância, de um tipo de objecto, é única em relação a todas as outras instâncias, desse mesmo tipo de objecto. Por outras palavras, a restrição de unicidade evita a duplicação de informação, garantido a sua singularidade. Existem vários tipos de representação para as restrições de unicidade, consoante ao grau do predicado. A sua representação é feita colocando o símbolo ⟷ sobre as regras do predicado, as mais comuns são apresentadas na tabela 3.

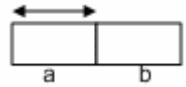
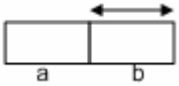
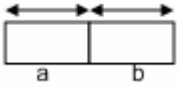
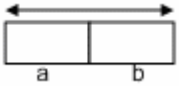
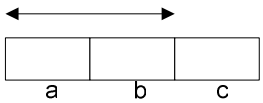
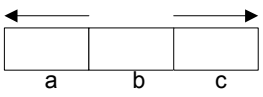
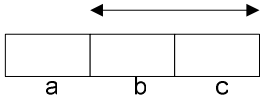
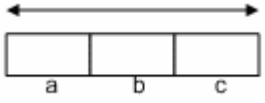
Restrição	Verbalização
	Para cada "A" existe um "B" O mesmo que ter uma relação de "Um para Muitos"
	Para cada "B" existe um "A" O mesmo que ter uma relação de "Muitos para Um"
	Para um único "A" corresponde um único "B" O mesmo que ter uma relação de "Um para Um"
	"A"e "B" são únicos O mesmo que ter uma relação de "Muitos para Muitos"
	"A" e "B" são únicos
	"A" e "C" são únicos
	"B" e "C" são únicos
	"A", "B" e "C" são únicos

Tabela 3 – Exemplos de restrição de unicidade

3.5.5.2. Restrição Obrigatória (Mandatory role constraints)

A restrição obrigatória garante que cada instância de um tipo de objecto só ocorre quando respeitada determinada regra, ou seja, para um objecto (A) ser instanciado terá de existir obrigatoriamente uma relação com objecto (B). A sua representação é feita com o símbolo • do lado do objecto que deverá respeitar a regra, conforme apresentado na Figura 8.

Figura 8 – Restrição de Obrigatoriedade

3.5.5.3. Restrições de Frequência (Frequency Constraints)

A restrição de frequência garante o número vezes que pode ocorrer um determinado valor, ou conjunto de valores, num determinado predicado ou relação. Esta restrição é representada colocando o número de ocorrências por cima da regra do objecto, conforme apresentado na Figura 9.

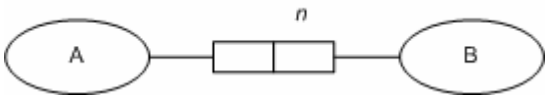


Figura 9 – Restrição de Frequência

3.5.5.4. Restrições de Igualdade (Equality contraints)

A restrição de igualdade garante que quando uma determinada regra é respeitada uma outra também tem o de ser. Trata-se de uma restrição condicional, quando uma regra se verifica outra também tem de se verificar para que se possa ser instanciar determinado objecto. Esta relação é representada pelo símbolo – – –(=)– – – ou pelo símbolo ←– –→ entre os predicados dos objectos, conforme apresentado na Figura 10.

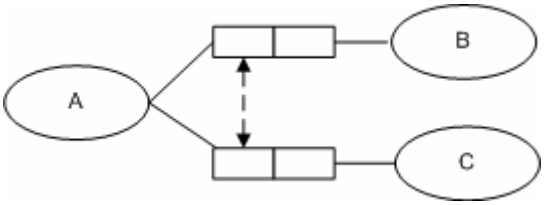


Figura 10 - Restrição de Igualdade

3.5.5.5. Restrições de Subconjuntos (Subset constraints)

Esta restrição exige que a população de determinada regra tenha que ser um subconjunto da população de outra regra. Esta relação é representada pelo símbolo - - →, ou pelo símbolo - ⊂ →, que é colocado entre os predicados dos objectos conforme apresentado na Figura 11.

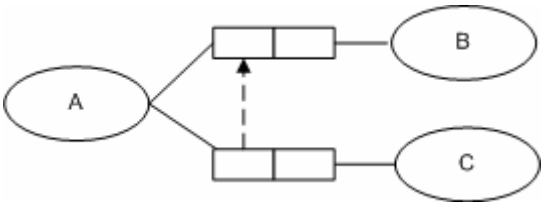


Figura 11 - Restrição de Subconjunto

3.5.5.6. Restrições de Exclusão (Exclusion constraints)

A restrição de exclusão exige que a população de uma determinada regra seja a disjunção da população de outra regra. Esta relação é representada pelo símbolo⊗, mas no caso de ter que existir obrigatoriamente uma das regras, o símbolo passa a ser o seguinte: ⊙. O símbolo é colocado entre os predicados conforme se ilustra na Figura 12.

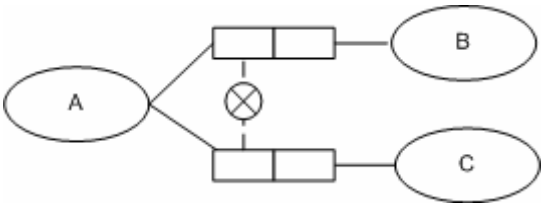


Figura 12 - Restrição de Exclusão

3.5.5.7. Restrições em Anel (Ring Constraints)

Esta restrição exige que uma regra seja aplicada sobre o mesmo objecto. A sua representação é feita ligando o predicado ao mesmo objecto e indicando o tipo de relação que se está a aplicar, conforme apresentado na Figura 13.

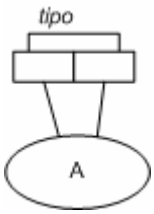


Figura 13 - Restrição em anel

Existem vários tipos de restrições em anel que podem ser aplicados ao objecto, conforme se ilustram na tabela 4.

Tipos de restrições em anel	
Irreflexivo Não pode existir a combinação (a, a).	
Anti Simétrico Se existe uma combinação (a, b) logo pode existir uma combinação (a, a), mas não pode existir uma combinação (b, a).	
Simétrico Se existe uma combinação (a, b) logo pode existir uma combinação (b, a).	
Intransitivo Se existe uma combinação (a, b) e (b, c) logo não pode existir uma combinação (a, c).	
Acíclico Se existir uma combinação (a, b) e (b, c) logo não pode existir uma combinação (c, a).	
Assimétrico Se existe uma combinação (a, b) logo não pode existir uma combinação (b, a).	

Tabela 4- Tipos de restrições em anel

3.5.5.8. Restrições de Valor (Value Constraints)

As restrições de valor indicam quais os valores, ou conjunto de valores, que podem ser considerados para um objecto tipo valor. Esta regra é representada colocando entre {} os valores, ou intervalos de valores, que podem ser assumidos pelo objecto, conforme apresentado na Figura 14.

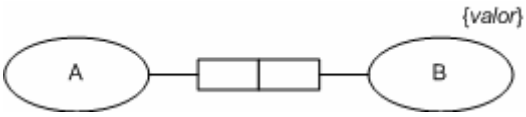


Figura 14 – Restrição de Valor

3.5.5.9. Restrição de Índice (Index constraints)

A restrição de indexe não é propriamente uma restrição. Um indexe é aplicado para aumentar o desempenho de uma base de dados no acesso à informação. Esta regra é representada pelo símbolo Ⓘ sobre a regra onde se pretende aplicar o indexe, conforme apresentado na Figura 15.

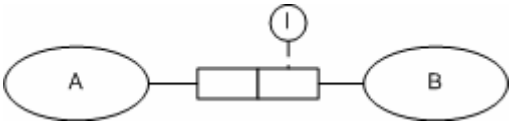


Figura 15 – Restrição de Indexe

4. Unified Modeling Language (UML)

O UML (Unified Modelling Language) é uma linguagem constituída por vários diagramas, utilizável para construção, especificação, visualização e documentação de sistemas de software. O UML surge em 1997, na sequência de um esforço de unificação de três das principais linguagens de modelação orientadas por objectos – OMT, BOOCH e OOSE. Seguidamente, adquiriu o estatuto de norma no âmbito da OMG e da ISO[1], tendo vindo a ser adoptado pela indústria e academia em todo o mundo.

De acordo com a OMG, o UML é uma linguagem com as seguintes particularidades:

- Semântica e notação para tratar um grande número de tópicos actuais de modelação;
- Semântica para tratar tópicos de modelação futura, relacionados em particular com a tecnologia de componentes, computação distribuída, *frameworks* e Internet;
- Mecanismo de extensão, de modo a permitir que futuras aproximações e notações de modelação possam continuar a ser desenvolvidas sobre o UML;
- Semântica e sintaxe para facilitar a troca de modelos entre ferramentas distintas.

4.1. Tipos de Diagramas

Os diagramas são conceitos que traduzem a possibilidade de agrupar elementos básicos e as suas relações de uma forma lógica ou de uma forma estrutural. Existem diferentes tipos de diagramas em UML. Em cada tipo de diagrama é usado um subconjunto de elementos básicos, com diferentes tipos de relações que façam sentido em existir.

Por exemplo, um diagrama de classes UML é composto por um conjunto de ícones representantes de classes em simultâneo (e opcionalmente), com a representação explícita das suas relações.

[1] International Standards Organization

O UML define diferentes tipos de diagramas, cuja utilização e aplicação permitem dar visões complementares de um sistema de software.

— Diagramas de casos de utilização que representam a visão do sistema na perspectiva do seu utilizador;

— Diagrama de classes que permitem especificar a estrutura estática de um sistema, segundo a abordagem orientada por objectos;

— Diagrama de interacção entre objectos (diagramas de sequência e diagramas de colaboração), diagramas de transição de estados e diagramas de actividades, que permitem especificar a dinâmica ou o comportamento de um sistema, segundo a abordagem orientada por objectos;

— Diagramas de componentes e diagramas de instalação, que permitem dar uma visão da disposição dos componentes físicos (software e hardware) de um sistema.

4.2. Diagrama de Classes

No âmbito deste trabalho, apenas se descreve o diagrama de classes, que, de acordo com Booch (1999), é aquele que permite a modelação do esquema conceptual de uma estrutura de dados, quer seja uma base de dados relacional ou uma base de dados orientada por objectos, por forma a guardar a informação do sistema.

O diagrama de classes consiste, então, na descrição formal da estrutura de objectos de um sistema de informação. Para cada objecto descreve a sua identidade, os seus relacionamentos com os outros objectos, os seus atributos e as suas operações.

A criação de um diagrama de classes resulta de um processo de abstracção, através do qual se identificam os objectos (entidades e conceitos) relevantes ao contexto que se pretende modelar, se descreve as características comuns em termos de propriedades (atributos) e de comportamentos (operações). A essa descrição genérica designa-se por classe.

As classes descrevem objectos com atributos e operações comuns, e servem dois propósitos: permitem compreender o mundo real naquilo que é relevante para o sistema de

informação que se pretende desenvolver e fornecem uma base prática para a implementação de sistema.

Um diagrama de classes é composto pelos seguintes elementos abstractos de modelação:

- Classes de objectos;
- Relações de Associação, Generalização e Dependência;
- Multiplicidade.

4.3. Notação e semântica

4.3.1. Objecto

Um objecto reflecte, em geral, uma entidade do mundo real e apresenta um estado e um comportamento próprio. Os objectos interactuam entre si por troca de mensagens. Uma classe consiste numa estrutura que permite criar objectos semelhantes, que apresentem estado e comportamento semelhante. Deste modo, diz-se que uma classe é uma fábrica de objectos e que um objecto é uma instância de uma classe.

4.3.2. Classe

Uma classe é a descrição de um conjunto de objectos que partilham os mesmos atributos, operações, relações e a mesma semântica. Uma classe corresponde a algo tangível ou a uma abstracção conceptual existente no domínio do utilizador ou no domínio do informático.

Uma classe é representada em UML por um rectângulo (Figura 16) com uma, duas ou três secções. Na primeira secção apresenta-se o nome da classe, na segunda a lista de atributos e na terceira a lista de métodos.

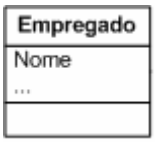


Figura 16 – UML exemplo de Classe

4.3.3. Associações

No diagrama de classes, as associações representam as relações entre os objectos. As associações são caracterizadas por possuir um nome, e, quando necessário podem também incluir o papel que os objectos têm na relação, a Figura 17 representa uma associação binária entre a classe "Empregado" e a classe "Departamento".

Figura 17 – UML exemplo de Associação binária

Uma classe pode possuir uma associação consigo própria, o que significa que um objecto da classe se relaciona com um ou vários objectos da mesma classe. Tipicamente, esta relação surge em situações de hierarquia como, por exemplo, o responsável de um conjunto de empregados é também um empregado, conforme se ilustra na Figura 18.

Figura 18 – UML exemplo de Associação com a própria classe

4.3.4. Multiplicidade

As associações são também caracterizadas por possuir uma multiplicidade, que indica quantos objectos participam na relação. A multiplicidade pode assumir muitas formas, mas as mais comuns são:

0..1 - Opcional.
1..1 - Obrigatório existir um objecto, frequentemente representado por apenas 1.
1..10 - Um valor entre o intervalo estabelecido, neste caso de um a dez.
0..* - Zero ou infinitos objectos da classe, também representado por apenas *.
1..* - Um ou infinitos objectos da classe.

É possível efectuar várias combinações de multiplicidade numa associação. Por exemplo, a Figura 19 representa a relação "Um para Muitos" entre a Classe A e a Classe B, que significa que um objecto da Classe B está associado a um só objecto da Classe A e que um objecto da Classe A pode estar associado ou não (opcional) a muitos objectos da Classe B.

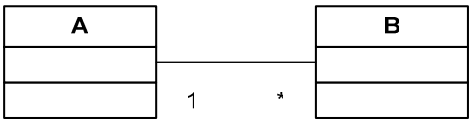


Figura 19 – Multiplicidade de um para muitos

4.3.5. Classes Associativas

Este tipo de classes surge da necessidade de reforçar o detalhe de informação de uma associação e é representado conforme se ilustra na Figura 20.

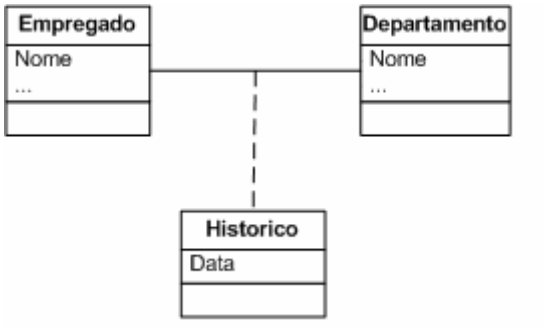


Figura 20 – UML Exemplo de Classe associativa

Uma classe associativa só existe em resultado da relação entre duas classes, sendo que por si só não terá significado. Normalmente, as classes associativas surgem nas relações de "Muitos para Muitos" e o nome da classe é dado pelo nome da associação.

4.3.6. Agregação e Composição

No UML existe o conceito de agregação e de composição, que são casos especiais das associações e que são utilizadas em situações especiais.

Uma agregação é utilizada quando se pretende destacar que um objecto consiste na agregação de um conjunto de outros objectos, graficamente é representada por um losango

colocado junto da classe que representa os objectos que agregam os objectos da classe oposta, conforme se ilustra na Figura 21, onde se representa que uma Empresa é constituída por um conjunto de Departamentos.

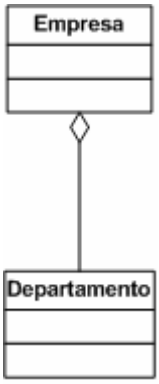


Figura 21 – Agregação

Uma composição é utilizada para destacar uma noção de um todo composto por partes, ou seja, de objecto composto por objectos. A noção de composição é mais forte do que a agregação porque, no caso da composição, assume que os objectos utilizados, por si só, não se distinguem dos restantes. Os objectos só podem ser referidos no contexto do objecto que compõem.

Graficamente é representada por um losango preenchido colocado junto da classe que representa os objectos que agregam os objectos da classe oposta, conforme se ilustra na Figura 22 onde se representa que uma factura é composta por um conjunto de linhas e que a linha da factura só pode ser referida (distinguida das restantes) se for indicada a factura correspondente.

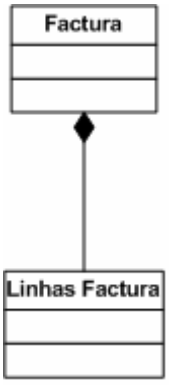


Figura 22 – Composição

4.3.7. Generalização

A generalização é um caso especial no diagrama de classes, que demonstra a noção de super-classe e subclasse na perspectiva de uma relação "pai e filho". A Figura 23 representa uma generalização entre as classes “Pessoa”e “Empregado”, onde a classe “Empregado” herda características da classe “Pessoa”.

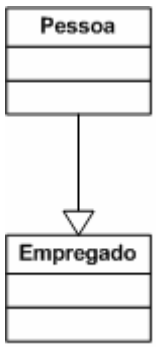


Figura 23 – UML exemplo de Generalização

O conceito de herança está presente, pois as subclasses ("filhos") herdam da super-classe ("pai") a estrutura em termos de atributos e operações.

4.3.8. Dependência

De acordo com Silva (2001), uma relação de dependência indica que a alteração na especificação de um elemento pode afectar outro elemento que a usa, mas não necessariamente o oposto. A dependência é representada em UML através de uma linha dirigida a tracejado (Figura 24). No contexto de classes, usam-se dependências para ilustrar que uma classe usa outra classe como argumento na assinatura de uma das suas operações ou como tipo na definição dos seus atributos. Mas por motivos de simplicidade e clareza não se explícita este tipo de relações nos diagramas de classes, já que esse tipo de dependência encontra-se especificado implicitamente

Figura 24 - UML - Dependência

4.3.9. Restrições

No UML existe representação gráfica para as restrições de subconjunto e de exclusão, que são a aplicadas às associações no diagrama de classes. Nas figuras abaixo são ilustradas as representações das respectivas restrições.

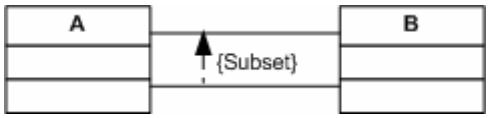


Figura 25 – UML Restrição Subconjunto

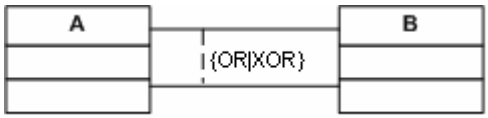


Figura 26 – UML Restrição Exclusão

A restrição de subconjunto (Figura 25) implica que a selecção de uma associação é um subconjunto de outra associação da mesma classe.

Por exemplo, para se especificar que um empregado para ser gestor de um departamento tem também de ser, necessariamente, membro desse departamento é utilizada a restrição de subconjunto (SUBSET), que se representa conforme o ilustrado na Figura 27.

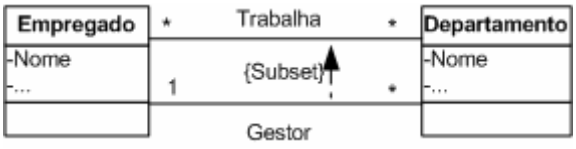


Figura 27 - UML - Exemplo de restrição de subconjunto

A restrição de exclusão (Figura 26) implica na selecção exclusiva (OR) ou mutuamente exclusiva (XOR) entre duas ou mais associações existentes em uma classe.

Por exemplo, para se especificar que uma conta bancária pertence obrigatoriamente a uma pessoa colectiva ou a uma pessoa individual utiliza-se a restrição exclusa (XOR), que se representa conforme ilustrado na Figura 28

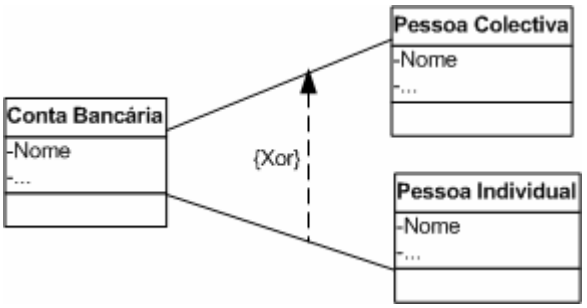


Figura 28 – UML - Exemplo de restrição de exclusão

De acordo com o autor Nunes (2007), para além destas restrições, também, é possível ter restrições para os valores dos atributos das classes, que quando não exigem uma grande precisão podem ser indicadas através de texto livre entre chavetas ou colocando uma nota junto dos elementos, a que dizem respeito, ou ligadas a eles através de uma linha a tracejado.

Mas quando as restrições exigem mais precisão é, então, utilizada a linguagem OCL (Object Constraint Language) que é uma linguagem para especificação formal de restrições que faz parte integrante do UML.

A Linguagem de Restrição de Objectos, abreviado OCL, é uma linguagem precisa, textual e formal utilizada para descrever regras através expressões matemáticas simples, nos diagramas UML. As expressões OCL garantem a não existência de interpretações ambíguas, para as mesmas restrições, e têm a vantagem de não exigirem um forte conhecimento matemático para serem utilizadas correctamente.

Uma restrição OCL é indicada por uma nota posicionada junto aos elementos a que diz respeito, ou a eles ligada por linhas a tracejado e sem setas. A Figura 29 ilustra a representação de uma expressão OCL.

Figura 29 – Restrição OCL

As expressões OCL são utilizadas para definir condições invariantes nas classes representadas e, também, são utilizadas para especificar as pré-condições e pós-condições em operações aplicadas às classes. Ou seja, existem três tipos de expressões no OCL:

— Invariantes, são condições que os objectos devem respeitar durante toda sua existência no sistema;

— Pré-Condições são declarações que reflectem o estado no qual se deve encontrar o sistema, para que as operações sobre os objectos possam ser executadas;

— Pós-Condições são declarações que apresentam o estado do sistema após a execução de uma determinada operação sobre um objecto.

Por exemplo, quando se pretende referir que a data de nascimento de uma pessoa deve ser superior a "01-01-1990" utiliza-se uma expressão OCL para representar essa restrição, conforme ilustrado na Figura 30

Figura 30 – UML - Exemplo de Expressão OCL

5. Caso Prático "Leilão on-line"

O caso prático apresentado diz respeito à informação de suporte a uma aplicação para disponibilização de artigos através de leilões on-line. Sobre cada artigo leiloado, para além da sua descrição e categoria (Música, Mobiliário, etc.), é necessário conhecer a base de licitação (valor a partir do qual são efectuadas as licitações), o seu valor mínimo de venda (valor a partir do qual o vendedor aceita vender o artigo a quem efectuou a maior licitação) e a data e hora a partir da qual não são possíveis mais licitações. Sobre as licitações é necessário conhecer a data, hora, valor licitado e identificação do licitador. Os artigos leiloados podem ser usados ou novos, no caso de usados é necessário conhecer o número de meses de uso e caso de serem novos é necessário identificar a data da garantia.

De forma a proteger os clientes de pessoas com comportamentos menos correctos (por exemplo, o artigo estar danificado, o vendedor não entregar o artigo, o comprador recusar-se a pagar o licitado, etc.), existe a possibilidade de o comprador e o vendedor (opcionalmente) se avaliarem mutuamente relativamente a uma transacção específica. Quer o vendedor quer o comprador podem atribuir uma nota (com a possibilidade de justificar a nota dada) ao comportamento do outro relativamente à transacção.

De acordo com o enunciado, as secções seguintes pretendem mostrar as especificidades da notação e semântica, na definição do modelo conceptual de dados do sistema, utilizando as linguagens ORM e UML (diagrama de classes). Em anexo apresenta-se o modelo de dados global, no Anexo I na modelação ORM e no Anexo II na modelação UML, assim como as DDL's geradas por ambas as linguagens nos Anexos IV e V respectivamente.

É com base neste exercício, que nos capítulos seguintes, se ilustra a comparação entre as linguagens e a sua avaliação.

5.1. ORM – "Leilão On-line"

5.1.1. Entidades e valores

Face ao enunciado apresentado, foram identificadas sete entidades: Artigo Leiloado, Artigo Novo, Artigo Usado, Categoria, Licitação, Utilizador e Avaliação, com os seus respectivos valores.

A entidade "Leilão" irá permitir guardar toda a informação sobre os artigos leiloados como a descrição dos artigos, a base de licitação, o valor da venda, a data e a hora limite de cada artigo a leilão, conforme se ilustra na Figura 31.

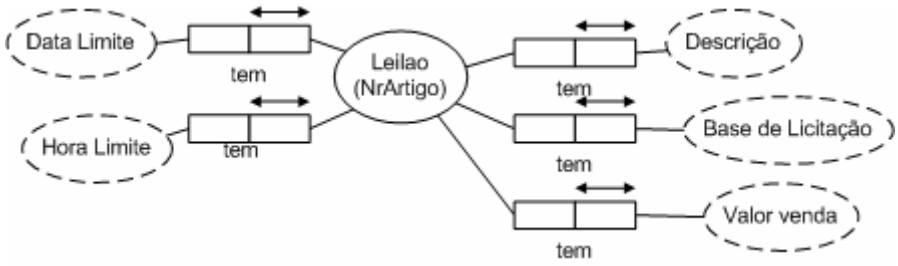


Figura 31 – ORM – Entidade "Leilão"

Como as características dos artigos a leilão são diferentes, quer por se tratarem de artigos novos ou usados e cada um com as suas propriedades, foram identificadas duas entidades para guardar a informação sobre os artigos, a entidade "Artigo Usado" que irá guardar toda a informação que diz respeito aos artigos usados como o número de meses do artigo, e a entidade "Artigo Novo" que irá guardar toda a informação sobre os artigos novos, como a data para efeitos de garantia, conforme se ilustra nas figuras 32 e 33.

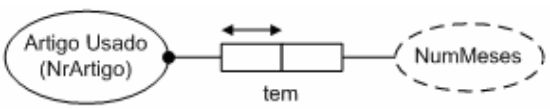


Figura 32 – ORM – Entidade "Artigo Usado"

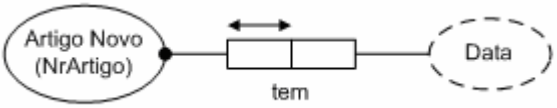


Figura 33 – ORM – Entidade "Artigo Novo"

A entidade "Licitação" irá permitir guardar toda a informação sobre as licitações efectuadas pelos utilizadores sobre um artigo em leilão como o valor de cada licitação efectuada, a data e a hora da licitação, conforme se ilustra na Figura 34.

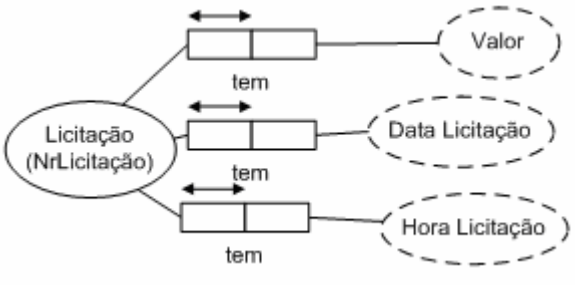


Figura 34 – ORM – Entidade "Licitação"

A entidade "Utilizador" irá permitir guardar a informação sobre os dados dos utilizadores que participam nos leilões, quer sejam vendedores ou compradores como o seu nome e e-mail, conforme se ilustra na Figura 35.

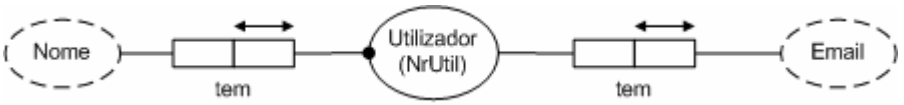


Figura 35– ORM – Entidade "Utilizador"

A entidade "Categoria" irá permitir guardar a informação sobre as categorias dos artigos a leilão, conforme se ilustra na Figura 36.

Figura 36– ORM – Entidade "Categoria"

A entidade "Avaliação" irá permitir guardar a informação sobre a nota dada a uma transacção específica, quer pelo comprador quer pelo vendedor, assim como permitir guardar o comentário sobre a nota dada à transacção, conforme apresentado na Figura 37.

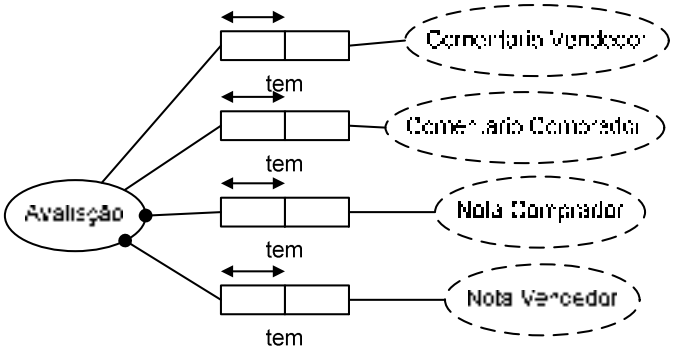


Figura 37 – ORM – Entidade "Avaliação"

5.1.2 Relações

Para cada artigo em leilão é necessário conhecer a sua categoria, pelo que se criou uma relação binária entre a entidade "Leilão" e a entidade "Categoria", onde a entidade "Leilão" recebe o valor da entidade "Categoria", conforme se ilustra na Figura 38.

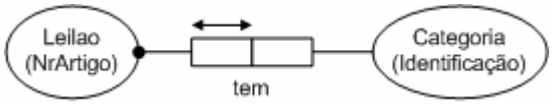


Figura 38 – ORM – Relação "Leilão"- "Categoria"

Para cada artigo em leilão é necessário conhecer o dono do artigo ou vendedor, pelo que se criou uma relação binária entre a entidade "Leilão" e a entidade "Utilizador", onde a entidade "Leilão" recebe o valor da entidade "Utilizador", conforme se ilustra na Figura 39.

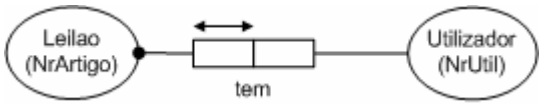


Figura 39 – ORM – Relação "Leilão"- "Utilizador"

Para cada licitação é necessário conhecer o utilizador que faz uma licitação sobre o artigo, pelo que se criou uma relação binária entre a entidade "Licitação" e a entidade "Utilizador", onde a entidade "Licitação" recebe o valor da entidade "Utilizador", conforme se ilustra na Figura 40.

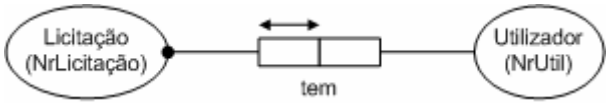


Figura 40 – ORM – Relação "Licitação"- "Utilizador"

Para cada artigo em leilão é necessário guardar a informação sobre as licitações que lhe são efectuadas. Mas a relação entre estas duas entidades tem uma particularidade, só podem existir licitações para um artigo se ele estiver em leilão. Nesta situação foi necessário usar a restrição de obrigatoriedade na entidade licitação, conforme se ilustra na Figura 41.

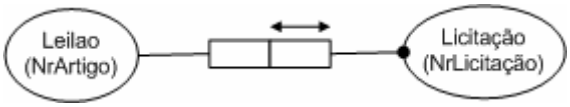


Figura 41 - ORM - Relação "Leilão" - "Licitação"

5.1.3. Subtipo

O facto de um artigo leiloado poder ser um artigo novo ou um artigo usado, significa que um artigo em leilão é um subtipo de um artigo novo ou de um artigo usado, ou seja, o artigo leiloado herda características próprias, quer seja de um artigo usado ou de um artigo novo. A sua representação é feita conforme se ilustra na Figura 42.

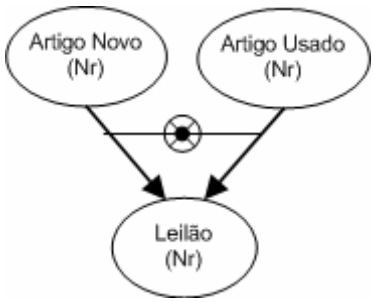
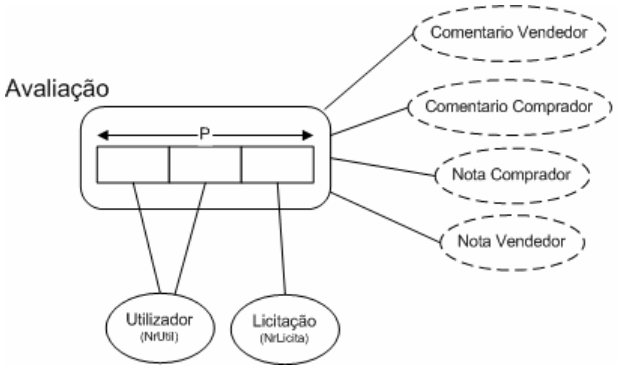


Figura 42 –ORM - Leilão On-line - Subtipo

5.1.4. Objecto aninhado

A entidade "Avaliação" é uma relação entre três objectos, o vendedor, o comprador (ambos utilizadores) e a licitação, mas existem outros objectos a participarem nesta relação, que são as notas e comentários do vendedor e comprador. Esta relação é mais do que um predicado ternário e é chamada de objecto aninhado. A sua é representação é feita conforme ilustra a Figura 43.



NrUtil Vendedor	NrUtil Comprador	NrLicita	Nota Vend	Com Vend	Nota Comp	Com Comp
2	3	10	15	bom	17	Muito bom
1	3	22	10	Mau	12	Mau
3	1	40	10	Mau	12	Mau
1	4	45	...	...	...	...

Figura 43 – ORM - On-line - Objecto aninhado

5.1.5. Restrição de Unicidade

Por exemplo, cada artigo leiloado é caracterizado por apenas uma categoria das existentes, de modo a garantir que essa restrição é aplicada é utilizada a restrição de unicidade, conforme se ilustra na Figura 44.

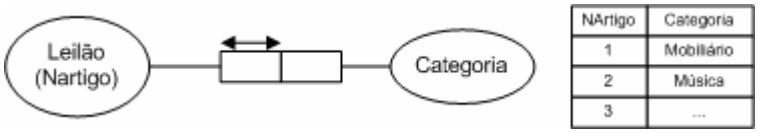


Figura 44 – ORM - Leilão On-line - Restrição Unicidade

5.1.6. Restrição obrigatória

Por exemplo, para fazer cumprir a regra de que cada artigo leiloado deve ter obrigatoriamente uma categoria, teria de se aplicar uma restrição obrigatória, conforme se ilustra na Figura 45.

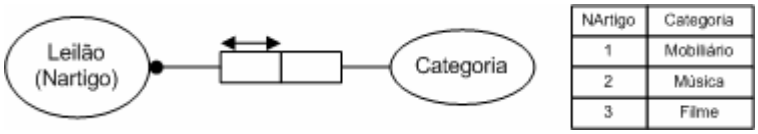


Figura 45 – ORM -Leilão On-line - Restrição obrigatória

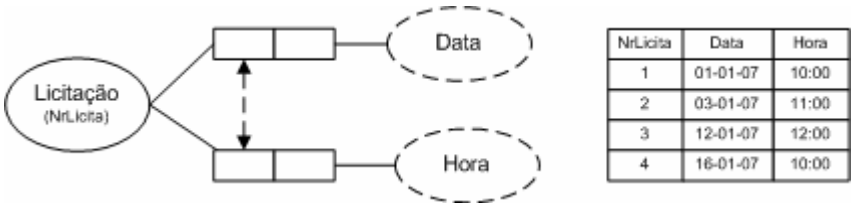
5.1.7. Restrição de frequência

Por exemplo, se cada utilizador só pudesse efectuar duas licitações por cada artigo em leilão, teria de se aplicar uma restrição de frequência, como se ilustra na Figura 46.

Figura 46 – ORM -Leilão On-line - Restrição de frequência

5.1.8. Restrição de Igualdade

Por exemplo, para garantir que existe uma data e uma hora para uma licitação, ou seja, o conjunto das licitações com os valores data terão de ser iguais ao conjunto das licitações com valores hora. O que significa se a data for preenchida para uma licitação a hora também tem de ser. Nesta situação seria aplicada uma restrição de igualdade, conforme se ilustra na Figura 47.

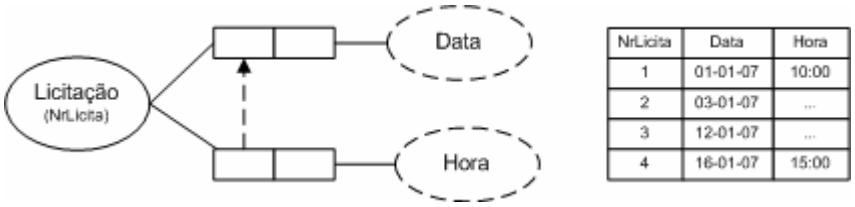


NrLicita	Data	Hora
1	01-01-07	10:00
2	03-01-07	11:00
3	12-01-07	12:00
4	16-01-07	10:00

Figura 47 – ORM - Leilão On-line - Restrição de igualdade

5.1.9. Restrições de Subconjuntos

Agora supondo que para uma licitação que a hora só poderia estar preenchida se a data estivesse também preenchida. O conjunto de licitações com o valor hora preenchido tem estar contido no conjunto de licitações com o valor data preenchido. A sua representação faz-se conforme ilustrado na Figura 48.



NrLicita	Data	Hora
1	01-01-07	10:00
2	03-01-07	...
3	12-01-07	...
4	16-01-07	15:00

Figura 48 - ORM - Leilão On-line - Restrição de subconjunto

5.1.10. Restrições de Exclusão

Para esta restrição, supondo que existia uma regra em que uma licitação tinha de ter a data ou a hora preenchida, a sua representação seria conforme o ilustrado na Figura 49.

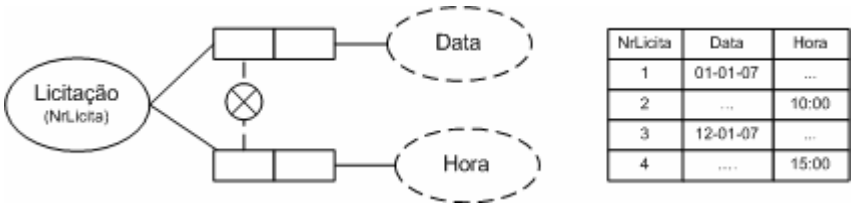


Figura 49 – ORM - Leilão On-line - Restrição de exclusão

Caso alguma a exclusão fosse obrigatória, ou seja, ter uma data ou uma hora preenchida obrigatoriamente, a sua representação seria ilustrada com o símbolo ⊛.

5.1.11. Restrições em Anel

Cada categoria tem uma subcategoria que por sua vez a subcategoria é caracterizada por uma categoria, mas para garantir que uma categoria não pode ser a subcategoria dela própria teria de se aplicar uma restrição em anel do tipo irreversível.

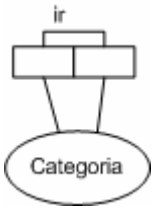


Figura 50 - ORM - Leilão On-line - Restrição em anel

5.1.12. Restrições de Valor

Uma forma de caracterizar os utilizadores seria fazer referência ao sexo de cada um. O sexo só pode ter os valores “M”, de masculino, e “F”, de feminino, para garantir que só ocorrem estes valores teria de se aplicar uma restrição de valor que define quais os valores validos para o tipo de valor “Sexo”. A sua representação faz-se conforme ilustra Figura 51.

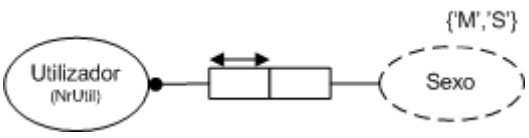


Figura 51 – ORM - Leilão On-line - Restrição de valor

5.1.13. Restrição de Indexes

Um índex tem por finalidade optimizar o acesso à informação, por exemplo se verificasse que os utilizadores (vendedores e compradores) eram consultados no sistema recorrendo à pesquisa por nome, seria necessário aplicar uma restrição de indexe ao tipo de valor "Nome" para optimizar a consulta aos dados dos utilizadores. A sua representação faz-se conforme ilustra na Figura 52.

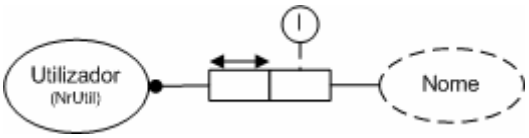


Figura 52 - ORM - Leilão On-line - Restrição indexe

5.2. UML – "Leilão On-line"

5.2.1. Entidades e valores

Face ao enunciado apresentado, foram identificadas sete entidades: Artigo Leiloado, Artigo Novo, Artigo Usado, Categoria, Licitação, Utilizador e Avaliação, com os seus respectivos valores.

A entidade "Leilão" irá permitir guardar toda a informação sobre os artigos leiloados como a descrição dos artigos, a base de licitação, o valor da venda, a data e a hora limite de cada artigo a leilão, conforme se ilustra na Figura 53.

Figura 53 – UML – Entidade "Leilão"

Como as características dos artigos a leilão são diferentes, quer por se tratarem de artigos novos ou usados e cada um com as suas propriedades, foram identificadas duas entidades

para guardar a informação sobre os artigos, a entidade "Artigo Usado" que irá guardar toda a informação que diz respeito aos artigos usados como o número de meses do artigo, e a entidade "Artigo Novo" que irá guardar toda a informação sobre os artigos novos como a data para efeitos de garantia, conforme apresentado nas figuras 54 e 55.

Figura 54 – UML – Entidade "Artigo Novo"

Figura 55 – UML – Entidade "Artigo Usado"

A entidade "Licitação" irá permitir guardar toda a informação sobre as licitações efectuadas pelos utilizadores sobre um artigo em leilão como o valor de cada licitação, a data e a hora da licitação, conforme se ilustra na Figura 56.

Figura 56 – UML – Entidade "Licitação"

A entidade "Utilizador" irá permitir guardar a informação sobre os dados dos utilizadores que participam nos leilões, quer sejam vendedores ou compradores como o seu nome e e-mail, conforme se ilustra na Figura 57.

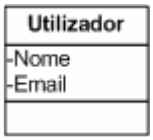


Figura 57 – UML – Entidade "Utilizador"

A entidade "Categoria" irá permitir guardar a informação sobre as categorias dos artigos a leilão, conforme se ilustra na Figura 58.

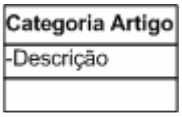


Figura 58 – UML – Entidade "Categoria Artigo"

A entidade "Avaliação" irá permitir guardar a informação sobre a nota dada a uma transacção específica, quer pelo comprador quer pelo vendedor, assim como guardar o comentário sobre a nota dada à transacção, conforme se ilustra na Figura 59.

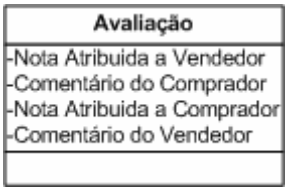


Figura 59 – UML – Entidade "Avaliação"

5.2.2. Relações

Para cada artigo em leilão é necessário conhecer a sua categoria, pelo que se criou uma relação binária entre a entidade "Leilão" e a entidade "Categoria", onde a entidade "Leilão" recebe o valor da entidade "Categoria", conforme se ilustra na Figura 60.

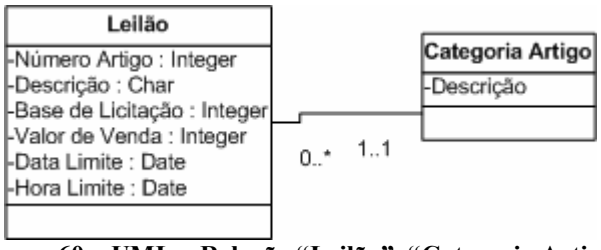


Figura 60 – UML – Relação "Leilão"–"Categoria Artigo"

Para cada artigo em leilão é necessário conhecer o dono do artigo ou vendedor, pelo que se criou uma relação binária entre a entidade "Leilão" e a entidade "Utilizador", onde a entidade "Leilão" recebe o valor da entidade "Utilizador", conforme se ilustra na Figura 61.

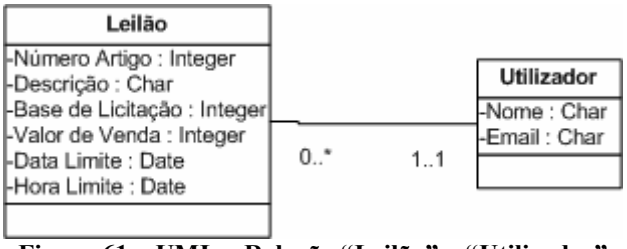


Figura 61 – UML – Relação "Leilão" – "Utilizador"

Para cada licitação é necessário conhecer o utilizador que faz uma licitação sobre o artigo, pelo que se criou uma relação binária entre a entidade "Licitação" e a entidade "Utilizador", onde a entidade "Licitação" recebe o valor da entidade "Utilizador", conforme se ilustra na Figura 62.

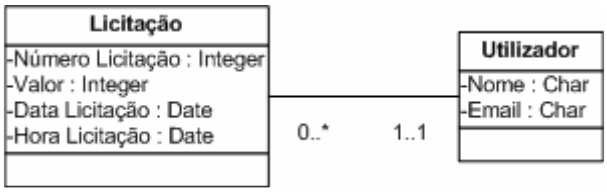


Figura 62 – UML – Relação "Licitação" – "Utilizador"

O facto de um artigo leiloado ser um artigo novo ou um artigo usado, significa que um artigo leiloado é um subtipo de um artigo novo ou de um artigo usado, ou seja o artigo em leilão herda características do artigo novo ou do artigo usado, quer seja de um artigo usado ou de um artigo novo. A sua representação é feita conforme ilustrado na Figura 63.

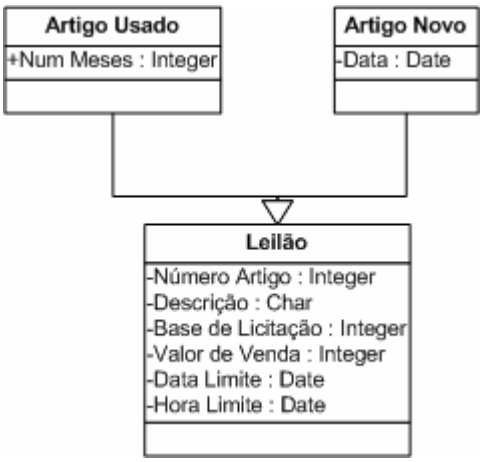


Figura 63 – UML - Leilão On-line -Generalização

A entidade "Avaliação" é gerada a partir de uma relação associativa entre três objectos, o vendedor, o comprador e a licitação. As notas e comentários do vendedor e comprador são atributos da própria classe associativa. Conforme ilustrado na Figura 64

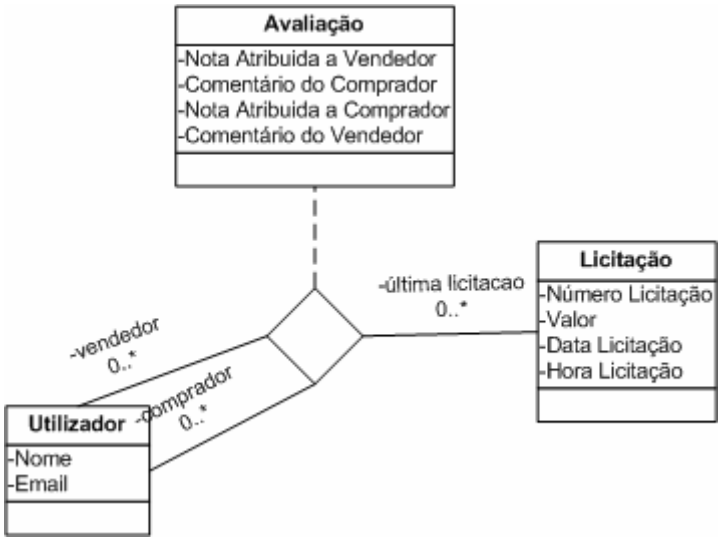


Figura 64 - UML - Leilão On-line – Classe Associativa

Para cada artigo em leilão é necessário guardar a informação sobre as licitações que lhe são efectuadas, mas a relação entre estas duas entidades tem uma particularidade, é só podem existir licitações para um artigo se ele estiver em leilão, ou seja, uma licitação só pode ser referida no contexto de um artigo em leilão. Para esta situação foi necessário criar uma composição entre a entidade "Artigo em Leilão "e a entidade "Licitação", conforme se ilustra na Figura 65.

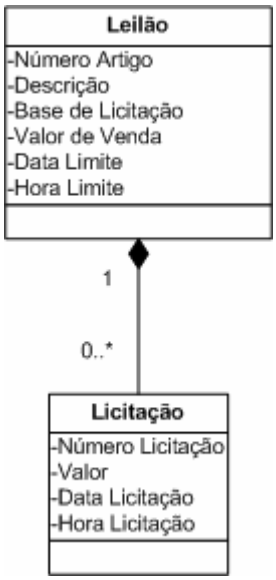


Figura 65 - UML- Leilão On-line – Composição

Cada categoria tem uma subcategoria que por sua vez a subcategoria é caracterizada por uma categoria, pelo que se criou uma relação binária entre a própria entidade "Categoria", conforme se ilustra na Figura 66.

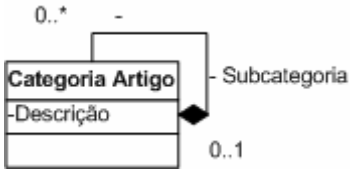


Figura 66 - UML Leilão On-line - Associação sobre mesma classe

5.2.3. Restrições

Como já foi referido, no capítulo 4, o UML recorre à linguagem OCL para representar as restrições no modelo de dados. Para demonstrar como é representada a modelação através da linguagem OCL, aqui se apresentam alguns exemplos.

5.2.3.1. Restrição de Igualdade

Por exemplo, para garantir que existe uma data e uma hora para uma licitação, ou seja, o conjunto das licitações com os valores data terão de ser iguais ao conjunto das licitações com valores hora. O que significa se a data for preenchida para uma licitação a hora também tem de ser. Nesta situação seria aplicada uma restrição de igualdade, conforme se ilustra na Figura 67.

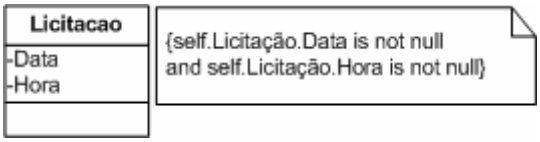


Figura 67 – UML - Leilão On-line - Restrição de igualdade

5.2.3.2. Restrições de Subconjuntos

Supondo que para uma licitação que a hora só poderia estar preenchida se a data estivesse também preenchida, ou seja, o conjunto de licitações com o valor hora preenchido tem estar contido no conjunto de licitações com o valor data preenchido. A sua representação faz-se conforme ilustrado na Figura 68.

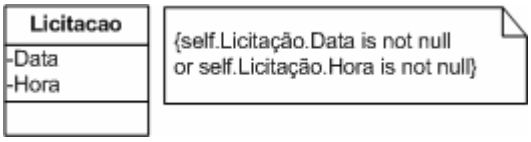


Figura 68 - UML - Leilão On-line - Restrição de subconjunto

5.2.3.3. Restrições de Exclusão

Para esta restrição, supondo que a regra era que uma licitação tinha de ter a data ou a hora preenchida, a sua representação seria conforme ilustrado na Figura 69.

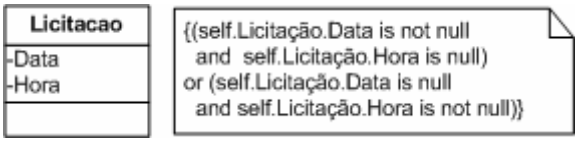


Figura 69 – UML - Leilão On-line - Restrição de exclusão

5.2.3.4. Restrições de Valor

Uma forma de caracterizar os utilizadores seria fazer referência ao sexo de cada um. O sexo só pode ter os valores "M", de masculino, e "F", de feminino, para garantir que só ocorrem estes valores, teria de se aplicar uma restrição de valor que define quais os valores validos para o tipo de valor "Sexo". A sua representação faz-se conforme ilustra a Figura 70.

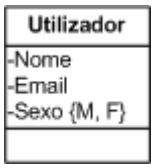


Figura 70 – UML - Leilão On-line - Restrição de valor

6. Comparação das linguagens

Este capítulo pretende ilustrar as diferenças existentes na semântica e notação gráfica entre o ORM e o UML.

6.1. Entidades e atributos/valores

A nível de estrutura de dados existem diferenças entre a modelação em ORM e o diagrama de classes do UML. O ORM representa graficamente dois tipos de objectos, as entidades e os valores, através de eclipses conectados entre si, e a sua representação distingue-se pela linha do eclipse, linha contínua para as entidades e linha tracejada para os valores (Figura 71). No diagrama de classe, do UML, apenas é considerado um objecto, as entidades ou classes, e a sua representação gráfica é feita de forma diferente, as entidades são representadas por uma caixa rectangular dividida por zonas, na primeira coloca-se o nome da entidade e na segunda colocam-se os nomes dos atributos (Figura 72), aqui os atributos representam valores.

Figura 71 - ORM - Entidade e Valor

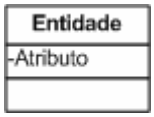


Figura 72 – UML – entidade e atributo

Uma das diferenças entre o ORM e o UML, é que o ORM não faz o uso explícito dos atributos como o UML. Os atributos são substituídos por objectos do tipo valor e têm o mesmo tratamento que um objecto do tipo entidade. Segundo o autor Halpin (2001), esta representação gráfica trás algumas vantagens na estabilidade e uniformidade do diagrama, porque qualquer alteração ao modelo não destabiliza o desenho do diagrama.

Por exemplo, ao caracterizar a entidade "Leilão", faz-se referência aos seus atributos: a data do leilão, o tipo de leilão, o número do artigo, a base de licitação, etc. (figuras 73 e 74). Se no futuro o tipo de leilão tiver que evoluir para uma relação, teria de se criar uma nova entidade e relacionar o "Leilão" com o "Tipo de Leilão", o que significa a criação de mais um objecto no diagrama de classes (Figura 76). No ORM, não seria necessário criar

mais um objecto, sendo a evolução de um objecto tipo valor para um objecto tipo entidade um pormenor gráfico na representação do tipo de objecto. Ou seja, em vez do objecto ser representado por uma eclipse com linha tracejada, passava a ter uma linha contínua e passava a representar um objecto tipo entidade (Figura 75).

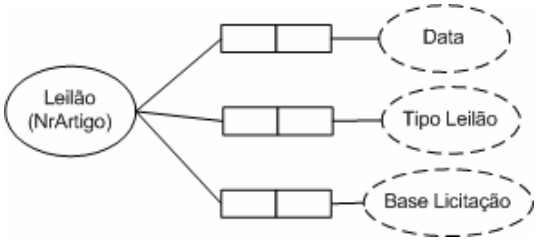


Figura 73 – ORM – Tipo leilão como valor

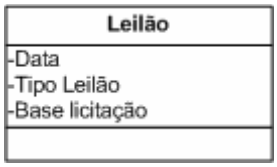


Figura 74 – UML – Tipo leilão como atributo

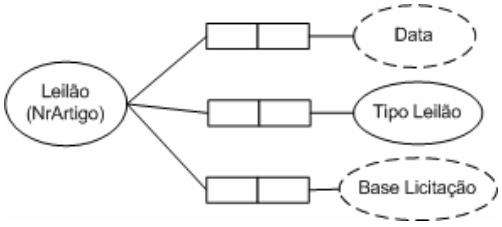


Figura 75 – ORM - Tipo leilão como entidade

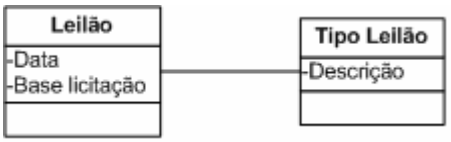


Figura 76 - UML - Tipo leilão como entidade

Outra das diferenças, é que o UML não utiliza nenhum atributo como modo de referência, ou seja, o modo pelo qual uma entidade é identificada univocamente no sistema e que, normalmente, é a referência pela qual a entidade é identificada no mundo real. Para o UML esta característica não é explicitamente representada, o diagrama de classes recorre a extensões da linguagem e, algumas vezes, surge identificada colocando com um "P", à frente do atributo, mas é meramente indicativo e não significa que este atributo seja usado como chave primária no sistema. No ORM, esta representação é obrigatória e explícita, quando se trata da representação do tipo objecto entidade é obrigatório colocar entre parêntesis "( )" o modo de referência por baixo do nome da entidade.

Figura 77 – ORM - Modo de Referência

Figura 78 – UML - Modo de referência

6.2. Relacionamentos e tipos de associações

Como referido anteriormente, o ORM utiliza a notação de predicado para representar o relacionamento que cada objecto tem isoladamente ou com outros objectos, sendo a sua representação ilustrada com pequenos rectângulos entre os objectos. O UML não utiliza esta notação, sendo o relacionamento entre as várias entidades/classes representadas por linhas, não permitindo relacionamentos entre entidades e atributos, ao contrário do ORM que permite relacionar entidades com valores (Figura 79).

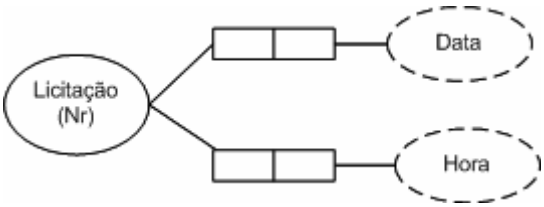


Figura 79 - ORM - Relação entre entidades e valores

Com a notação de predicado, o ORM tem a possibilidade de representar qualquer tipo de relação/associação, desde uma relação unária até às relações *n*-árias, bastando acrescentar um rectângulo de acordo com as relações existente entre os objectos. Assim no ORM, por exemplo, é possível representar graficamente uma relação unária (Figura. 80), já no UML essa representação é feita através de um atributo do tipo booleano (Figura 81).

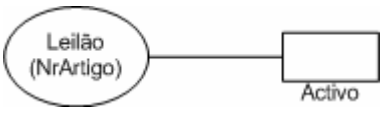


Figura. 80 – ORM – relação unária

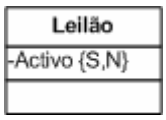


Figura 81 – UML – relação unária

A representação das relações binárias são muito semelhantes, em ambas as linguagens, mas é de notar que o ORM recorre às restrições de obrigatoriedade e de unicidade para representar os vários tipos de relações/associações: um para muitos, muitos para um, um para um e muitos para muitos, conforme as figuras abaixo apresentadas:

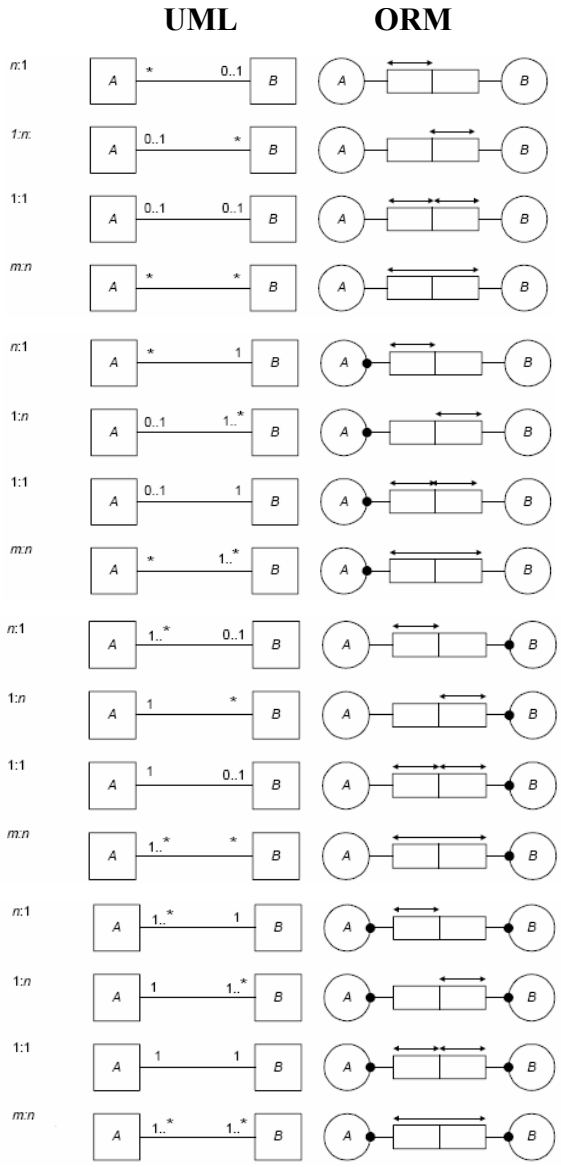


Figura 82 - Equivalência da relação binária

Segundo Silva (2001), no UML a representação gráfica de uma relação *n*-ária, a partir da quaternária, já começa a tornar-se difícil, sendo necessário recorrer à transformação de várias relações binárias entre a classe associativa e as restantes classes participantes, de modo a permitir e facilitar a sua representação. Numa relação ternária o UML usa um losango para representar o relacionamento entre as entidades (Figura 84), enquanto o ORM utiliza três rectângulos (três predicados) para a sua representação (Figura 83).

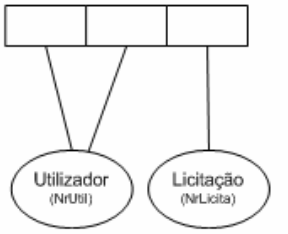


Figura 83 – ORM - relação ternária

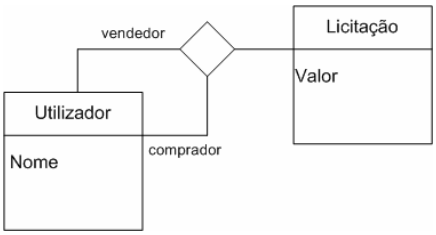


Figura 84 – UML – relação ternária

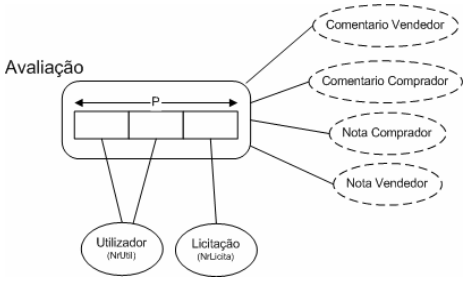


Figura 85 – ORM – relação ternária com objecto aninhado

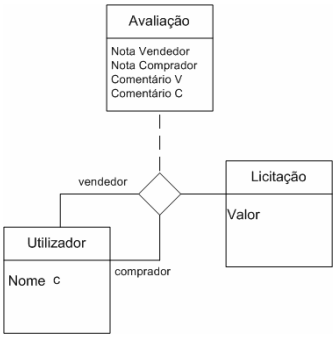


Figura 86 – UML – relação ternária com classe associativa

No ORM quando um objecto participa numa relação de um ou mais objectos designa-se como um objecto aninhado, ou objecto associativo, e representa-se colocando um eclipse à volta da relação (Figura 85). No UML significa ter uma classe associativa em que essa associação tem os seus próprios atributos (Figura 86).

No UML são identificados casos de especiais dos tipos de associações, como é o caso das agregações e composições. Segundo Halpin (2001), o ORM não tem notação gráfica própria para estes tipos de associações, sendo utilizada sua notação para os tipos de associações sem dar ênfase à esta semântica

Figura 87 – ORM – exemplo de uma agregação

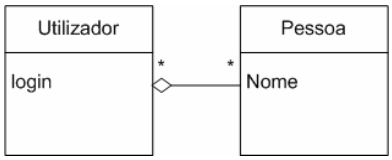


Figura 88 – UML - Agregação

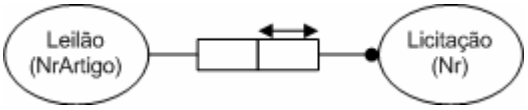


Figura 89 – ORM – exemplo de uma composição

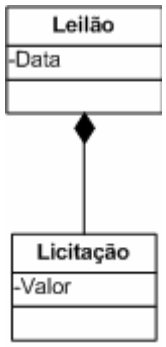


Figura 90 – UML - Composição

As generalizações também são outro caso especial das associações no UML, para estes casos o ORM também tem notação própria, a que chama de subtipos de entidades.

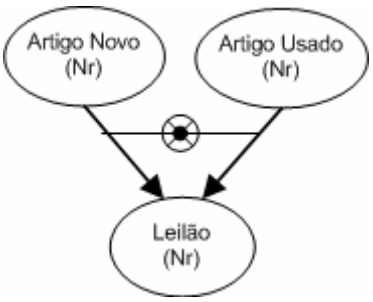


Figura 91 – ORM - Subtipos

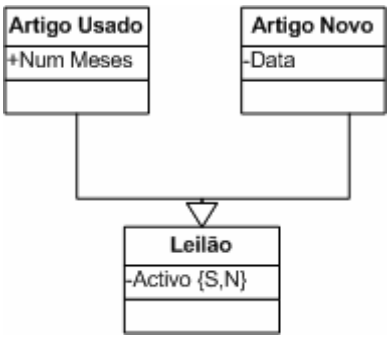


Figura 92 – UML – Generalização

6.3. Restrições

O ORM disponibiliza notação gráfica para várias restrições, enquanto o UML tem uma notação gráfica mais reduzida para estas restrições e quando se trata de aplicar as restrições entre atributos de uma entidade o UML recorre à linguagem OCL para complementar os seus diagramas.

O OCL é uma linguagem de expressões para especificar restrições sobre modelos, e é parte integrante do UML**.** É uma linguagem precisa, textual e formal que permite descrever regras através de expressões matemáticas simples, garantindo a não existência de interpretações ambíguas, para as mesmas restrições, e com a vantagem de não exigir um forte conhecimento matemático para ser utilizada correctamente.

6.3.1. Restrição de unicidade

No UML não existe nenhuma notação gráfica para esta restrição. O UML recorre a extensões da própria linguagem para a sua representação, quando se trata de aplicar esta restrição entre entidades e atributos, colocando um "U" à frente do atributo que pretende ser único. O ORM apresenta notação para esta restrição e é, também, através desta que define a multiplicidade dos tipos de objectos.

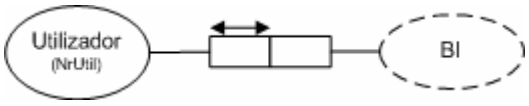


Figura 93 – ORM – Restrição de unicidade

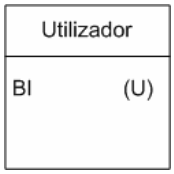


Figura 94 – UML – Restrição de Unicidade

6.3.2. Restrição de obrigatoriedade

No UML esta representação é feita colocando, à frente do nome dos atributos, entre parêntesis rectos o tipo de multiplicidade do atributo, por exemplo, [0…1] para opcional e [1…1] para obrigatório (Figura 96). Segundo a OMG (2007), caso não seja dada nenhuma indicação, por defeito, os atributos são considerados obrigatórios. Já o ORM só utiliza esta restrição caso os valores sejam de preenchimento obrigatório (Figura 95) colocando um círculo preenchido de preto (•) do lado do objecto que deve respeitar esta restrição.

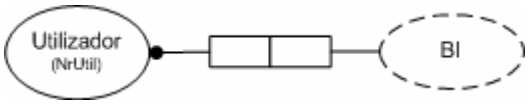


Figura 95 - ORM – Restrição obrigatória

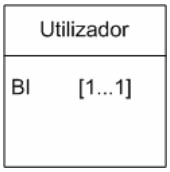


Figura 96 – UML - Restrição obrigatória

6.3.3. Restrição de Frequência

No UML esta restrição é definida no tipo de multiplicidade, substituindo o "*" pelo número de ocorrências permitidas. O ORM coloca o valor da restrição dentro de um círculo sobre o predicado correspondente, conforme ilustrado na Figura 97.

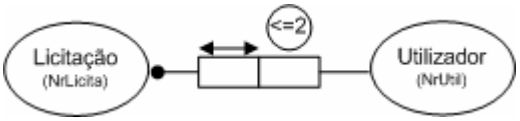


Figura 97 - ORM – Restrição de Frequência

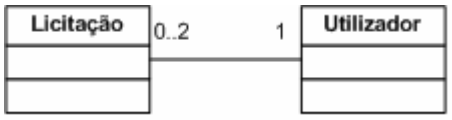


Figura 98 – UML - Restrição obrigatória

6.3.4. Restrição de Subconjunto

O UML não tem representação gráfica para esta restrição, quando se trata de restrições entre atributos da mesma classe mas consegue representa-la através da linguagem OCL, onde é colocada uma nota com a respectiva expressão junto da classe, como se ilustra na Figura 100.

O UML consegue representar esta restrição graficamente mas considerando os atributos como classes, o que seria dar a importância de uma classe a um atributo, e quando geradas as definições de dados (DDL) seriam criadas as tabelas "Data" e "Hora" em vez de atributos na tabela "Licitação", conforme ilustrado na Figura 101.

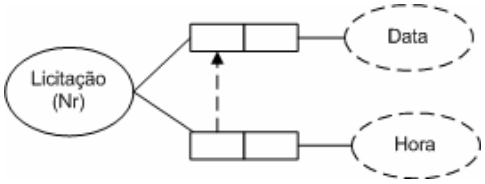


Figura 99 – ORM- Restrição Subconjunto

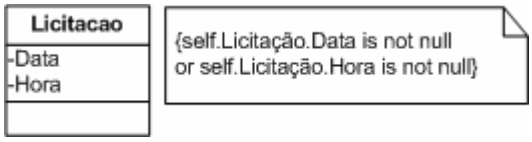


Figura 100 – UML - Restrição Subconjunto entre atributos

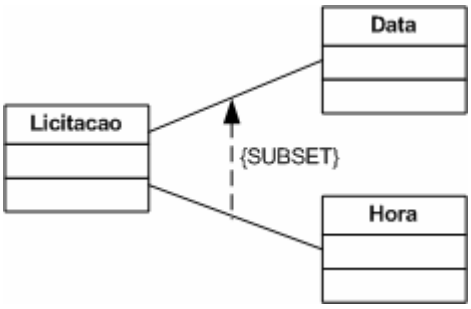


Figura 101 - UML - Restrição Subconjunto entre entidades

6.3.5. Restrição de Igualdade

O UML não tem representação gráfica para esta restrição, quando se trata de restrições entre atributos da mesma classe mas consegue representa-la através da linguagem OCL, colocando uma nota com a respectiva expressão junto da classe.

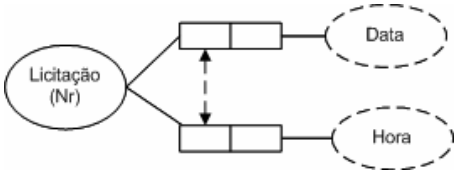


Figura 102 – ORM - Restrição Igualdade

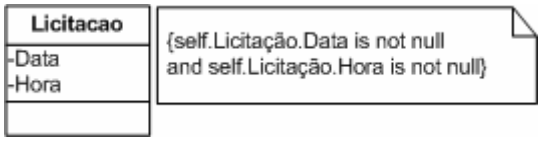


Figura 103 – UML - Restrição Igualdade

6.3.6. Restrição de Exclusão

O UML não tem representação gráfica para esta restrição, quando se trata de restrições entre atributos da mesma classe, mas consegue representa-la através da linguagem OCL, colocando uma nota com a respectiva expressão junto da classe como se ilustra na Figura 105.

O UML pode representar esta restrição mas considerando os atributos como classes, o que seria dar a importância de uma classe a um atributo, ou seja, quando gerada a estrutura de

dados (DDL) seriam criadas as tabelas "Data" e "Hora" em vez de atributos na tabela "Licitação", conforme o ilustrado na Figura 106.

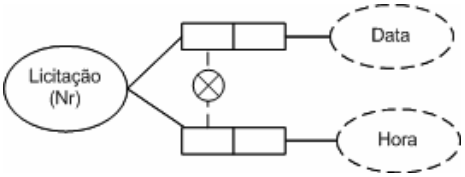


Figura 104 – ORM – Restrição Exclusão

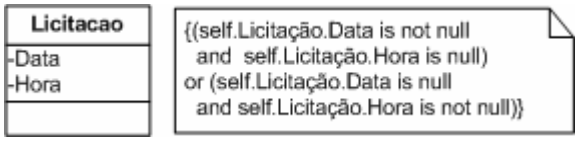


Figura 105 – UML – Restrição Exclusão entre atributos

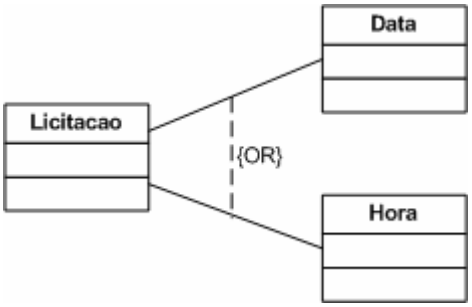


Figura 106 – UML - Restrição Exclusiva entre entidades

6.3.7. Restrição de Anel

A restrição em anel no ORM, é o mesmo que ter uma associação binária, no UML, sobre a mesma entidade. O ORM tem notação para vários tipos de restrição em anel que o UML não representa (tabela 5).

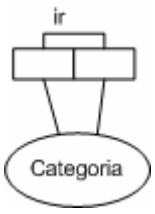


Figura 107 – ORM - Restrição Anel

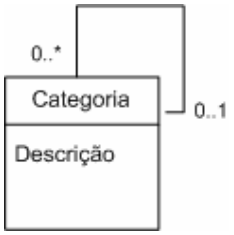


Figura 108 – UML – Associação reflexiva

Tipos de restrições em anel	
Irreflexivo	ir
Anti Simétrico	ans
Simétrico	sym
Intransitivo	it
Acíclico	ac
Assimétrico	as

Tabela 5- ORM – restrições anel

6.3.8. Restrição de Valor

A restrição de valor tem uma representação semelhante em ambas as linguagens, a representação desta é feita colocando entre parêntesis ({}) os valores, ou intervalos de valores, que podem ser assumidos pelo objecto, no caso do ORM, ou pelo atributo, no caso do UML.

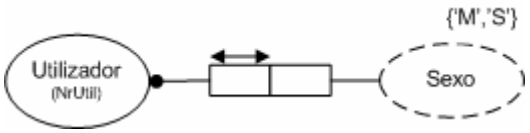


Figura 109 – ORM – Restrição Valor

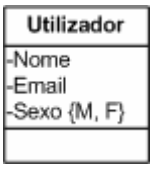


Figura 110 – UML - Restrição Valor

6.3.9. Restrição de indexe

No UML não existe nenhuma notação gráfica para esta restrição, O UML recorre a extensões da própria linguagem para a sua representação, colocando um "I" à frente do atributo que pretende aplicar o indexe.

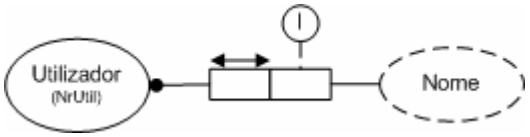


Figura 111 – ORM – Restrição Indexe

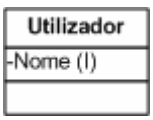


Figura 112 – UML - Restrição Indexe

Na tabela 6 é apresentado um resumo sobre a comparação das linguagens.

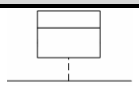
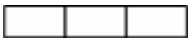
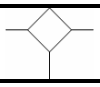
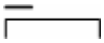
	ORM	UML
Estrutura de dados		
Entidade/Classe		
Valor/Atributo		
Tipos de Associações		
Objecto aninhado/Classe associativa/		
Agregação	Sem representação gráfica	
Composição	Sem representação gráfica	
Subtipos/Generalização/		
Relações		
Unária		Equivale a um atributo do tipo booleano
Binária		
Ternária		
***n*-enaria**		Sem representação gráfica
Restrições		
Obrigatoriedade		Multiplicidade
Unicidade		Sem representação gráfica
Frequência		Multiplicidade
Subconjunto		Recorre OCL, para restrições entre atributos
Igualdade		Recorre OCL, para restrições entre atributos
Exclusão		Recorre OCL, para restrições entre atributos
Anel		Relação com a própria entidade
Valor	{}	{}
Índex		Sem representação gráfica

Tabela 6 – Grelha de comparação das linguagens

7. Avaliação das linguagens

Após comparadas as linguagens, procedemos à sua avaliação de acordo com os critérios definidos por Patrício (2003), baseados nos autores Lindland, Sindre e Solvberg (1994) Para construção da grelha de avaliação foi, também, adoptado o método de valoração proposto no modelo de avaliação PHIMA, do autor Sousa (1998), que será aplicado às ponderações que adiante se descrevem.

O objectivo desta avaliação é analisar a qualidade semântica e sintáctica das linguagens de modelação de dados aqui apresentadas, de forma a apurar qual das linguagens apresenta uma semântica e uma notação gráfica mais eficaz e expressiva, na representação e na completude da definição do modelo conceptual de dados, que permitam a geração de modelos com o máximo de automatismo e rigor sobre domínio da aplicação.

7.1. Qualidade Semântica

A qualidade semântica pretende avaliar a completude e a validade das linguagens. A completude corresponde à capacidade do modelo em conter todas as declarações que são correctas e relevantes para o domínio, se o modelo não representa todos os elementos relevantes do mundo real diz-se incompleto ou omisso. A validade significa que todas as declarações no modelo são correctas e relevantes ou que a linguagem de modelação específica correcta e relevantemente os mecanismos de abstracção e as restrições utilizadas. A validade divide-se em quatro sub critérios que são a: correcção, univocidade, minimalismo e sentido.

A correcção pretende medir o grau de aproximação do modelo à semântica dos dados descritos em linguagem natural, avaliando se existem elementos no modelo que apesar de corresponderem aos requisitos possam estar definidos de forma incorrecta.

A univocidade pretende medir a unidade de interpretação semântica de cada elemento do modelo, avaliando a sua ambiguidade, se existirem vários aspectos do mundo-real que possam ser mapeados para o mesmo aspecto do modelo, este diz-se ambíguo.

O minimalismo pretende medir a ausência de redundância no modelo, avaliando às vezes que um aspecto do domínio aparece no modelo, diz-se minimal se cada aspecto aparecer uma só vez, e, se não for possível eliminar nenhum conceito do modelo sem se perder informação.

O sentido pretende medir o sentido do modelo, avaliando se nele apenas estão representados os elementos do domínio da aplicação ou se, também, existem elementos que não correspondem a nenhum requisito de informação e que por isso são desnecessários.

7.2. Qualidade Sintáctica

A qualidade sintáctica pretende avaliar a correspondência entre o modelo e a linguagem, no que se refere aos símbolos do alfabeto da linguagem de modelação e da obediência do modelo às regras gramaticais da linguagem da modelação.

Um dos critérios desta avaliação é o da coerência gramatical que mede a coerência das regras gramaticais de modelação, tal coerência permite que o utilizador saiba identificar claramente um dado objecto no modelo, quantas mais regras existirem para representar elementos mais confusa se torna a linguagem.

O outro critério de avaliação da qualidade sintáctica é a simplicidade, que mede a simplicidade dos elementos sintácticos da linguagem, o que significa que um modelo deve conter o mínimo número possível de objectos. A simplicidade da linguagem de modelação permite que um modelo seja facilmente compreendido.

7.3. Quadro de avaliação

Uma vez que o pretendido é avaliar a qualidade das linguagens na sua capacidade de representação e completude na definição do modelo conceptual de dados, através da análise da sua expressividade gráfica, considera-se mais importante a qualidade semântica da linguagem do que propriamente a sua qualidade sintáctica, pois, uma linguagem pode

ser simples e coerente gramaticalmente mas não quer dizer que reflicta totalmente todos os aspectos relevantes do domínio da aplicação.

Outra razão, por se minimizar a importância da qualidade sintáctica das linguagens, nesta avaliação, deve-se ao facto dos modelos de dados serem desenhados e definidos com o recurso a ferramentas CASE. O que significa que a qualidade sintáctica fica garantida pelas as ferramentas, que validam a coerência gramatical das linguagens, o que já não acontece com a qualidade semântica, pois as ferramentas apenas representam a notação que lhes está associada de acordo com linguagem que estão a utilizar.

A qualidade semântica é, então, aquela que permite medir capacidade que um modelo tem em conter todas as declarações que são correctas e relevantes para o domínio da aplicação e não a qualidade sintáctica. Assim, a ponderação a atribuir à dimensão da qualidade semântica terá o dobro do peso da ponderação a atribuir à dimensão da qualidade sintáctica. O valor da ponderação a atribuir a cada dimensão é calculado de acordo o número dimensões a multiplicar pelo número total de critérios a avaliar, conforme a seguinte expressão:

$$PesoDimensão = Nº\,Dimensões \times Nº\,Critérios \quad (1)$$

Existindo duas dimensões (semântica e sintáctica) e quatro critérios (completude, validade, clareza e simplicidade), a ponderação a atribuir à qualidade semântica será de 16 pontos, resultante da seguinte expressão:

$$Qualidade\_Semântica = 2 \times PesoDimensão \quad (2)$$

Já a ponderação a atribuir à dimensão da qualidade sintáctica é de 8 pontos, de acordo com o valor calculado através da seguinte expressão:

$$Qualidade\_Sintáctica = PesoDimensão \quad (3)$$

Em relação aos critérios de avaliação para cada dimensão, a ponderação a atribuir corresponde à distribuição do peso de cada dimensão pelos seus critérios, conforme a seguinte expressão:

$$PesoCritério = PesoDimensão / Nº Critérios \quad (4)$$

Ficando com o peso de 8 pontos os critérios de completude e validade, da dimensão da qualidade semântica, que se subdividem em ponderações iguais pelos vários sub critérios.

No entanto, em relação aos critérios da dimensão sintáctica, é dada maior importância à coerência gramatical do que à simplicidade morfológica, pelo que, se atribui 5 pontos à coerência gramatical e 3 pontos à simplicidade morfológica, por se considerar mais importante a clareza de uma linguagem do que a sua complexidade pois a falta de clareza numa linguagem pode levar a interpretações menos correctas do modelo, podendo gerar modelos que não reflectem realidade do domínio da aplicação.

Na tabela abaixo faz-se a distribuição dos pesos pelos diferentes critérios, dando-se maior ponderação aos critérios relativos à semântica das linguagens (peso 16) do que à qualidade sintáctica das linguagens (peso 8).

Dimensão de Analise	Critério	Sub-critério	Descrição	Métrica	Valores		Peso
Qualidade Semântica 16	Completude		Possibilidade representar todos os requisitos do domínio	Número de aspectos não representados	0	Nenhuma representação	8
					0,25	Pouca representação	
					0,50	Razoável representação	
					1	Total representação	
	Validade	Correcção	Respeito pelos requisitos do domínio de aplicação	Número de aspectos incorrectos	0	Total incorrecção	2
					0,25	Pouca correcção	
					0,50	Correcção razoável	
					1	Total correcção	
	Validade	Univocidade	Validade Unidade de interpretação semântica	Número de ambiguidades	0	Total ambiguidade	2
					0,25	Pouca univocidade	
					0,50	Univocidade razoável	
					1	Total univocidade	
	Validade	Minimalismo	Inexistência de redundância	Número de aspectos redundantes	0	Total redundância	2
					0,25	Pouco minimalismo	
					0,50	Minimalismo Razoável	
					1	Total Minimalismo	
	Validade	Sentido	Todos os elementos do modelo têm mapeamento no mundo real	Número de elementos sem sentido	0	Total ausência de sentido	2
					0,25	Pouco sentido	
					0,50	Sentido razoável	
					1	Total Sentido	
Qualidade Sintáctica 8	Clareza		Definição clara das regras gramaticais	Número de elementos sem sentido	0	Ausência de clareza	5
					0,25	Pouca clareza	
					0,50	Clareza razoável	
					1	Total Clareza	
	Simplicidade Morfológica		Mínimo número possível de elementos	Número de elementos	0	Excessivo número de elementos	3
					0,25	Muitos elementos	
					0,50	Número razoável de elementos	
					1	Poucos elementos	

Tabela 7 - Quadro de ponderações

7.4. Atribuição de pontuação

Código: 1.1	**Parâmetro:** Entidade	**Tipo:** Objectos	
Explicação: Descrição decisiva do objecto do mundo real			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	0,50	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	0,50

a) O ORM torna-se mais completo na representação de uma entidade, porque na sua definição é identificada obrigatoriamente o seu modo de referência, ou seja, a referência pela qual a entidade é identificada univocamente no sistema, que corresponde à chave primária que é propriedade fundamental de uma tabela no modelo relacional. O UML não exige que essa identificação seja obrigatória, o que leva a posteriores correcções ao modelo de dados depois de gerado. Por esta razão, se atribui a pontuação de 0,5 ao UML e ao ORM a pontuação máxima.

Figura 113 – Entidade ORM

Figura 114 – Entidade UML

g) Pela mesma razão, da alínea anterior, o UML torna-se mais simples na representação de uma entidade porque não obriga a identificação do elemento pela qual a entidade é univocamente identificada. O que também pode ser uma vantagem pois podem existir tabelas no modelo onde não seja necessária a criação de chave primária. Pelo que, se atribui a pontuação de 0,5 ao ORM e ao UML a pontuação máxima.

Código: 1.2	**Parâmetro:** Valor/Atributo	**Tipo:** Objectos	
Explicação: Referência ou propriedade de uma entidade			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	0,50
	g) Simplicidade	1	0,50

f) A representação gráfica de um valor no ORM é muito semelhante ao de uma entidade podendo, por vezes, tornar o modelo menos claro e mais confuso. Como se pode verificar, no exemplo abaixo, a única diferença entre a representação de uma entidade e um valor é, apenas, a linha do eclipse, contínua para a entidade e tracejada o para o valor, podendo não sobressair o que é uma entidade e o que é um valor. Por esta razão, se atribui a pontuação de 0,5 ao ORM e ao UML a pontuação máxima.

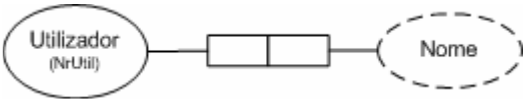


Figura 115- Valor ORM

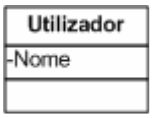


Figura 116 - Atributo UML

g) A inexistência de representação gráfica para um atributo no UML torna-o mais simples, pois é menos um elemento gráfico a aplicar ao modelo apenas é descrito em linguagem natural na definição da classe, como mostra a figura anterior. Por esta razão, se atribui a pontuação de 0,5 ao ORM e ao UML a pontuação máxima.

Código: 2.1	**Parâmetro:** Predicado	**Tipo:** Relação	
Explicação: Papel que o objecto desempenha, isolado ou relacionado com outros objectos			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	0,25	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

a) O UML não utiliza a notação de predicado para representar o relacionamento ou a associação entre objectos, a associação faz-se através de uma linha contínua entre os

objectos, no caso do UML entre as classes. O ORM tira partido desta notação, pois permite-lhe representar relacionamentos quer entre entidades, quer entre entidades e valores, uma vez que existe a representação gráfica para os valores torna-o mais completo em relação ao UML. Pelo que se atribui, no critério de completude, a pontuação máxima ao ORM e a pontuação de 0,25 ao UML.

Código: 2.2	**Parâmetro:** Unária		**Tipo:** Relação	
Explicação: Relação entre um objecto				
Dimensão	**Critério**		**UML**	**ORM**
Qualidade Semântica	a) Completude		0,50	1
	b) Correcção		1	1
	c) Univocidade		1	1
	d) Minimalismo		1	1
	e) Sentido		1	1
Qualidade Sintáctica	f) Clareza		1	1
	g) Simplicidade		1	1

a) O ORM faz uso da representação gráfica da relação unária, recorrendo aos predicados. Para o UML não existe esta representação, sendo considerada uma relação unária a definição de um atributo do tipo booleano, mas é necessário que seja indicado o tipo de dados que atributo vai representar, caso contrário será necessário efectuar posteriores correcções ao modelo de dados gerado.

No exemplo abaixo, está representada a entidade "Leilão" com o atributo "Activo" que irá receber o valor verdadeiro ou falso. Como no ORM é necessário representar sempre as relações entre qualquer objecto, logo ao definir um predicado com uma relação está indicar que o atributo terá de ser booleano, enquanto que o UML não impõe a essa representação, tornando-o menos completo graficamente em relação ao ORM. Pelo que, se atribui a pontuação máxima ao ORM e ao UML a pontuação de 0,5.

Figura 117 – ORM – relação unária

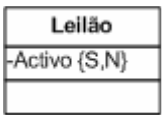


Figura 118 – UML – relação unária

Código: 2.3	**Parâmetro:** Binária	**Tipo:** Relação	
Explicação: Relação entre dois objectos			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	**a)** Completude	1	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	**f)** Clareza	1	1
	g) Simplicidade	1	0,50

g) Uma relação binária no ORM é representada recorrendo a dois elementos da linguagem, ao predicado para definir o número de regras e à restrição de unicidade para definir a multiplicidade, enquanto o UML apenas faz a representação da relação binária representado a multiplicidade que existe entre as classes, utilizando apenas uma linha contínua entre as classes.

No exemplo seguinte, mostra-se a representação da relação binária que existe entre a "Licitação" e o "Utilizador", em que uma licitação pode estar ou não associada a um utilizador. No ORM esta representação é feita a partir do predicado binário (duas caixas) e da restrição de unicidade. Mas, no caso, de uma licitação ter que estar obrigatoriamente associada a um utilizador já se teria de representar mais uma restrição, a restrição de obrigatoriedade, enquanto no UML seria só alterar o tipo de multiplicidade entre as classes, tornando-o muito mais simples graficamente em relação ao ORM. Pelo que, se atribui a pontuação de 0,25 ao ORM e ao UML a pontuação máxima.

Figura 119 – ORM Relação binária

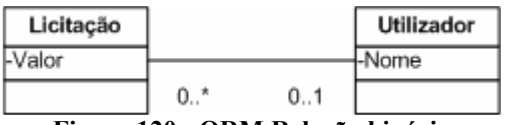


Figura 120 - ORM Relação binária

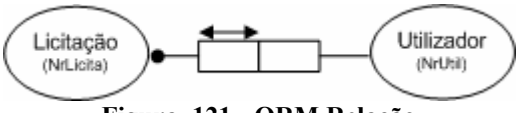


Figura. 121 - ORM Relação binária/obrigatoriedade

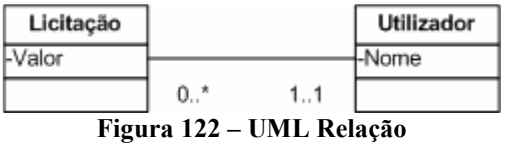


Figura 122 – UML Relação binária/obrigatoriedade

Código: 2.4	**Parâmetro:** *n*-ária	**Tipo:** Relação	
Explicação: Relação entre *n* objectos			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	0,50	1
	g) Simplicidade	1	1

f) O ORM permite a representação gráfica de qualquer relação *n*-ária, bastando colocar o número de predicados (caixas) necessários à sua representação. O UML só tem notação gráfica até à ternária, onde usa um losango para representar a relação entre três classes. Quando uma relação é superior, sua representação já não é tão directa pois tem decompor este tipo de relação em várias relações binárias, não ficando representada de maneira tão clara como no ORM. Por esta razão, se atribui pontuação máxima ao ORM e a pontuação de 0,5 ao UML.

Código: 3.1	**Parâmetro:** Objecto aninhado/ Classe Associativa/	**Tipo**: Associação	
Explicação: Um objecto participa numa relação de um ou mais objectos.			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	0,5	1
	g) Simplicidade	1	1

f) No ORM mais facilmente se identifica um objecto aninhado no modelo porque a relação que gera o objecto fica demarcada por um rectângulo à sua volta, como ilustra a figura abaixo. No UML uma classe associativa é identificada através de uma linha tracejada que a une a uma outra relação entre duas classes.

Na figura abaixo, mostra um exemplo como é representado um objecto aninhado e uma classe associativa. Torna-se mais evidente, através do ORM, que a entidade

"Avaliação" é gerada a partir da relação entre outras entidades, neste caso, uma associação ternária entre as entidades "Utilizador" (vendedor e comprador) e "Licitação". Por esta razão, se atribui pontuação máxima ao ORM e a pontuação de 0,5 ao UML.

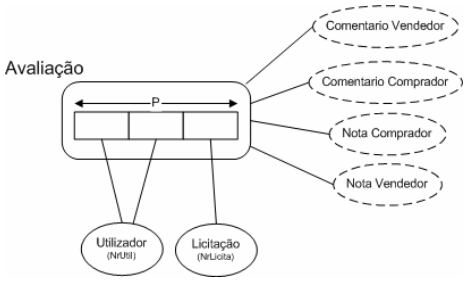


Figura 123 – ORM – relação ternária com objecto aninhado

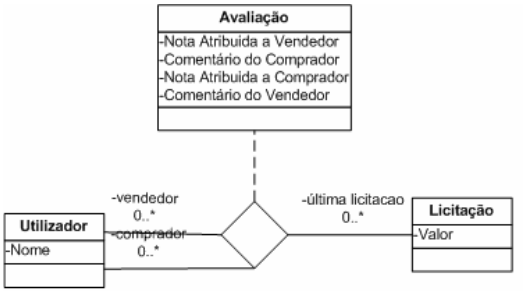


Figura 124 – UML – relação ternária com classe associativa

Código: 3.2	**Parâmetro:** Agregação	**Tipo:** Associações	
Explicação: Um objecto que consiste na agregação de um conjunto de objectos			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	0,5
	b) Correcção	1	1
	c) Univocidade	1	0,25
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

a) O ORM não tem notação própria para este caso especial da associação que é utilizada para expressar que um objecto se compõe pela agregação de um conjunto de outros objectos. Dado que um modelo deve ilustrar a realidade, da forma mais aproximada, considera-se que o UML é mais completo do que o ORM. Atribuindo-se pontuação máxima ao UML e 0,5 pontos ao ORM.

Por exemplo, supondo que no sistema "Leilão On-line" se queria expressar que um utilizador é constituído por um conjunto de pessoas, ou seja, que um utilizador é atribuído a um conjunto de pessoas, o UML assinalava este tipo de associação com um losango (Figura 126), o ORM representa-a como sendo uma associação de um para muitos (Figura 125), não sendo expressa essa realidade.

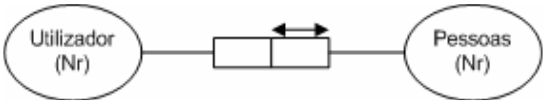


Figura 125 – ORM – exemplo de Agregação

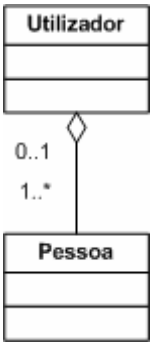


Figura 126 – UML – Exemplo de Agregação

c) Pelo facto do ORM não ter notação própria para este caso especial e por utilizar a notação já existente para outro tipo de associações, considera-se o ORM mais ambíguo, sendo-lhe atribuído 0,25 pontos e ao UML atribuído a máxima pontuação.

Código: 3.3	**Parâmetro:** Composição	**Tipo:** Associações	
Explicação: Um objecto é composto por outros objectos			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	0,5
	b) Correcção	1	1
	c) Univocidade	1	0,25
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

a) O ORM não tem notação própria para este caso especial da associação, que serve para expressar que um objecto é composto por um conjunto de outros objectos. Dado que um modelo deve ilustrar a realidade, da forma mais aproximada, considera-se que o UML é mais completo do que o ORM. Atribuindo-se pontuação máxima ao UML e 0,5 pontos ao ORM.

Por exemplo, no sistema "Leilão On-line" um artigo em leilão pode ter várias licitações mas essas licitações só podem existir quando associados a um leilão. As licitações distinguem-se pelo artigo em leilão a que pertencem e não podem existir isoladamente. No UML este tipo associação sobressai no modelo porque é assinalada com um losango negro (Figura 128). O ORM representa esta associação como sendo uma associação de um para muitos (Figura 127), não sendo expressa essa realidade.

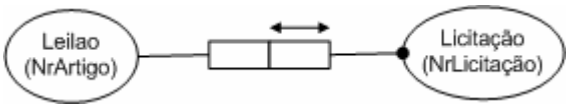


Figura 127 – ORM – exemplo de Composição

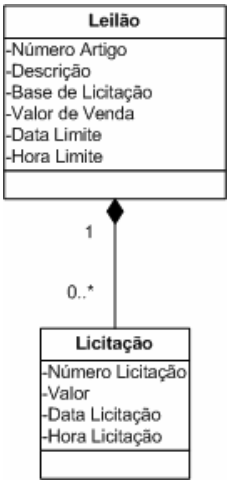


Figura 128 – UML – Exemplo de Composição

d) Pelo facto do ORM não ter notação própria para este caso especial e por utilizar a notação já existente para outro tipo de associações considera-se o ORM mais ambíguo, sendo-lhe atribuído 0,25 pontos e ao UML atribuído a máxima pontuação.

Código: 3.4	**Parâmetro:** Generalização/Subtipo		**Tipo:** Associações
Explicação: Noção de			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

A representação deste tipo de associação é muito semelhante quer no UML quer no ORM, não havendo nada a assinalar.

Código: 4.1	**Parâmetro:** Obrigatoriedade	**Tipo:** Restrições
Explicação: Garante que cada instância de um tipo de objecto só ocorre quando respeitada a regra.		

Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	0,50

g) Por defeito no UML os atributos são considerados de preenchimento obrigatório, não sendo necessário qualquer representação para o expressar. No ORM é o contrário, sempre que um valor é de preenchimento obrigatório tem que se aplicar a restrição de obrigatoriedade. Por esta razão o UML torna-se mais simples do que o ORM porque evita a utilização de mais elemento no modelo. Por isto se atribui pontuação máximo ao UML e ao ORM é dada a pontuação de 0,5.

Por exemplo se considerarmos a entidade "Leilão", faz sentido que todos seus atributos sejam de preenchimento obrigatório, pois existindo um artigo em leilão é necessário conhecer a sua descrição, a sua base de licitação, o seu valor, a sua data e hora limite. Como se pode verificar na Figura 130, o UML não fez uso de mais nenhum elemento na representação da entidade "Leilão".

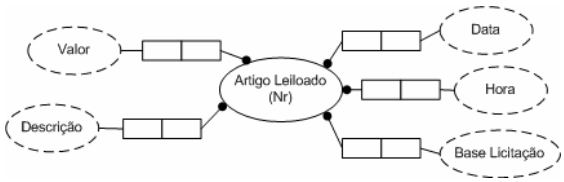


Figura 129 – ORM – Restrição Obrigatoriedade

Figura 130 – UML - Restrição Obrigatoriedade

Código: 4.2	**Parâmetro:** Unicidade	**Tipo:** Restrições	
Explicação: Garante que uma instância, de um tipo de objecto, é única.			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	0,5
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	0,5

e) A restrição de unicidade no ORM serve para definir a multiplicidade do tipo de associações entre entidades como também a unicidade de um valor numa entidade valor. Sempre que se define um valor numa entidade é necessário aplicar esta restrição. No UML esta restrição só é aplicada caso seja um requisito de informação. Por esta razão, faz mais sentido a representação da unicidade dos atributos no UML, porque só é mencionada quando de facto é um requisito. Pelo que se atribui, ao critério do sentido, a pontuação máxima ao UML e a pontuação de 0,5 ao ORM.

g) Pela mesma razão, da alínea anterior, o UML é mais simples a representação da restrição de unicidade, por defeito nenhum atributo é único, só no caso de ser dada indicação em contrário é que o UML faz uso desta restrição, o que o torna mais simples, pois, evita-se a utilização de mais um elemento. Pelo que se atribui ao critério de simplicidade a pontuação máxima ao UML e a pontuação de 0,5 ao ORM.

Código: 4.3	**Parâmetro:** Frequência	**Tipo:** Restrições	
Explicação: Garante o número vezes que pode ocorrer um determinado valor, ou conjunto de valores numa relação.			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	0,5

f) O UML define esta restrição quando a está a definir a multiplicidade quer das entidades quer dos atributos, não sendo necessária a utilização de uma notação própria para esta

restrição, o que torna o UML mais simples sendo-lhe dada a pontuação máxima e ao ORM 0,5 pontos.

Por exemplo no sistema "Leilão On-line", se um artigo em leilão só pudesse receber duas licitações do mesmo utilizador teria que se aplicar esta restrição, no UML era indicado através do intervalo da multiplicidade da associação, como mostra Figura 132. No ORM teria de se utilizar mais elemento sobre o predicado, como mostra a Figura 131.

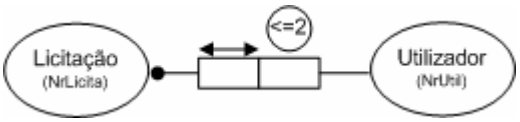


Figura 131 – ORM – Restrição de Frequência

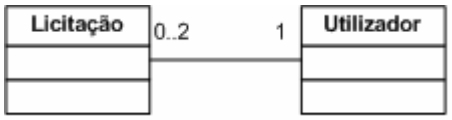


Figura 132 – UML - Restrição de Frequencia

Código: 4.4	**Parâmetro:** Igualdade	**Tipo:** Restrições	
Explicação: Garante que quando uma determinada regra é respeitada uma outra também tem o de ser.			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	0,5	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

a) O ORM tem notação gráfica para a restrição de igualdade, tanto para as restrições entre entidades, como para as restrições entre entidades e valores (Figura 133). O UML, por sua vez, só tem notação gráfica para as relações entre entidades, tendo que recorrer ao OCL (linguagem de expressões matemáticas) para expressar esta restrição entre atributos da própria classe (Figura 134). Por isto, se considera o ORM graficamente mais completo, sendo-lhe atribuída a pontuação máxima, enquanto ao UML é lhe atribuída a pontuação de 0,5.

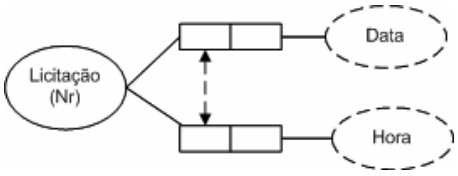


Figura 133 – ORM - Restrição Igualdade

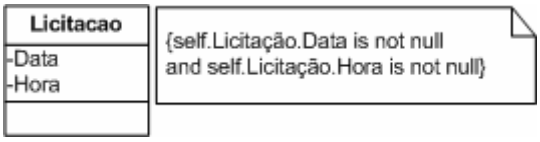


Figura 134 – UML - Restrição Igualdade

Código: 4.5	**Parâmetro:** Subconjunto	**Tipo:** Restrições	
Explicação: Garante que a população de determinada regra tenha que ser um subconjunto da população de outra regra			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	0,5	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

a) O ORM tem notação gráfica para a restrição de subconjunto, tanto para as restrições entre entidades, como para as restrições entre entidades e valores (Figura 135). O UML, por sua vez, só tem notação gráfica para as relações entre entidades, tendo que recorrer ao OCL (linguagem de expressões matemáticas) para expressar esta restrição entre entidades e atributos (Figura 136). Por isto, se considera o ORM graficamente mais completo, sendo-lhe atribuída a pontuação máxima, enquanto ao UML é lhe atribuída a pontuação de 0,5.

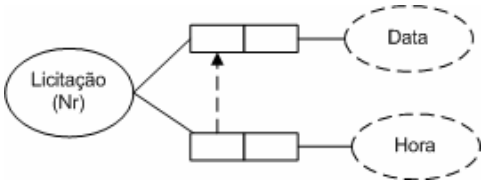


Figura 135 ORM- Restrição Subconjunto

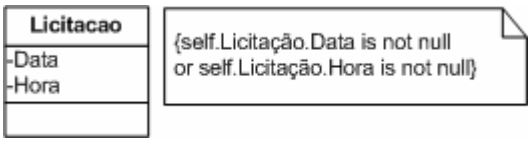


Figura 136 – UML - Restrição Subconjunto

Código: 4.6	**Parâmetro:** Exclusão	**Tipo:** Restrições	
Explicação: Garante que a população de uma determinada regra seja a disjunção da população de outra regra.			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	0,5	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

a) O ORM tem notação gráfica para a restrição de exclusão, tanto para as restrições entre entidades, como para as restrições entre entidades e valores (Figura 137). O UML, por sua vez, só tem notação gráfica para as relações entre entidades, tendo que recorrer ao OCL (linguagem de expressões matemáticas) para expressar esta restrição entre entidades e atributos (Figura 138). Por isto, se considera o ORM graficamente mais completo, sendo-lhe atribuída a pontuação máxima, enquanto ao UML é lhe atribuída a pontuação de 0,5.

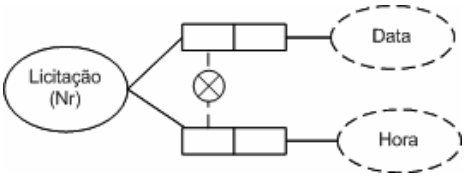


Figura 137 – ORM - Restrição Exclusão

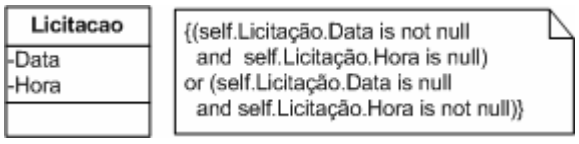


Figura 138 – UML - Restrição Exclusão

Código: 4.7	**Parâmetro:** Anel	**Tipo:** Restrição	
Explicação: Garante que uma regra seja aplicada sobre o próprio objecto.			
Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	0,5	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

a) A representação gráfica da restrição em anel no ORM permite a definição do tipo de restrição (Figura 139), enquanto no UML esta restrição não existe, sendo apenas considerada como uma associação sobre a mesma classe (Figura 140), por isso se considera o UML menos completo do que o ORM na representação desta restrição, sendo atribuída a pontuação máxima ao ORM e a pontuação de 0,5 ao UML.

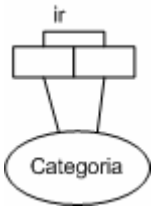


Figura 139 – ORM – Restrição Anel

Figura 140 – UML – Associação reflexiva

Código: 4.8	**Parâmetro:** Valor	**Tipo:** Restrição
Explicação: Garante que só determinados valores, ou conjunto de valores, podem ser considerados para um objecto tipo valor.		

Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	1	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

a) A restrição de valor tem a mesma representação quer no ORM quer no UML, ambos representam esta restrição colocando entre parêntesis ({}) os valores, ou intervalos de valores, que podem ser assumidos pelo objecto, no caso do ORM ou pelo atributo no caso do UML, conforme se ilustra nas figuras abaixo. Por esta razão se atribui a mesma pontuação às duas linguagens.

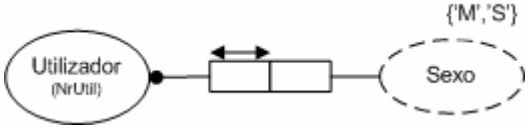


Figura 141 – ORM – Restrição Valor

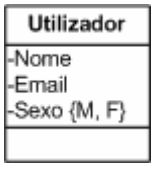


Figura 142 – UML - Restrição Valor

Código: 4.9	**Parâmetro:** Indexe	**Tipo:** Restrição
Explicação: Garante o aumento do desempenho de uma base de dados no acesso à informação.		

Dimensão	**Critério**	**UML**	**ORM**
Qualidade Semântica	a) Completude	0,5	1
	b) Correcção	1	1
	c) Univocidade	1	1
	d) Minimalismo	1	1
	e) Sentido	1	1
Qualidade Sintáctica	f) Clareza	1	1
	g) Simplicidade	1	1

a) O UML não tem nenhuma representação gráfica para a restrição de indexe, representa-a colocando a letra "I" à frente da identificação do atributo e só contempla esta representação no caso dos atributos (Figura 144), o que o torna menos completo em relação ao ORM, pois este possibilita a representação desta restrição em relações entre objectos do tipo entidade e em relações entre objectos do tipo entidade com objectos do tipo valor (Figura 143). Por esta razão se atribui, no critério de completude, a pontuação máxima ao ORM e a pontuação de 0,5 ao UML.

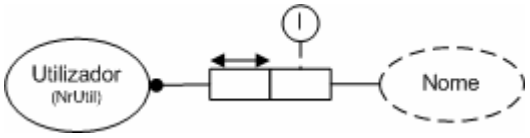


Figura 143 – ORM – Restrição Indexe

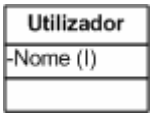


Figura 144 – UML - Restrição Indexe

7.5. Classificação final

A tabela, abaixo apresentada, indica a classificação ponderada aplicada as pontuações totais, nas tabelas em adenda no Anexo I:

$$Classificação = \frac{PontuaçãoTotalCritério * PesoCritério}{NúmeroParâmetros} \quad (5)$$

Dimensão	Critério	Objectos (2)		Relações (4)		Associações (4)		Restrições (9)		Média Ponderada	
		UML	ORM	UML	ORM	UML	ORM	UML	ORM	UML	ORM
Qualidade Semântica	**a) Completude** (valor máximo 8)	6,0	8,0	5,5	8,0	8,0	6,0	5,8	8,0	6,3	7,5
	b) Correcção (valor máximo 2)	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0
	c)Univocidade (valor máximo 2)	2,0	2,0	2,0	2,0	2,0	1,3	2,0	2,0	2,0	1,8
	d) Minimalismo (valor máximo 2)	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0
	e) Sentido (valor máximo 2)	2,0	2,0	2,0	2,0	2,0	2,0	2,0	1,9	2,0	2,0
Qualidade Sintáctica	**f) Clareza** (valor máximo 5)	5,0	3,8	4,4	5,0	4,4	5,0	5,0	5,0	4,7	4,7
	g) Simplicidade (valor máximo 3)	3,0	1,5	3,0	2,6	3,0	3,0	3,0	2,5	3,0	2,4
Média Ponderada		**3,1**	**3,0**	**3,0**	**3,4**	**3,3**	**3,0**	**3,1**	**3,3**	**3,1**	**3,2**

Tabela 8 – Classificação final

Da análise desta tabela resultam as seguintes observações, relativamente, à avaliação dos critérios:

— O UML perde pontuação no critério da completude, no que diz respeito à representação dos objectos, das relações e, principalmente, das restrições. Isto porque o UML não contempla a representação gráfica das restrições entre atributos e entidades. Assim, se verifica que o ORM é uma linguagem mais completa para a representação do modelo conceptual de dados, permitindo uma maior capacidade em conter todas as declarações que são correctas e relevantes para o domínio.

— Quanto à validade ambas as linguagens são muito semelhantes, apenas, o ORM perdeu umas décimas, em relação ao UML, no critério da univocidade pelo facto de não ter representação para os casos especiais da associação. No entanto, verifica-se que todas as declarações efectuadas pelas linguagens são correctas e relevantes.

— Em relação, ao critério clareza ambas as linguagens ficam com a mesma avaliação, o UML ganha em relação à clareza da representação dos tipos de objectos mas perde em relação à clareza na representação gráfica das relações e das associações.

— Por último, verifica-se que o UML é uma linguagem mais simples do que o ORM. Isto porque o UML contém um menor número de elementos para a representação do modelo conceptual de dados permitindo que o modelo seja facilmente compreendido.

Resumidamente, conclui-se que a linguagem UML consegue uma representação gráfica do modelo de dados conceptual com uma maior qualidade sintáctica e que, por sua vez, a linguagem ORM consegue uma representação gráfica com maior qualidade semântica, como ilustra na tabela abaixo, com as médias ponderadas por dimensão.

	UML	**ORM**
Qualidade Semântica	2,9	3,1
Qualidade Sintáctica	3,8	3,5

Tabela 9 - Resumo da classificação

8. Conclusões

Neste capítulo é apresentada uma síntese do trabalho realizado, bem como as respectivas conclusões e perspectivas de trabalho futuro.

Este trabalho começou por mostrar a importância dos modelos de dados no desenvolvimento dos sistemas de informação, assim como, a importância de existirem modelos completos e expressivos capazes de representar todos os detalhes relevantes no domínio da aplicação. Os modelos de dados são definidos por de linguagens próprias, designadas por linguagens de modelação de dados, que podem ser mais ou menos expressivas de acordo com a sua notação e com a sua semântica.

O ORM e o UML (diagrama de classes) são duas linguagens que permitem efectuar a modelação de dados, dos sistemas informação, que apesar de terem características distintas tem a mesma finalidade, pois, ambas estão vocacionadas para representar estruturas de dados a um nível de abstracção acima dos detalhes de implementação.

Por forma avaliar a qualidade destas linguagens, na representação e na completude da definição do modelo conceptual de dados, efectuou-se uma comparação entre a notação e a semântica das duas linguagens, tendo por base um caso prático de modelação de dados sobre um sistema de gestão de "Leilões On-line". Com este caso prático, foi possível identificar as principais diferenças entre as linguagens para, posteriormente, se proceder a uma avaliação da qualidade das linguagens ao nível da sua semântica e da sua sintaxe.

Esta avaliação teve por base os critérios definidos por Patrício (2003), baseados nos autores Lindland, Sindre e Solvberg (1994), para os quais se criou uma grelha de avaliação onde se aplicou o método de PHIMA, do autor Sousa (1998), para a distribuição das ponderações atribuídas.

Os critérios a avaliar foram a completude e a validade para a qualidade semântica, a clareza e a simplicidade para a qualidade sintáctica, onde se concluiu o seguinte:

— Quanto ao critério da completude que o ORM é uma linguagem mais completa pois contempla uma maior número de elementos no que diz respeito, principalmente, à representação das restrições entre objectos;

— Quanto ao critério da validade ambas as são muito semelhantes, no entanto, o facto da linguagem UML incluir representação gráfica para os casos especiais da associação torna-a uma linguagem mais unívoca e, consequentemente, mais válida;

— Quanto ao critério da clareza ambas as linguagens ficam equiparadas pois os seus elementos permitem uma identificação clara dos objectos num modelo conceptual de dados;

— E por último, quanto à simplicidade a linguagem UML é nitidamente uma linguagem mais simples que a linguagem ORM, pois utiliza menos elementos para a representação do modelo conceptual de dados, uma vez, que nem sempre recorre aos diagramas gráficos mas sim aos textuais.

Quando obtidas as médias dimensão, conclui-se que a linguagem ORM dispõe de uma qualidade semântica ligeiramente superior à linguagem UML, mas em contrapartida a linguagem UML consegue ter uma maior qualidade sintáctica em relação à linguagem ORM.

Quando obtidas as médias globais, verifica-se que as duas linguagens ficam com uma pontuação muito idêntica, com apenas 0.1 pontos de diferença, ficando o ORM com a pontuação de 3.2 e o UML com a pontuação de 3.1. O que significa que as linguagens estão muito próximas, no que respeita à qualidade na representação e na completude da definição do modelo conceptual de dados.

Para confirmar a proximidade da qualidade das linguagens, na geração do modelo de dados foram utilizadas as ferramentas PowerDesigner 11 da Sybase e Visio Entreprise Arcthitects da Microsoft para a criação automática da DDL a partir dos modelos UML e ORM respectivamente, conforme apresentado nos Anexos IV e V. Onde se conclui que ambas linguagens permitiram a geração da mesma estrutura de dados para a implementação do sistema de gestão de “Leilões On-Line”.

Deste modo, foi possível concluir que ambas as linguagens permitem a definição do modelo conceptual de dados com o mesmo rigor e automatismo sobre o domínio da aplicação, quando incorporadas em ferramentas CASE.

No futuro, seria necessário realizar um levantamento sobre quais as ferramentas CASE que disponibilizam estas linguagens e medir grau de satisfação dos seus utilizadores, por forma a efectuar-se uma análise mais aprofundada sobre as suas vantagens e desvantagens da sua utilização na definição do modelo conceptual de dados. Podendo, assim, contribuir para escolha das ferramentas CASE a utilizar, caso se opte pelas linguagens UML ou ORM.

9. Referências Bibliograficas

AZEVEDO, ANA [*et al*] (2002) – *Desenho e Implementação de Bases de Dados com Microsoft Access X,* Lisboa, Edições Centro Atlântico, 2002.

BOOCH, GRADY [*et al*] (1999) - *The Unified Modeling Language User Guide*, Addison Wesley, 1999.

HALPIN, TERRY [*et al*] - *Data modeling in UML and ORM: a comparison*, [consult. 20 Junho 2007]. Disponível em WWW: http://www.orm.net/pdf/JDM99.pdf.

HALPIN, TERRY - *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*, Morgan Kaufman Publishers, 2001.

HALPIN, TERRY [*et al*] - *Database Modeling with Microsoft Visio for Enterprise Architects*, Morgan Kaufman Publishers, 2003.

HALPIN, TERRY (2006) - *Object-Role Modeling (ORM/NIAM) in Handbook on Architectures of Information Systems*, Günter (Eds.), 2006. [Consult. 13 de Outubro 2007]. Disponível em WWW: http://www.orm.net/pdf/springer.pdf

LINDLAND, ODD IVAR; SINDRE, GUTTORM; SOLVBERG, ARNE (1994)– *Understanding quality inconceptual modeling*. "IEEE Software", (Mar 1994) 42-49.

NUNES, MAURO [*et al*] (2007) – *Fundamental de UML*, Lisboa, Editora FCA, 2007.

OMG Unified Modeling Language Infrastructure, V2.1.2, OMG, 2007. [Consult. 10 de Abril 2008]. Disponível em WWW: http://www.omg.org/docs/formal/07-11-04.pdf

PATRÍCIO, HELENA (2003) – *Análise comparativa da aplicação do modelo relacional e do formalismo RDF à modelação de dados legislativos*, Dissertação, ISCTE, 2003.

RAMOS, PEDRO (2006) – Desenhar bases de dados com UML, Lisboa, Edições Sílabo, 2006.

SILVA, ALBERTO [*et al*] (2001) – UML, Metodologias e Ferramentas CASE, Lisboa, Edições Centro Atlântico, 2001.

SOUSA, IVO (1998) – *PHIMA: uma visão da presença na Internet,* Sistemas de informação: revista da Associação Portuguesa de Sistemas de Informação, 1998.

STANLEY D. BLUM (1995) - *A Primer on Object Role Modeling*, University of California, 1995, [Consult. 29 Junho 2007]. Disponível em WWW: http://www.mip.berkeley.edu/mvz/cis/primer.pdf

Referências WWW:

OBJECT ROLE MODELING, *The official site for conceptual data modelling*, [consult. 20-07-2007] Disponível em http://www.orm.net.

THE ORM FOUNDATION, [consult. 20-07-2007] Disponível em http://www.ormfoundation.org.

OBJECJT ROLE MODELING, *Portal focuses on Object-Role Modeling*, [consult. 20-07-2007] Disponível em http://www.objectrolemodeling.com.

OBJECT MANAGEMENT GROUP, *Unified Modeling Language 2.0 (UML)*, [consult. 20-07-2007] Disponível em http://www.uml.org/

ANEXO I – Tabelas de Classificação

Objectos

Dimensão	Critério	1.1		1.2		Total	
		UML	ORM	UML	ORM	UML	ORM
Qualidade Semântica	**a) Completude** (valor máximo 8)	0,50	1	1	1	1,5	2
	b) Correcção (valor máximo 2)	1	1	1	1	2	2
	c)Univocidade (valor máximo 2)	1	1	1	1	2	2
	d) Minimalismo (valor máximo 2)	1	1	1	1	2	2
	e) Sentido (valor máximo 2)	1	1	1	1	2	2
Qualidade Sintáctica	**f) Clareza** (valor máximo 5)	1	1	1	0,5	2	1,5
	g) Simplicidade (valor máximo 3)	1	0,5	1	0,5	2	1
TOTAL						**13,5**	**12,5**

Relações

Dimensão	Critério	2.1		2.2		2.3		2.4		Total	
		UML	ORM	UML	ORM	UML	ORM	UML	ORM	UML	ORM
Qualidade Semântica	**a) Completude** (valor máximo 8)	0,25	1	0,5	1	1	1	1	1	2,75	4
	b) Correcção (valor máximo 2)	1	1	1	1	1	1	1	1	4	4
	c)Univocidade (valor máximo 2)	1	1	1	1	1	1	1	1	4	4
	d) Minimalismo (valor máximo 2)	1	1	1	1	1	1	1	1	4	4
	e) Sentido (valor máximo 2)	1	1	1	1	1	1	1	1	4	4
Qualidade Sintáctica	**f) Clareza** (valor máximo 5)	1	1	1	1	1	1	0,5	1	3,5	4
	g) Simplicidade (valor máximo 3)	1	1	1	1	1	0,50	1	1	4	3
TOTAL										**26,25**	**27,5**

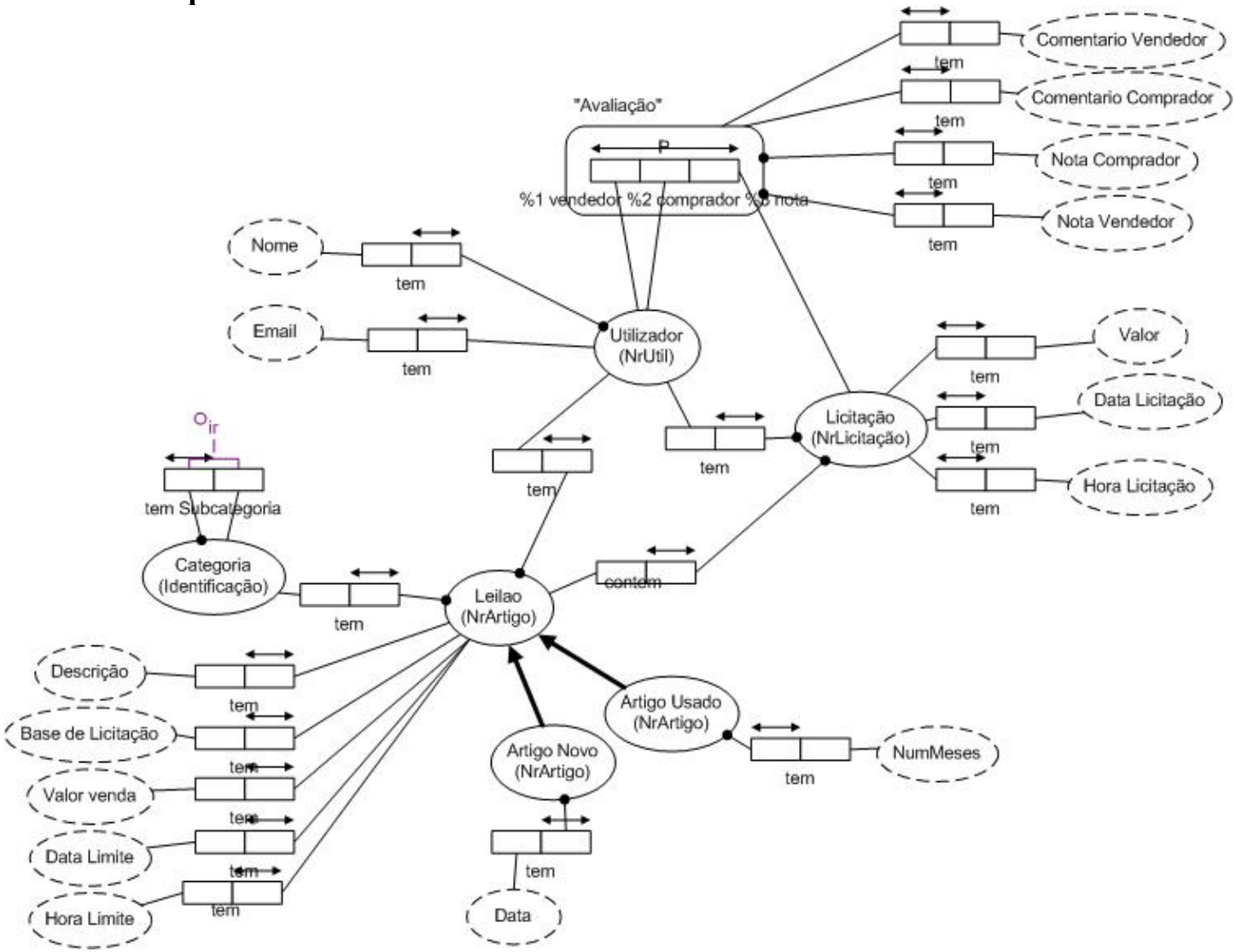
Associações

Dimensão	Critério	3.1		3.2		3.3		3.4		Total	
		UML	ORM	UML	ORM	UML	ORM	UML	ORM	UML	ORM
Qualidade Semântica	**a) Completude** (valor máximo 8)	1	1	1	0,5	1	0,5	1	1	4	3
	b) Correcção (valor máximo 2)	1	1	1	1	1	1	1	1	4	4
	c)Univocidade (valor máximo 2)	1	1	1	0,25	1	0,25	1	1	4	2,5
	d) Minimalismo (valor máximo 2)	1	1	1	1	1	1	1	1	4	4
	e) Sentido (valor máximo 2)	1	1	1	1	1	1	1	1	4	4
Qualidade Sintáctica	**f) Clareza** (valor máximo 5)	0,50	1	1	1	1	1	1	1	3,5	4
	g) Simplicidade (valor máximo 3)	1	1	1	1	1	1	1	1	4	4
TOTAL										**27,5**	**25,5**

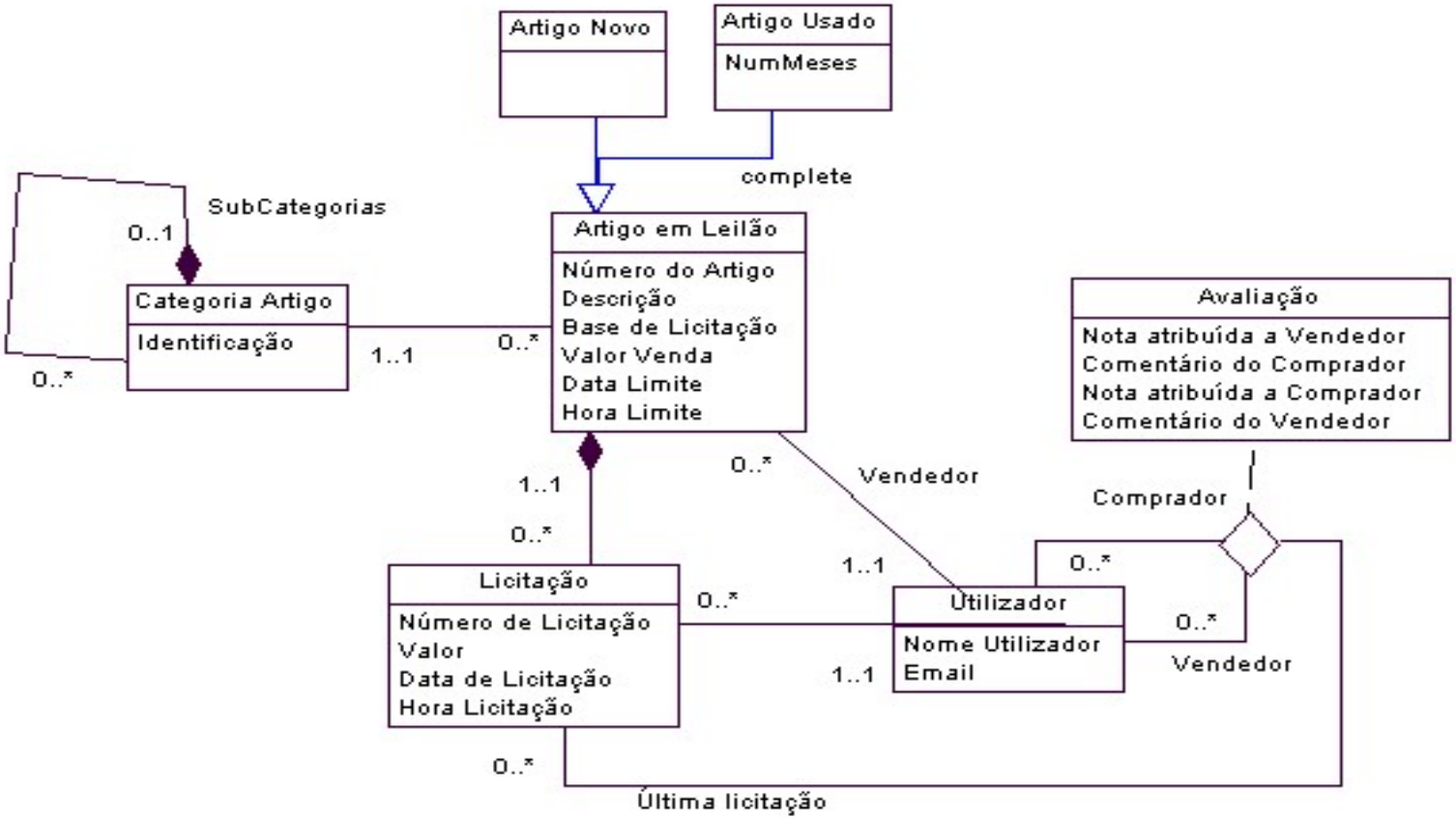
Restrições

Dimensão	Critério	4.1		4.2		4.3		4.4		4.5		4.6		4.7		4.8		4.9		Média Ponderada	
		UML	ORM	UML	ORM	UML	ORM	UML	ORM	UML	ORM	UML	ORM	UML	ORM	UML	ORM	UML	ORM	UML	ORM
Qualidade Semântica	**a) Completude** (valor máximo 8)	1	1	1	1	1	1	0,5	1	0,5	1	0,5	1	0,5	1	1	1	0,5	1	6,5	9
	b) Correcção (valor máximo 2)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	9	9
	c)Univocidade (valor máximo 2)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	9	9
	d) Minimalismo (valor máximo 2)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	9	9
	e) Sentido (valor máximo 2)	1	1	1	0,5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	9	8,5
Qualidade Sintáctica	**f) Clareza** (valor máximo 5)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	9	9
	g) Simplicidade (valor máximo 3)	1	0,50	1	0,5	1	0,5	1	1	1	1	1	1	1	1	1	1	1	1	9	7,5
TOTAL																				**60,5**	**61**

ANEXO II – Modelo conceptual de dados ORM



ANEXO III - Modelo conceptual de dados UML (Diagrama de Classes)



ANEXO IV – DDL gerado a partir do modelo ORM

```
/*    This SQL DDL script was generated by Microsoft Visual Studio
(Release Date: LOCAL BUILD). */

/*    Driver Used : Microsoft Visual Studio - Microsoft SQL Server
Driver.                    */
/*    Document    : D:\TESE\CASO PRÁTICO - LEILÃO\LEILAO PROJECTO.VSD.
*/
/*    Time Created: 12 de August de 2008 21:36.
*/
/*    Operation   : From Visio Generate Wizard.
*/
/*    Connected data source : No connection.
*/
/*    Connected server      : No connection.
*/
/*    Connected database    : Not applicable.
*/



/* Create Leilao On-Line database.
*/
use master

go

create database "Leilao On-Line"

go

use "Leilao On-Line"

go

/* Create new table "Licitação".
*/
/* "Licitação" : Table of Licitação
*/
/*    "Licitação" : NrLicitação identifies Licitação
*/
/*    "Leilao contem NrArtigo" : Leilao contem Licitação
*/
/*    "Tem Hora Licitação" : Licitação tem Hora Licitação
*/
/*    "Tem Data Licitação" : Licitação tem Data Licitação
*/
/*    "Tem Valor" : Licitação tem Valor
*/
/*    "Utilizador tem NrUtil" : Utilizador tem Licitação
*/
create table "Licitação" (
      "Licitação" char(10) not null,
      "Leilao contem NrArtigo" char(10) not null,
      "Tem Hora Licitação" smalldatetime null,
      "Tem Data Licitação" smalldatetime null,
      "Tem Valor" smallint null,
```

```
      "Utilizador tem NrUtil" char(10) not null)

go

alter table "Licitação"
      add constraint "Licitação_PK" primary key ("Licitação")


go

/* Create new table "Categoria".
*/
/* "Categoria" : Table of Categoria
*/
/*    "Categoria Identificação" : Identificação identifies Categoria
*/
/*    "Tem Subcategoria Categoria Identificação" : Categoria tem
Subcategoria Categoria         */
create table "Categoria" (
      "Categoria Identificação" char(10) not null,
      "Tem Subcategoria Categoria Identificação" char(10) not null)

go

/* Create table level checks for table Categoria.
*/
alter table "Categoria" add constraint Categoria_ring check ( "Categoria
Identificação" <> "Tem Subcategoria Categoria Identificação" )

go

alter table "Categoria"
      add constraint "Categoria_PK" primary key ("Categoria
Identificação")


go

/* Create new table "Utilizador".
*/
/* "Utilizador" : Table of Utilizador
*/
/*    "Utilizador NrUtil" : NrUtil identifies Utilizador
*/
/*    "Nome tem" : Nome tem Utilizador
*/
/*    "Email tem" : Email tem Utilizador
*/
create table "Utilizador" (
      "Utilizador NrUtil" char(10) not null,
      "Nome tem" char(10) not null,
      "Email tem" char(10) null)

go

alter table "Utilizador"
      add constraint "Utilizador_PK" primary key ("Utilizador NrUtil")


go
```

```
/* Create new table "Leilao".
*/
/* "Leilao" : Table of Leilao
*/
/*    "Utilizador tem NrUtil" : Utilizador tem Leilao
*/
/*    "Leilao NrArtigo" : NrArtigo identifies Leilao
*/
/*    "Categoria tem Identificação" : Categoria tem Leilao
*/
/*    "Descrição tem" : Descrição tem Leilao
*/
/*    "Base de Licitação tem" : Base de Licitação tem Leilao
*/
/*    "Valor venda tem" : Valor venda tem Leilao
*/
/*    "Data Limite tem" : Data Limite tem Leilao
*/
/*    "Hora Limite tem" : Hora Limite tem Leilao
*/
create table "Leilao" (
      "Utilizador tem NrUtil" char(10) not null,
      "Leilao NrArtigo" char(10) not null,
      "Categoria tem Identificação" char(10) not null,
      "Descrição tem" char(10) null,
      "Base de Licitação tem" char(10) null,
      "Valor venda tem" smallint null,
      "Data Limite tem" smalldatetime null,
      "Hora Limite tem" smalldatetime null)

go

alter table "Leilao"
      add constraint "Leilao_PK" primary key ("Leilao NrArtigo")


go

/* Create new table "Avaliação".
*/
/* "Avaliação" : Table of Avaliação
*/
/*    "Utilizador vendedorUtilizador NrUtil" : Role one (Utilizador) of
fact:                    */
/*          Role one (Utilizador) of fact:
*/
/*          Role two (NrUtil) of fact: Utilizador is identified by
*/
/*    "Vendedor Utilizador compradorUtilizador NrUtil" : Role two
(Utilizador) of fact: Utilizador vendedor  */
/*          Role two (Utilizador) of fact: Utilizador vendedor
*/
/*          Role two (NrUtil) of fact: Utilizador is identified by
*/
/*    "Tem Nota Comprador" : Avaliação tem Nota Comprador
*/
/*    "Tem Comentario Vendedor" : Avaliação tem Comentario Vendedor
*/
/*    "Tem Comentario Comprador" : Avaliação tem Comentario Comprador
*/
```

```
/*    "Tem Nota Vendedor" : Avaliação tem Nota Vendedor
*/
/*    "Licitação" : Role three (Licitação) of fact: Utilizador vendedor
Utilizador comprador    */
/*          Role three (Licitação) of fact: Utilizador vendedor
Utilizador comprador                  */
/*          Role two (NrLicitação) of fact: Licitação is identified by
*/
create table "Avaliação" (
      "Utilizador vendedorUtilizador NrUtil" char(10) not null,
      "Vendedor Utilizador compradorUtilizador NrUtil" char(10) not null,
      "Tem Nota Comprador" smallint not null,
      "Tem Comentario Vendedor" smallint null,
      "Tem Comentario Comprador" smallint null,
      "Tem Nota Vendedor" smallint not null,
      "Licitação" char(10) not null)

go

alter table "Avaliação"
      add constraint "Avaliação_PK" primary key ("Utilizador
vendedorUtilizador NrUtil", "Vendedor Utilizador compradorUtilizador
NrUtil", "Licitação")


go

/* Create new table "Artigo Usado".
*/
/* "Artigo Usado" : Table of Artigo Usado
*/
/*    "Tem NumMeses" : Artigo Usado tem NumMeses
*/
/*    "Artigo Usado NrArtigo" : Artigo Usado is identified by NrArtigo
*/
create table "Artigo Usado" (
      "Tem NumMeses" char(10) not null,
      "Artigo Usado NrArtigo" char(10) not null)

go

alter table "Artigo Usado"
      add constraint "Artigo Usado_PK" primary key ("Artigo Usado
NrArtigo")


go

/* Create new table "Artigo Novo".
*/
/* "Artigo Novo" : Table of Artigo Novo
*/
/*    "Data tem" : Data tem Artigo Novo
*/
/*    "Artigo Novo NrArtigo" : Artigo Novo is identified by NrArtigo
*/
create table "Artigo Novo" (
      "Data tem" char(10) not null,
      "Artigo Novo NrArtigo" char(10) not null)

go
```

```
alter table "Artigo Novo"
      add constraint "Artigo Novo_PK" primary key ("Artigo Novo
NrArtigo")


go

/* Add foreign key constraints to table "Licitação".
*/
alter table "Licitação"
      add constraint "Leilao_Licitação_FK1" foreign key (
            "Leilao contem NrArtigo")
       references "Leilao" (
            "Leilao NrArtigo")

go

alter table "Licitação"
      add constraint "Utilizador_Licitação_FK1" foreign key (
            "Utilizador tem NrUtil")
       references "Utilizador" (
            "Utilizador NrUtil")

go

/* Add foreign key constraints to table "Categoria".
*/
alter table "Categoria"
      add constraint "Categoria_Categoria_FK1" foreign key (
            "Tem Subcategoria Categoria Identificação")
       references "Categoria" (
            "Categoria Identificação")

go

/* Add foreign key constraints to table "Leilao".
*/
alter table "Leilao"
      add constraint "Utilizador_Leilao_FK1" foreign key (
            "Utilizador tem NrUtil")
       references "Utilizador" (
            "Utilizador NrUtil")

go

alter table "Leilao"
      add constraint "Categoria_Leilao_FK1" foreign key (
            "Categoria tem Identificação")
       references "Categoria" (
            "Categoria Identificação")

go

/* Add foreign key constraints to table "Avaliação".
*/
alter table "Avaliação"
      add constraint "Utilizador_Avaliação_FK1" foreign key (
            "Utilizador vendedorUtilizador NrUtil")
       references "Utilizador" (
            "Utilizador NrUtil")
```

```
go

alter table "Avaliação"
      add constraint "Utilizador_Avaliação_FK2" foreign key (
            "Vendedor Utilizador compradorUtilizador NrUtil")
       references "Utilizador" (
            "Utilizador NrUtil")

go

alter table "Avaliação"
      add constraint "Licitação_Avaliação_FK1" foreign key (
            "Licitação")
       references "Licitação" (
            "Licitação")

go

/* Add foreign key constraints to table "Artigo Usado".
*/
alter table "Artigo Usado"
      add constraint "Leilao_Artigo Usado_FK1" foreign key (
            "Artigo Usado NrArtigo")
       references "Leilao" (
            "Leilao NrArtigo")

go

/* Add foreign key constraints to table "Artigo Novo".
*/
alter table "Artigo Novo"
      add constraint "Leilao_Artigo Novo_FK1" foreign key (
            "Artigo Novo NrArtigo")
       references "Leilao" (
            "Leilao NrArtigo")

go



/* This is the end of the Microsoft Visual Studio generated SQL DDL
script.                  */
```

ANEXO V - DDL gerado a partir do modelo UML

```
/*==============================================================*/
/* DBMS name:      Microsoft SQL Server 2000                    */
/* Created on:     24-08-2008 23:12:34                          */
/*==============================================================*/


alter table ArtigoLeilao
   drop constraint FK_ARTIGOLE_ASSOCIATI_CATEGORI
go

alter table ArtigoLeilao
   drop constraint FK_ARTIGOLE_ASSOCIATI_UTILIZAD
go

alter table ArtigoNovo
   drop constraint FK_ARTIGONO_GENERALIZ_ARTIGOLE
go

alter table ArtigoUsado
   drop constraint FK_ARTIGOUS_GENERALIZ_ARTIGOLE
go

alter table Avaliacao
   drop constraint FK_AVALIACA_ASSOCIATI_UTILIZAD
go

alter table Avaliacao
   drop constraint FK_VENDEDOR_ASSOCIATI_UTILIZAD
go

alter table Avaliacao
   drop constraint FK_AVALIACA_ASSOCIATI_LICITACA
go

alter table CategoriaArtigo
   drop constraint FK_CATEGORI_ASSOCIATI_CATEGORI
go

alter table Licitacao
   drop constraint FK_LICITACA_ASSOCIATI_ARTIGOLE
go

alter table Licitacao
   drop constraint FK_LICITACA_ASSOCIATI_UTILIZAD
go

if exists (select 1
            from  sysobjects
           where  id = object_id('ArtigoLeilao')
            and   type = 'U')
   drop table ArtigoLeilao
go

if exists (select 1
            from  sysobjects
           where  id = object_id('ArtigoNovo')
```

```
            and   type = 'U')
   drop table ArtigoNovo
go

if exists (select 1
            from  sysobjects
           where  id = object_id('ArtigoUsado')
            and   type = 'U')
   drop table ArtigoUsado
go

if exists (select 1
            from  sysobjects
           where  id = object_id('Avaliacao')
            and   type = 'U')
   drop table Avaliacao
go

if exists (select 1
            from  sysobjects
           where  id = object_id('CategoriaArtigo')
            and   type = 'U')
   drop table CategoriaArtigo
go

if exists (select 1
            from  sysobjects
           where  id = object_id('Licitacao')
            and   type = 'U')
   drop table Licitacao
go

if exists (select 1
            from  sysobjects
           where  id = object_id('Utilizador')
            and   type = 'U')
   drop table Utilizador
go

/*==============================================================*/
/* Table: Leilao                                                */
/*==============================================================*/
create table Leilao (
   NrArtigo             int                 not null,
   identificacao        int                 null,
   IdUtil               int                 null,
   descricao            char(1)             null,
   baseLicitacao        int                 null,
   valorVenda           int                 null,
   dataLimite           datetime            null,
   horaLimite           datetime            null,
   constraint PK_ARTIGOLEILAO primary key  (NrArtigo)
)
go

/*==============================================================*/
/* Table: ArtigoNovo                                            */
/*==============================================================*/
create table ArtigoNovo (
   obser                char(1)             not null,
   NrArtigo             int                 not null,
```

```
   constraint PK_ARTIGONOVO primary key  (NrArtigo)
)
go

/*==============================================================*/
/* Table: ArtigoUsado                                           */
/*==============================================================*/
create table ArtigoUsado (
   numMeses             int                 null,
   NrArtigo             int                 not null,
   constraint PK_ARTIGOUSADO primary key  (NrArtigo)
)
go

/*==============================================================*/
/* Table: Avaliacao                                             */
/*==============================================================*/
create table Avaliacao (
   notaAtribuidaVendedor int                  null,
   comentarioDoComprador int                  null,
   notaAtribuidaComprador int                  null,
   comentarioDoVendedor int                 null,
   IdUtil_Vend          int                 not null,
   IdUtil_Comp          int                 not null,
   NumLicitacao         int                 not null,
   constraint PK_AVALIACAO primary key  (IdUtil_Vend, IdUtil_Comp,
NumLicitacao)
)
go

/*==============================================================*/
/* Table: CategoriaArtigo                                       */
/*==============================================================*/
create table CategoriaArtigo (
   identificacao        int                 not null,
   "Sub-Categoria"      int                 null,
   constraint PK_CATEGORIAARTIGO primary key  (identificacao)
)
go

/*==============================================================*/
/* Table: Licitacao                                             */
/*==============================================================*/
create table Licitacao (
   NumLicitacao         int                 not null,
   IdUtil               int                 null,
   NrArtigo             int                 null,
   numeroLeilao         int                 null,
   valor                int                 null,
   dataLicitacao        datetime            null,
   horaLicitacao        datetime            null,
   constraint PK_LICITACAO primary key  (NumLicitacao)
)
go

/*==============================================================*/
/* Table: Utilizador                                            */
/*==============================================================*/
create table Utilizador (
   nomeUtilizador       char(1)             null,
   Email                char(1)             null,
```

```
   IdUtil               int                  not null,
   constraint PK_UTILIZADOR primary key  (IdUtil)
)
go

alter table Leilao
   add constraint FK_ARTIGOLE_ASSOCIATI_CATEGORI foreign key
(identificacao)
      references CategoriaArtigo (identificacao)
go

alter table Leilao
   add constraint FK_ARTIGOLE_ASSOCIATI_UTILIZAD foreign key (IdUtil)
      references Utilizador (IdUtil)
go

alter table ArtigoNovo
   add constraint FK_ARTIGONO_GENERALIZ_ARTIGOLE foreign key (NrArtigo)
      references Leilao (NrArtigo)
go

alter table ArtigoUsado
   add constraint FK_ARTIGOUS_GENERALIZ_ARTIGOLE foreign key (NrArtigo)
      references Leilao (NrArtigo)
go

alter table Avaliacao
   add constraint FK_AVALIACA_ASSOCIATI_UTILIZAD foreign key
(IdUtil_Comp)
      references Utilizador (IdUtil)
go

alter table Avaliacao
   add constraint FK_VENDEDOR_ASSOCIATI_UTILIZAD foreign key
(IdUtil_Vend)
      references Utilizador (IdUtil)
go

alter table Avaliacao
   add constraint FK_AVALIACA_ASSOCIATI_LICITACA foreign key
(NumLicitacao)
      references Licitacao (NumLicitacao)
go

alter table CategoriaArtigo
   add constraint FK_CATEGORI_ASSOCIATI_CATEGORI foreign key ("Sub-
Categoria")
      references CategoriaArtigo (identificacao)
go

alter table Licitacao
   add constraint FK_LICITACA_ASSOCIATI_ARTIGOLE foreign key (NrArtigo)
      references Leilao (NrArtigo)
go

alter table Licitacao
   add constraint FK_LICITACA_ASSOCIATI_UTILIZAD foreign key (IdUtil)
      references Utilizador (IdUtil)
```