A Outra Face dos Modelos:
Técnicas de visualização para explorar modelos

DISSERTAÇÃO DE MESTRADO

Juan Manuel Jardim Mendes
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

Setembro | 2011

Orientador:

Leonel Domingos Telo Nóbrega

Professor Auxiliar do Centro de Ciências Exatas e da Engenharia

Universidade da Madeira

Abstract

The models are traditionally used in software engineering to document the aspects of design and, in some cases, they are used to generate part or the entire of IT systems they describe. There is still the debate about this kind of approaches and the role and qualities that the models must have in this area of engineering. The existence of models that conform to well-defined modeling languages allow other uses that go beyond the mentioned above. Based on the existing knowledge about data visualization, this dissertation demonstrate the use of visualization techniques that allow to extract information on the models in an innovative perspective, which contributes positively to a better understanding, analysis and validation of models.

Keywords

Visualization

Model

Meta-model

Model-driven

OCL

Query

Resumo

Os modelos são tradicionalmente utilizados na engenharia de *software* para documentar aspetos do desenho e, em alguns casos, como base para a geração de parte ou a totalidade dos sistemas informáticos que descrevem. Embora subsista o debate sobre este tipo de abordagens e sobre o papel e qualidades que os modelos devem possuir nesta área de engenharia, a existência de modelos que estejam em conformidade com linguagens de modelação bem definidas permite outro tipo de utilizações que vão além das anteriormente referidas. Assente no conhecimento existente sobre a visualização de dados, nesta dissertação irá ser demonstrado a utilização de técnicas de visualização que permitem extrair informação sobre os modelos numa perspetiva inovadora e que contribui favoravelmente para uma melhor compreensão, análise e validação dos mesmos.

Palavras-chaves

Visualização

Modelo

Meta-modelo

Model-driven

OCL

Consultas

À Susana

Aos meus irmãos

E em especial aos meus pais

Agradecimentos

O meu agradecimento é dirigido ao Professor Doutor Leonel Domingos Telo Nóbrega, por ter aceitado ser meu orientador neste trabalho, pela confiança depositada, pelos ensinamentos transmitidos desde o início da licenciatura e pelas críticas construtivas efetuadas.

Aos meus amigos que sempre me apoiaram desde muito cedo. A equipa da *ZON Service Engineering* pela ajuda, conselhos e brincadeiras proporcionadas durante a dissertação.

Um agradecimento muito especial à Susana, pelo seu constante apoio, conselhos, suporte e ajuda ao longo da elaboração da dissertação.

À minha família, em especial à aos meus pais e irmãos, pelo seu apoio incondicional ao longo destes anos e pelos seus conselhos e ensinamentos sem os quais não teria sido possível este meu percurso.

Índice

Índice de Figura

Índice de Tabelas

1 Introdução

Nos dias de hoje, o desenvolvimento do *software* não é uma tarefa fácil devido a vários fatores: o planeamento, levantamento de requisitos, escolha e implementação de padrões arquiteturais, de desenho, entre outros [1].

Cada vez mais procura-se otimizar o processo de desenvolvimento devido a sua elevada dificuldade. Por um lado, são utilizados métodos de desenvolvimento, tais como: cascata, espiral, *eXtreme Programing* e, por outro lado, são usados os modelos como ponto-chave no desenvolvimento. Os modelos são tradicionalmente utilizados na engenharia de *software* para documentar aspetos do desenho e, em alguns casos, como base para a geração de parte ou a totalidade dos sistemas informáticos que descrevem, o que permite reduzir, na maior parte dos casos, o tempo de desenvolvimento do software. Esta abordagem tornou-se muito popular com o surgimento do *Unified Modeling Language* (UML) [2].

Existem diferentes abordagens de desenvolvimento do *software* que utilizam os modelos como ponto-chave do desenvolvimento, tais como: O *Model Driven Development* (MDD) [3], o *Model Driven Architecture* (MDA) [4] e o *Model Driven Engineering* (MDE) [5].

No entanto, existem desvantagens na utilização dos modelos, nomeadamente, o fator da mudança do *software*. Com o advento da mudança, é necessário também um esforço constante de atualização dos modelos por forma a manter a coerência com o produto final. Este esforço é tanto maior, quanto mais extensa for a utilização dos modelos. Para além desta desvantagem, os modelos são vistas parciais de todo o sistema o que dificulta a extração e compreensão das informações relevantes, de forma rápida e simples, de todo o sistema, o que muitas vezes dificulta a própria validação dos modelos.

Uma abordagem possível, com vista a uma melhor compreensão do *software* e diminuir o esforço na atualização dos modelos, é aplicar os conceitos de visualização de dados aos modelos. As visualizações permitem ao utilizador uma melhor compreensão sobre os dados que estão sendo apresentados, com uma menor carga cognitiva, permitindo de forma rápida e simples extrair informação sobre os objetos que estão sendo visualizados, neste caso sobre os modelos.

Este trabalho visa descrever uma abordagem que permita a extração de informação que se encontra nos modelos, através de diferentes visualizações, com o objetivo de dar ao utilizador a possibilidade de ter uma melhor compreensão, análise e validação dos modelos.

1.1 Motivação

Esta dissertação, na área da engenharia de *software*, tem por fundamento os seguintes aspetos:

A) A necessidade de poder visualizar a informação contida nos modelos, de uma forma fácil e rápida, a fim de melhorar a capacidade de compreensão, análise e validação dos modelos sem muito esforço.

B) Proporcionar a continuação da ferramenta protótipo de visualização que foi desenvolvida no âmbito do projeto de modelação de processos de uma secção do governo regional [6], através das notações gráficas *Human Activity Modeling* (HAM) [7] e *Business Process Modeling Notation* (BPMN) [8]. Esta ferramenta motivou o desenvolvimento de uma ferramenta mais genérica e extensível.

C) Este trabalho incide sobre o desenvolvimento de uma aplicação caracterizada por seguir uma arquitetura baseada em componentes e *plugins*, sendo implementada com recurso a linguagem *C#* [9] e o sistema gráfico *Windows Presentation Foundation* (WPF) [10].

1.2 Contribuição

A contribuição primordial desta tese prende-se com o desenvolvimento de uma ferramenta que permite aos investigadores extrair informação dos modelos a fim de poder melhorar a compreensão, análise e validação dos mesmos. São ainda salientadas as seguintes contribuições:

- A possibilidade de compreender melhor a informação que está presente nos modelos o que possibilita a utilização de formas inovadoras de lidar e usar modelos no desenvolvimento de *software*.
- A possibilidade de poder extrair informação dos modelos usando uma linguagem de consulta, mais concretamente o *Object Constraint Language* (OCL) [11].
- A possibilidade de criar diferentes tipos de visualizações para melhor compreender aspetos dos modelos.
- A possibilidade de poder criar diferentes componentes, a fim de possibilitar a execução de um conjunto específico de tarefas.

1.3 Organização

Esta dissertação encontra-se organizada da seguinte forma:

- O capítulo 2 "Estado da Arte" apresenta um conjunto de conceitos sobre a visualização, padrões de visualização e arquiteturas usadas nas visualizações. Também é apresentado um conjunto de conceitos associados à modelação e às abordagens que usam a modelação como ponto-chave do desenvolvimento. Por último, irá ser explorado como é que as visualizações podem contribuir para o desenvolvimento de *software* e para o *Model Driven Engineering* (MDE).
- O capítulo 3 "Proposta", apresenta a proposta com a qual pretende-se atender a necessidade cada vez maior de possuir uma ferramenta que permita extrair e visualizar a informação contida nos modelos, com o intuito de melhorar a compreensão e validação do sistema e melhorar a tomada de decisões.
- O capítulo 4 "Especificação", apresenta a especificação da aplicação que irá ser desenvolvida. Neste capítulo o leitor poderá encontrar o diagrama de casos de utilização e de atividades, qual é a arquitetura da aplicação e o *wireframe* que foi utilizado como base para a interface gráfica da aplicação.
- O capítulo 5 "Implementação", expõe como foi realizada a implementação, em termos de quais foram as tecnologias utilizadas, qual é a estrutura geral do projeto e das suas componentes, que bibliotecas foram utilizadas, como é que a ferramenta funciona e que tipos de componentes foram implementadas.
- O capítulo 6 "Caso de estudo", apresenta o caso de estudo que foi realizado com o intuito de saber se a aplicação desenvolvida de facto atinge os objetivos propostos nesta dissertação.
- O capítulo 7 "Conclusões e Perspetivas Futuras", apresenta as conclusões desta dissertação bem como algumas propostas de trabalho futuro a desenvolver.

2 Estado da Arte

2.1 Introdução

Neste capítulo é realizado um levantamento do estado da arte da visualização, do desenvolvimento de *software* através da utilização de modelos e como é que a utilização da visualização pode ajudar ao desenvolvimento de software.

Este capítulo está estruturado da seguinte forma: em primeiro lugar, descreve-se brevemente a importância da visualização para os seres humanos, a sua evolução ao longo do tempo, as variedades de visualização, os passos para criar visualizações, um breve estudo sobre as visualizações, os padrões de desenho das visualizações, as arquiteturas usadas nas visualizações e as principais ferramentas que permitem criar visualizações. Em segundo lugar, descreve-se o conceito de *Model-driven* (MD), as diferentes abordagens que utilizam os modelos como ponto-chave e as linguagens de modelação. Por último, é apresentado a visualização no contexto do MD e a linguagem de consulta OCL.

2.2 A visualização

***Uma imagem vale mais do que mil palavras*. Fred R. Barnard (1927).**

Devido a necessidade natural de comunicar dos seres humanos, tem-se vindo a elaborar, desde o início da nossa história, diversas formas de comunicação e transmissão de informação, sejam elas através da fala, sinais de fumo, pinturas, etc. De acordo com MacGarry [12], "a informação deve ser ordenada, estruturada, ou contida de alguma forma, senão permanecerá amorfa e inutilizável". Isto quer dizer que não basta representar a informação de qualquer maneira, esta tem que ser representada e estruturada de acordo com um "padrão" pré-estabelecido para a utilizar e compreender. Por exemplo, ao ler a seguinte frase: "palavras mil que do mais vale imagem uma", provavelmente a frase está mal construída porque não faz sentido. No entanto, se a regra ou o "padrão" para ler esta frase é que esta seja lida da direita para a esquerda é possível observar que a frase anterior fica "Uma imagem vale mais do que mil palavras" passando a fazer sentido.

Ainda segundo MacGarry [12], "A informação, portanto, deve ter alguma forma de veículo. Este veículo deve possuir um atributo essencial para que possa ser entendido pelo recetor". MacGarry [13] distingue três tipos de veículos: sinais, signos e símbolos. Os sinais estabelecem relações com as ações a serem desenvolvidas pelo recetor. Por sua vez, os signos indicam a presença física de algo ou algum evento relacionado a eles. Por último, os símbolos tendem a possuir significados mais duradouros e constituem-se em representações culturalmente

construídas e reconhecidas por uma comunidade específica. Para além destes três veículos, MacGarry [12] considera um outro veículo de comunicação e transmissão de informação: "a linguagem é o veículo fundamental da comunicação humana". No entanto, a linguagem muitas vezes pode tornar-se um entrave para a comunicação devido às suas características intrínsecas. Portanto, para comunicar informação é preciso um veículo, uma estrutura e uma organização para que a informação não permaneça amorfa e inutilizável e muitas vezes de uma visualização.

A visualização pode ser definida [14] como uma técnica para criar imagens, diagramas ou animações para comunicar uma mensagem. A Visualização através de imagens visuais tem sido uma forma eficaz de comunicar ideias tanto abstratas como concretas desde o início da história do homem. Exemplos da história incluem pinturas rupestres, os hieróglifos egípcios, a geometria grega, e métodos revolucionários de Leonardo da Vinci.

Também a visualização pode ser definida [15] como a representação gráfica da informação, com o objetivo de oferecer ao espectador uma compreensão qualitativa do conteúdo da informação. É também o processo de transformação de objetos, conceitos e números em uma forma que é visível aos olhos humanos. Quando se diz "informação", referimo-nos a dados, processos, relações e ou conceitos.

A frase: "uma imagem vale mais do que mil palavras", de Fred R. Barnard é uma frase que representa o porquê da visualização de dados. Uma visualização efetiva faz uso da nossa capacidade cognitiva para processar a informação visual. A visualização suporta o pensamento visual através de consultas de informação, ou seja, permite-nos de uma forma rápida e simples, extrair informação com o intuito de a poder perceber mais rapidamente. Isto prende-se com facto da apresentação visual de informações aproveitar a grande capacidade do olho humano para detetar informações em fotos e visualizações. A visualização transfere a carga do raciocínio numérico para o raciocínio visual. Assim, a obtenção de informações a partir de imagens é muito mais económica em termos de tempo do que a análise realizada a textos e números [16] [15].

É por esta razão que cada vez mais a visualização de dados está assumindo um papel muito importante na análise empresarial (*Business intelligence*) o que leva a que as visualizações sejam mais importantes para as pessoas que trabalham com dados, para as pessoas que estão ligadas a bolsa e especialmente para os analistas, embora também existam pessoas ligadas a análise empresarial que ignoram esta área que está em constante crescimento [17].

Portanto, a visualização hoje expande-se em varias áreas, tais como: ciência, educação, engenharia, multimídia interativa, medicina, etc. [14].

2.2.1 História da Visualização de dados

Esta secção abordará, sucintamente, a história da visualização de dados, de forma a termos uma melhor perceção da sua importância e o porquê dela ter surgido.

A visualização tem sido uma forma eficaz de comunicar ideias tanto abstratas como concretas desde o início da história do homem. No entanto, é comum pensar que os gráficos estatísticos e a visualização de dados fazem parte da evolução ressente da estatística. Na verdade, a representação gráfica da informação quantitativa tem raízes ainda mais profundas [18].

A primeira tabela de que existe registo foi criada no 2 º século DC, no Egipto, para organizar as informações astronómicas como uma ferramenta para navegação. Uma tabela é basicamente uma representação textual dos dados, mas também usa características visuais tais como: alinhamento, espaços em branco, linhas verticais e/ou horizontais, entre outras, para organizar a informação em colunas e linhas. As tabelas, juntamente com gráficos e os diagramas, são inseridos na categoria de representações de dados chamados de gráficos [17].

A representação visual de dados quantitativos utilizando duas dimensões, ou aquilo a que é designado de gráficos, não surgiu até muito mais tarde, mais precisamente no século 17. René Descartes, o filósofo e matemático francês, provavelmente, mais conhecido pela expressão "Cogito ergo sum" ("Penso, logo existo") inventou este método de representação de dados quantitativos inicialmente, não para a representação de dados, mas sim para a realização de um modelo matemático baseado num sistema de coordenadas. Mais tarde, porém, esta representação foi reconhecida como um meio eficaz para apresentar informações [17]. A Figura 1 ilustra o sistema de coordenadas proposto por René Descartes.

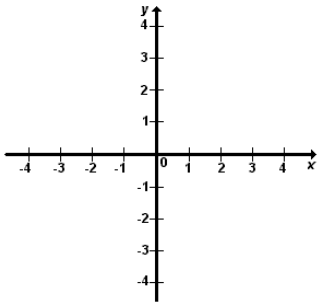

Figura 1. Plano cartesiano.

Após a inovação de Descartes, não foi até o final do século 18 e início do século 19 que muitos dos gráficos que são usados hoje, incluindo gráficos de barras e gráficos circulares, foram inventados ou melhorados dramaticamente por um cientista social escocês chamado William Playfair. Mais de um século se passou antes que o valor destas técnicas torna-se reconhecido ao

ponto de serem introduzidos em cursos académicos, inicialmente na Iowa State University em 1913 [17].

A pessoa que nos apresentou o poder de visualização de dados como um meio de explorar e fazer sentido dos dados foi o professor de estatística John Tukey de Princeton, que em 1977 desenvolveu uma abordagem predominantemente visual para explorar e analisar dados chamada de análise exploratória de dados [17].

Em 1983, Edward Tufte publicou o seu livro revolucionário *The Visual Display of Quantitative Information*, no qual mostrou que havia maneiras eficazes de apresentar dados visualmente. Um ano depois, em 1984, a Apple lançou o primeiro computador popular e acessível que incidiu sobre os gráficos como um modo de interação e exposição. Isso abriu caminho para o uso de visualização de dados como meio de visualização e interação com o computador. Dada a disponibilidade de computadores a preços acessíveis com gráficos poderosos para a época, uma nova especialidade surgiu no mundo académico, a qual foi dado o nome de Visualização de informação. Em 1999, o livro *Readings in Information Visualization:Using Vision to Think* recolhe todo o trabalho realizado na área da visualização de informação e tornando-o acessível para além dos muros da academia [17].

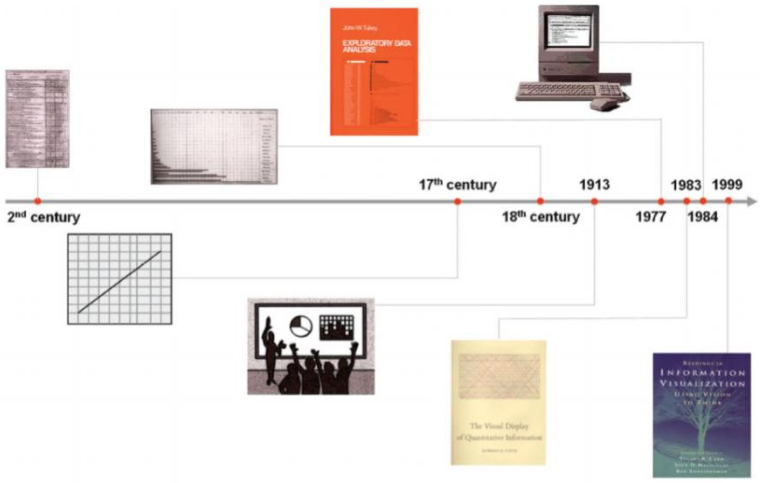

Figura 2. Linha do tempo da Visualização de Dados.

A Figura 2 ilustra, de uma forma bastante resumida, a linha de tempo da história da visualização de dados que foi descrita anteriormente.

No sítio da internet [19] é possível encontrar uma linha de tempo, Figura 3, ainda mais recente que chega até aos nossos dias e esta em constante atualização.

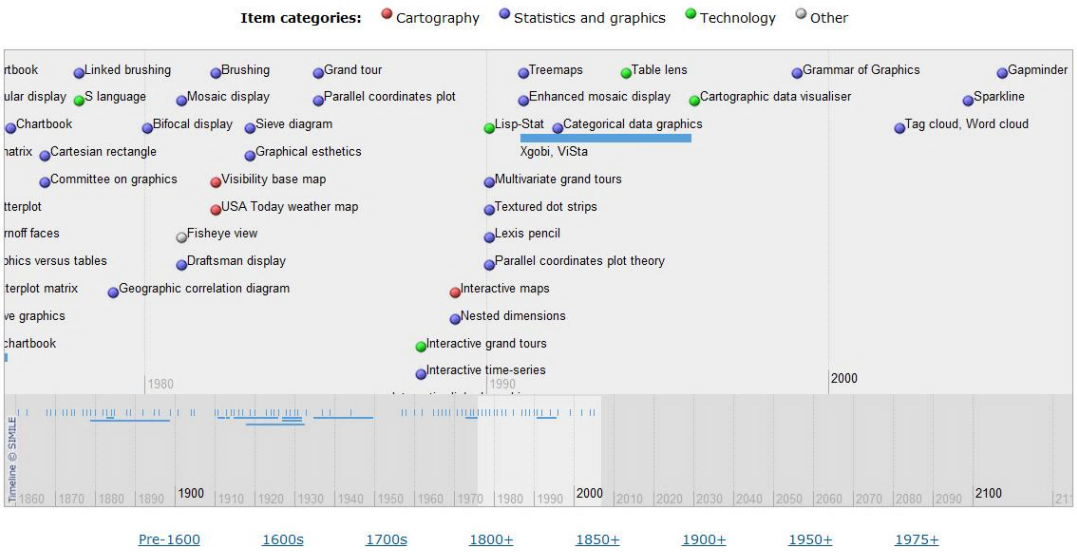

Figura 3. Linha de tempo atual da visualização de dados.

2.2.2 Variedades de visualizações

Esta secção irá abordar as três variedades de visualizações de dados que existem: a visualização de informação, a visualização científica e a visualização de dados, sendo que a visualização de informação e de dados são as duas variedades de visualizações com maior interesse para o trabalho desenvolvido.

O termo "Visualização de informação" é um termo bastante amplo que nos leva a vários tipos de representação da informação, entre eles: tabelas, gráficos, mapas ou até mesmo texto, independentemente da visualização ser estática ou dinâmica. Tudo se resume à capacidade de encontrar informação de forma rápida e fácil sem ter de consultar grandes quantidades de informação.

Mas, hoje em dia o termo "Visualização de Informação" é geralmente aplicado à representação visual de grandes agrupamentos não numéricos de informação, tais como: ficheiros, linhas de código nos sistemas de *software*, base de dados, entre outros [20].

A visualização de informação pressupõe que as representações visuais e técnicas de interação aproveitam a capacidade do olho humano que permitem aos utilizadores ver, explorar e compreender grandes quantidades de informação de uma vez. Portanto, a visualização de informação utiliza o suporte computacional, interativo, para representações visuais de dados abstratos de forma a ampliar o conhecimento através das capacidades do olho humano [21].

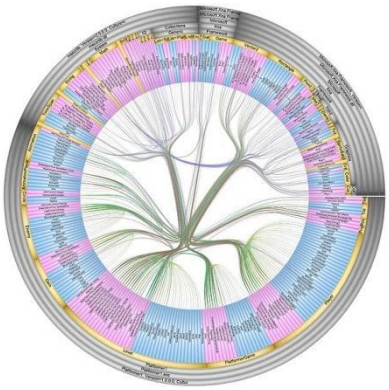

Figura 4. *Solid Software Xplore* (SolidSX).

A Figura 4 mostra um exemplo de uma visualização de informação. Esta visualização é produzida pelo *Solid Software Xplorer* [22], que é uma aplicação que permite visualizar os sistemas de *software* de maior porte.

Por outro lado, o termo "Visualização científica" [23], por vezes também referido por análise de dados visuais, é um termo que também é utilizado para representação gráfica de dados com o intuito de ganhar uma maior compreensão na análise dos dados. Segundo Friendly [20] , esta área da visualização está preocupada principalmente com a visualização dos fenómenos que estão a ser estudados em diversas áreas tais como a arquitetura, meteorologia, medicina, biologia, etc. Portanto, a visualização científica utiliza representações visuais interativas de dados científicos, normalmente cuja base é a física, para ampliar o conhecimento dos investigadores.

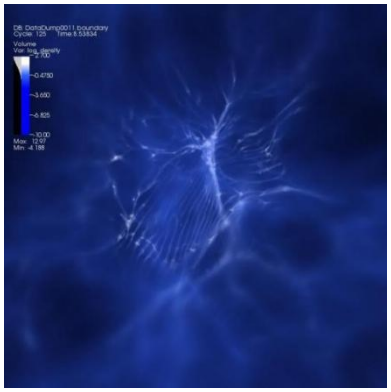

Figura 5. Formação de estrelas.

A Figura 5 ilustra um exemplo de uma visualização científica. Esta visualização é representada pelo gráfico que resulta da aplicação do logaritmo da densidade de gás/ poeira de uma estrela. As regiões de alta densidade estão representadas a branco e as regiões menos densas são as mais obscuras [24].

Por último o termo "Visualização de dados" [25] é utilizado para referir a compreensão das relações entre os números de forma a entender os padrões, tendências e as relações que existem

entre agrupamentos de números. Portanto, a visualização de dados é a representação gráfica da informação. Os gráficos de barras, de dispersão, e os mapas são alguns exemplos simples de visualizações de dados que já são usados há décadas. A tecnologia da informação combina os princípios da visualização com aplicações poderosos e grandes conjuntos de dados para criar imagens sofisticadas e animações. Uma nuvem de "tags" ou palavras, por exemplo, usa o tamanho do texto para indicar a frequência relativa de utilização de um conjunto de termos, Figura 6. Em muitos casos, os dados que "alimentam" uma nuvem provem de milhares de páginas Web, que representam talvez milhões de utilizadores. Toda esta informação está contida numa imagem simples que se pode entender com rapidez e facilidade [25].

Figura 6. Nuvem de palavras.

Por outro lado, a visualização de dados também pode ser definida [26] como um método de computação. Este método transforma os símbolos em formas geométricas, permitindo que os investigadores observem as suas simulações e cálculos. A Visualização oferece um método para ver o invisível. Isso enriquece o processo de descoberta científica e promove profundas descobertas inesperadas. O objetivo da visualização é promover os métodos científicos existentes, fornecendo uma nova visão científica através de métodos visuais.

Um dos maiores desafios na visualização de dados é encontrar representações gerais de dados que possam exibir uma estrutura multidiversificada de mais do que duas variáveis [27].

2.2.3 Passos para criar uma visualização

Esta secção esclarece quais são os passos que normalmente são utilizados para criar visualizações.

O processo de criação de uma visualização normalmente segue o seguinte conjunto de etapas [28]:

- Formulação da questão.
- Recolha de dados.
- Representação visual.

2.2.3.1 Formulação da questão

A formulação da questão, que está a conduzir aquilo que se pretende descrever, não é necessariamente uma etapa que deve ser feita no início da jornada da visualização. Por vezes é necessário possuir um bom entendimento dos dados de forma a formular uma boa questão. No entanto, ter presente uma pergunta ou um conjunto de perguntas pode ser útil quando se pretende reunir e filtrar os dados necessários. Uma das formas de formular a questão é através da utilização de um tópico com o intuito de focar a recolha de dados e refinar a questão formulada a medida que são obtidos mais dados. Por exemplo, a realização dos censos dos EUA é uma tarefa enorme. Este é um bom tema para iniciar a recolha de dados, isto porque este é amplo o suficiente para que surjam diferentes pedaços de dados que podem ajudar a dar um contexto a essa ideia [28]. É possível encontrar dados relevante e criar uma visualização baseada:

- No número de inquéritos preenchidos.
- No número de lápis usados.
- No número de quilómetros percorridos pelos trabalhadores dos sensos.

A questão especifica que irá ser formulada, possui um grande impacto sobre a representação final da visualização. Por exemplo, na questão: "Quanto trabalho é necessário para gravar todas as informações necessárias para o recenseamento?" Pode ser utilizada uma visualização com folhas que cubra uma pequena cidade. Também na questão: "Quantas pessoas são necessárias para realizar os sensos em todo o pais?" Podem ser utilizados ícones de pessoas para representar o número de pessoas necessárias em cada estado. Ambas as questões estão relacionadas com o tema original apresentado anteriormente. A partir destas questões, é possível obter diferentes tipos de dados e consequentemente diferentes formas de representar esses mesmos dados [28].

Quando se está a elaborar uma questão para fins de visualização, é necessário concentrar-se nas questões que são mais centradas nos dados. As questões que começam por: "onde", "quando" ou "quantas vezes", são geralmente bons pontos de partida, porque elas permitem concentrar a recolha de dados dentro de um conjunto especifico de parâmetros, de modo a que seja mais provável encontrar dados com interesse para serem mapeados visualmente. É necessário ter especial cuidado com as perguntas que começam por "porque", devido a que a sua utilização nos leva a uma interpretação mais formal de dados, o que representa um bom sinal [28].

2.2.3.2 Recolha de dados

O processo de criação de uma visualização pode começar a partir de um conjunto de dados que dão origem uma questão específica. Mas a recolha de dados é sempre uma tarefa difícil, porque nem sempre se encontra exatamente os dados que se pretendem. Muitas vezes, é preferível utilizar os dados que já se encontram disponíveis e representa-los da melhor forma possível do que recolher os dados. Existem bons lugares onde é possível começar a recolha de dados. Um dos maiores e mais diversificados repositórios pode ser encontrado no sítio da internet: Data.gov (www.data.gov) [28].

Na recolha de dados, também é necessário ter em conta os tipos de dados e as escalas que são usadas, isto porque quando se pretende visualizar os dados, é necessário ter em conta a natureza ou tipo dos dados. Existem diferentes tipos de dados, sendo que os principais são: os dados quantitativos, qualitativos, descritivos, Fluxo/processo, Hierárquicos e Geográficos. A Tabela 1 apresenta os diferentes tipos de dados que é provável encontrar na recolha dos dados.

Tabela 1. Tipos de dados.

Tipo	Descrição
Quantitativos	Os dados quantitativos [29] são aqueles que podem ser facilmente medidos e registados sob forma numérica. Estes tipos de dados podem ser analisados utilizando métodos estatísticos, como por exemplo a média, e o resultado pode ser exibidos utilizando tabelas, gráficos, etc. No entanto, nem todos os números são mesuráveis, por exemplo, o número do bilhete de identidade é um número mas não é algo com o qual se pode fazer adições ou subtrações.
Qualitativos	Os dados qualitativos [29] são aqueles que são representados por outros meios que não os números. Exemplos deste tipo de dados são o local de nascimento, escola, sexo, etc. Na maioria das situações estes tipos de dados são agrupados de forma a serem apresentados em números ou percentagens, por exemplo, o número de pessoas do sexo masculino numa determinada área.
Descritivos	Estes dados possuem conteúdo que está a descrever um objeto ou um fenómeno. Por exemplo um documento a descrever o propósito de uma empresa.
Fluxo/processo	Este tipo de dados possui elementos temporais ou que possuem a noção de fluxo ao longo do tempo. Por exemplo uma história ou um processo.
Hierárquicos	Este tipo de dados possui elementos que se encontram dentro de uma certa hierarquia. Por exemplo as árvores de dados.
Geográficos	Este tipo de dados possui características geográficas que permitem, por exemplo, localizar uma determina informação num mapa.

Todavia, para além dos tipos de dados que podem ser usados nas visualizações, existem também as escalas que podem ser utilizadas. Estas escalas são: nominal, ordinal, intervalar e rácio. Na Tabela 2, é possível encontrar uma descrição destes tipos de escalas.

Tabela 2. Tipos de escalas.

Tipo	Descrição
Nominal	Na escala nominal, os valores numéricos ou não numéricos, não possuem uma ordem intrínseca. Por exemplo: o tipo de sangue de uma pessoa (O, A, B e AB), a categoria das plantas ou animais [30].
Ordinal	Na escala ordinal, os valores, numéricos ou não, possuem uma ordem intrínseca. Por exemplo: as classificações obtidas no 2º e 3º ciclo do ensino básico (1 a 5), grupos etários (crianças, jovens, adultos e idosos) [30].
Intervalar	Na escala intervalar, os valores numéricos possuem ordem e essa ordem possui significado. Por exemplo: a temperatura média em graus [30].
Rácio	Na escala rácio, os valores numéricos possuem ordem e as diferenças têm significado. O valor zero representa a ausência de uma característica e múltiplos de valores possuem significado. Por exemplo: medidas de comprimento, áreas, pesos ou intervalos de tempo, número de filhos de um casal [30].

A seleção dos dados a apresentar é fundamental para a obtenção de uma visualização eficaz [31]. A customização dos dados, também conhecida como a preparação dos dados, dentro do contexto da visualização da informação, refere-se à forma como os dados são apresentados e estruturados. Existem três operações que podem ser usadas sobre os dados: filtragem da informação, alteração ou renomeação da informação existentes e obtenção de nova informação [16].

A filtragem de informação consiste em remover as informações que não são relevantes com o intuito que estas não influenciem a visualização. Por sua vez, a alteração ou renomeação das informações existentes, consiste na alteração da estrutura de dados sem adicionar ou remover qualquer coisa. Por último, a obtenção de novas informações, é uma tarefa complexa, mas quando realizada corretamente, leva a resultados mais interessantes [16].

Após obter os dados, é possível analisá-los, organizá-los, agrupá-los, ou normaliza-los para que assim seja possível identificar padrões ou extrair as informações específicas que se pretendem representar. Este processo é conhecido como "data munging" e normalmente consiste na manipulação dos dados a fim de encontrar padrões interessantes [28].

2.2.3.3 Representação visual

Depois de se obter os dados e a questão formulada, é necessário decidir a forma como os dados irão ser representados. Esta representação irá ajudar os utilizadores a compreender melhor os dados. No entanto, é necessário ter cuidado com as escalas que são utilizadas, com a semântica e o contexto dos dados, porque irá influenciar a forma como são visualizadas as coisas e pode levar a uma visualização enganosa.

Existem vários tipos de visualizações, como por exemplo: gráficos de barra, gráficos circulares, gráficos de dispersão, grafos, entre outros tipos de visualizações, que são utilizados para

representar os dados. A escolha do tipo de gráfico a usar, depende do tipo de dados que irá ser representado e o que se pretende mostrar. Por outro lado, existem elementos visuais que podem ser utilizados na visualização de dados e influenciam a forma como os dados são apresentados e compreendidos. Estes elementos são: ponto, linha, figura, espaço, cor e textura. A Tabela 3 apresenta uma descrição de cada um dos elementos visuais mencionados anteriormente.

Tabela 3. Elementos visuais.

Elemento	Descrição
Ponto	O ponto é o elemento mais pequeno e mais básico. Este pode variar de tamanho, cor, regularidade ou irregularidade, e pode ser usado como uma unidade ou como um agrupamento que forma uma linha ou uma imagem. Mesmo que exista um ponto numa página em branco, este chama atenção. Quando existem dois pontos, é possível fazer uma conexão entre esses dois pontos e visualizar uma linha, e quando existem três pontos é inevitável interpretar estes pontos com um triângulo porque a mente fornece a conexão entre os pontos [32].
Linha	Uma linha é uma forma que possui largura e comprimento mas não possui profundidade [32]. As linhas podem ser: horizontais, verticais, zig-zag, curvas, retas, diagonais, etc. Estas mostram direção, conduzem os olhos, dão contornos aos objetos, dividem o espaço e permitem comunicar um sentimento ou emoção [33].
Figura	Uma figura é uma área que está contida dentro de uma linha implícita, ou é vista e identificada por causa da cor que possui. As figuras bidimensionais têm duas dimensões, o comprimento e largura, e estas podem ser figuras geométricas, tais como o triângulo, o quadrado, etc., ou podem ser figuras livres. Quando os objetos possuem três dimensões, comprimento, largura e profundidade, são chamados de figuras tridimensionais, como por exemplo, o cubo, a pirâmide, etc. [32].
Espaço	O espaço é a área que uma figura ocupa. O espaço também se refere ao fundo onde é visualizado a figura. O espaço pode ser definido como positivo e negativo. O espaço positivo de um desenho é o que a figura ocupa no desenho. Por outro lado, o espaço negativo é o fundo onde esta o desenho. Ambos espaços são importantes nas visualizações [33].
Cor	A cor é um elemento de representação fantástico para conjuntos enormes de dados. É possível identificar diferentes tonalidades de cor, o que faz com que a cor seja utilizada para representar diferentes tendências, padrões e anomalias em grandes conjuntos de dados [28]. A cor é descrita em termos de tonalidade, valor e intensidade. A tonalidade refere-se ao nome da cor, por exemplo, vermelho ou azul, o valor diz respeito ao nível de claridade ou escuridão que uma tonalidade possui, e, por último, a intensidade se refere ao nível de brilho de uma cor [33].
Textura	A textura é a qualidade da superfície de um objeto. É aquilo que se sente, ou que parece que se sente, quando é tocada uma superfície [33].

Para além dos elementos visuais, também existem os princípios de desenho. Os principais princípios de desenho são: equilíbrio, proporção, perspetiva, enfâse, ritmo, harmonia e unidade [34], [35]. A Tabela 4 possui uma descrição de cada um dos principais princípios de desenho mencionados anteriormente.

Tabela 4. Princípios de desenho.

Princípio	Descrição
Equilíbrio	O Equilíbrio é a sensação de estabilidade [34]. Existem três tipos de equilíbrio: simétrico, radial e assimétrico. O equilíbrio simétrico, ou o equilíbrio formal, é o tipo mais simples. Um objeto que é simetricamente equilibrado tem a mesma forma em ambos os lados. Nossos corpos são um exemplo de equilíbrio formal. Por sua vez, o equilíbrio radial possui um ponto central, por exemplo um pneu, uma *pizza*, etc. Por último, o equilíbrio assimétrico é aquele que cria uma sensação de "peso igual" em ambos os lados, apesar dos lados não terem a mesmas aparência [34].
Proporção	A proporção se refere à relação entre uma parte de um desenho e uma outra parte ou o desenho completo. É uma comparação de tamanhos, formas e quantidades. Por exemplo, a relação entre as medidas verticais e horizontais de um quadro numa parede pode ser agradável, pois os comprimentos desiguais produzem um contraste interessante [34].
Perspetiva	A perspetiva é criada através da disposição dos objetos no espaço bidimensional. Esta acrescenta realismo a uma visualização porque utiliza os tamanhos relativos dos objetos, sobreposição de objetos e a nitidez dos objetos tal e qual como são encontrados na realidade [35].
Enfâse	A enfâse é um princípio de desenho cujo objetivo é chamar a atenção para uma parte do desenho. Existem várias formas de criar enfâse, através da utilização de uma cor de contraste, uma linha diferente ou incomum, uma figura muito grande ou muito pequena, etc.
Ritmo	O ritmo é a repetição do movimento dos elementos visuais tais como: cores, figuras, linhas, espaços e texturas. A variedade é essencial para manter os ritmos excitantes e ativos, e para evitar a monotonia. O Movimento e ritmo trabalham em conjunto para criar o equivalente visual de uma batida musical [34].
Harmonia	A harmonia é um princípio de desenho em que todas as partes de uma imagem se relacionam e se complementam [35].
Unidade	A unidade significa a harmonia de toda a composição. A unidade é a relação entre os elementos visuais que ajudam a que todos os elementos funcionam em conjunto [34].

2.2.4 Estudo das visualizações

Existem diversos tipos de visualizações que podem ser encontradas todos os dias. A maior parte de nós, já trabalhou com diferentes tipos de visualizações, desde gráficos de barras, gráficos circulares, entre outros. Esta secção apresenta um estudo realizado sobre os principais tipos de visualizações.

Um gráfico [36] é uma representação gráfica dos dados, no qual estes são representados por símbolos, tais como barras, linhas ou pontos. Existem diversos tipos de gráficos, entre os quais: Gráficos de linhas, barras, circulares, entre outros. Devido à extensão e às variedades de cada tipo de gráficos, recomenda-se ao leitor a consulta do anexo I.

2.2.4.1 Gráficos de Linhas

Os gráficos de linhas [37] são o tipo de gráficos mais populares porque eles são fáceis de criar e fáceis de entender. Estes gráficos apresentam os dados de uma forma muito clara, permitem visualizar as relações entre os dados e exibem uma mudança de direção.

Os gráficos de linhas podem mostrar dados contínuos ao longo do tempo, definidos contra uma escala comum, e são por isso ideais para mostrar tendências dos dados a intervalos regulares. Num gráfico de linhas, os dados das categorias são distribuídos regularmente ao longo do eixo horizontal, e todos os dados dos valores são distribuídos regularmente ao longo do eixo vertical [38]. Estes gráficos podem existir sobre várias formas: linhas com marcadores, linhas empilhadas, entre outros.

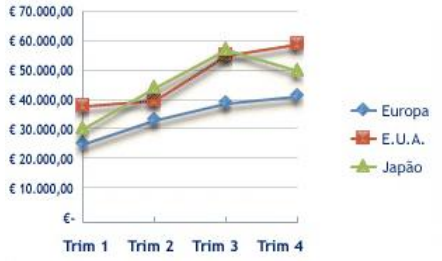

Figura 7. Gráfico de linhas.

A Figura 7 ilustra a evolução do volume de vendas, por trimestres, da Europa, E.U.A e o Japão, usando um gráfico de linhas.

2.2.4.2 Gráficos de barras

Os gráficos de barras [37] são utilizados para apresentar e comparar dados. Existem dois tipos de gráficos de barras: horizontal e vertical. Estes são fáceis de perceber porque consistem num conjunto de barras retangulares que possuem diferentes alturas ou comprimentos de acordo com o seu valor ou frequência e cada barra representa uma variável numérica ou categoria. As semelhanças dos gráficos de linhas, os gráficos de barra representam dados de séries temporais. No entanto, os gráficos de barras permitem visualizar uma mudança na magnitude e não na direção como nos gráficos de linhas.

Os gráficos de barra verticais, também conhecido como gráficos de colunas, são usados para comparação de dados de séries temporais e distribuição de frequências [37].

Figura 8- Gráfico de barras verticais.

A Figura 8 apresenta um gráfico de barras verticais que exibe o valor das vendas, por trimestre, realizadas nos anos de 2004 e 2005 na Ásia Oriental.

Por sua vez os gráficos de barras horizontais, são particularmente úteis quando os rótulos das categorias são longos e os valores mostrados são durações [38].

Figura 9. Gráfico de Barras.

A Figura 9 exibe um gráfico de barras que representa o valor das vendas por região.

Estes gráficos podem existir sob várias formas: barras agrupadas, barras empilhadas, cilindros, cones, pirâmides, entre outros, e são utilizados para comparar as características das series de uma só vez. No entanto, devido a estas variações apresentarem muita informação sobre as séries de dados, estes podem tornar-se confusos.

2.2.4.3 Gráficos circulares

Os gráficos circulares são fáceis de fazer, fáceis de ler e são os mais conhecidos. Estes são usados para representar dados quantitativos. Um gráfico circular é basicamente composto por um circulo que é dividido em segmentos ou categorias que refletem a proporção das variáveis em relação ao todo. Existem diferentes variações destes tipos de gráficos, entre elas: circular de circular e barra circular e gráfico circular destacado [37].

Estes tipos de gráficos não são recomendados quando existem muitos segmentos, já que dificulta a compreensão e leitura do gráfico. Para além disto, se os valores dos segmentos estiverem muito próximos, o gráfico também se torna difícil de ler [37].

Figura 10. Gráfico circular.

Na Figura 10, é possível observar um gráfico circular que mostra a percentagem dos itens vendidos na hora do almoço num estabelecimento comercial.

Para além dos gráficos, existem outro tipo de visualizações que podem ser utilizadas para melhorar compreensão dos dados. Estas visualizações são: mapas de árvores, mapas de calor, grafos, diagramas em arco, entre outros.

2.2.4.4 Mapa de árvore

O mapa de árvore ou *tree map* permite a visualização de estruturas hierárquicas. Esta visualização é muito eficaz para mostrar os atributos das estruturas hierárquicas ou em arvore utilizando o tamanho e cor como atributos do gráfico [39].

Figura 11. Mapa de árvore.

Cada item é representado por um retângulo, Figura 11, e este possui um tamanho diferente, sendo que a área do retângulo é proporcional ao atributo definido pelo utilizador [39].

2.2.4.5 Mapas de calor

Um mapa de calor ou HeatMap [40], Figura 12 ,é uma representação gráfica de pontos com diferentes valores, os pontos vermelhos normalmente são pontos altamente frequentes, ou de maior importância e os pontos azuis ou pretos são locais menos frequentes ou de menor importância.

Figura 12. Mapa de calor.

O intervalo de cores pode variar consoante os dados utilizados e também é possível definir a cor utilizada para o item de maior importância e o item de menor importância.

2.2.4.6 Grafos

Um grafo G = (V, E) [41] é um conjunto V de vértices e um conjunto E de arestas onde cada aresta é um par de vértices (Ex.: (v, w)). Um grafo é representado graficamente usando bolinhas para vértices e retas ou curvas para arestas. A Figura 13 ilustra um exemplo de um grafo:

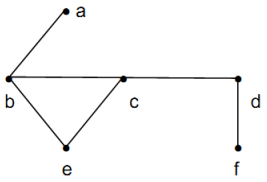

Figura 13. Exemplo de um Grafo.

2.2.4.7 Hyperbolic Tree

Baseia-se no uso da geometria hiperbólica que permite representar uma grande quantidade de elementos sem que estes ultrapassem o espaço limitado pela "borda". Este tipo de visualização, Figura 14, pode ser visto como um mapa conceitual onde os componentes diminuem ou aumentam de tamanho exponencialmente, em função da sua distância ao centro de um círculo de raio unitário [42].

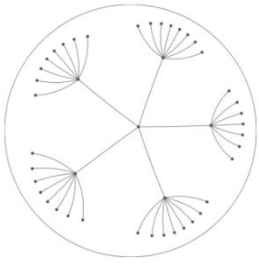

Figura 14. Exemplo de uma Hyperbolic Tree.

Para além destas visualizações, existem muitas outras visualizações, algumas das quais se encontram no anexo I, que permitem uma outra representação dos dados. A Figura 15 ilustra um

resumo dos tipos de visualizações, os tipos de dados que são suportados e qual é o seu propósito/ função [43].

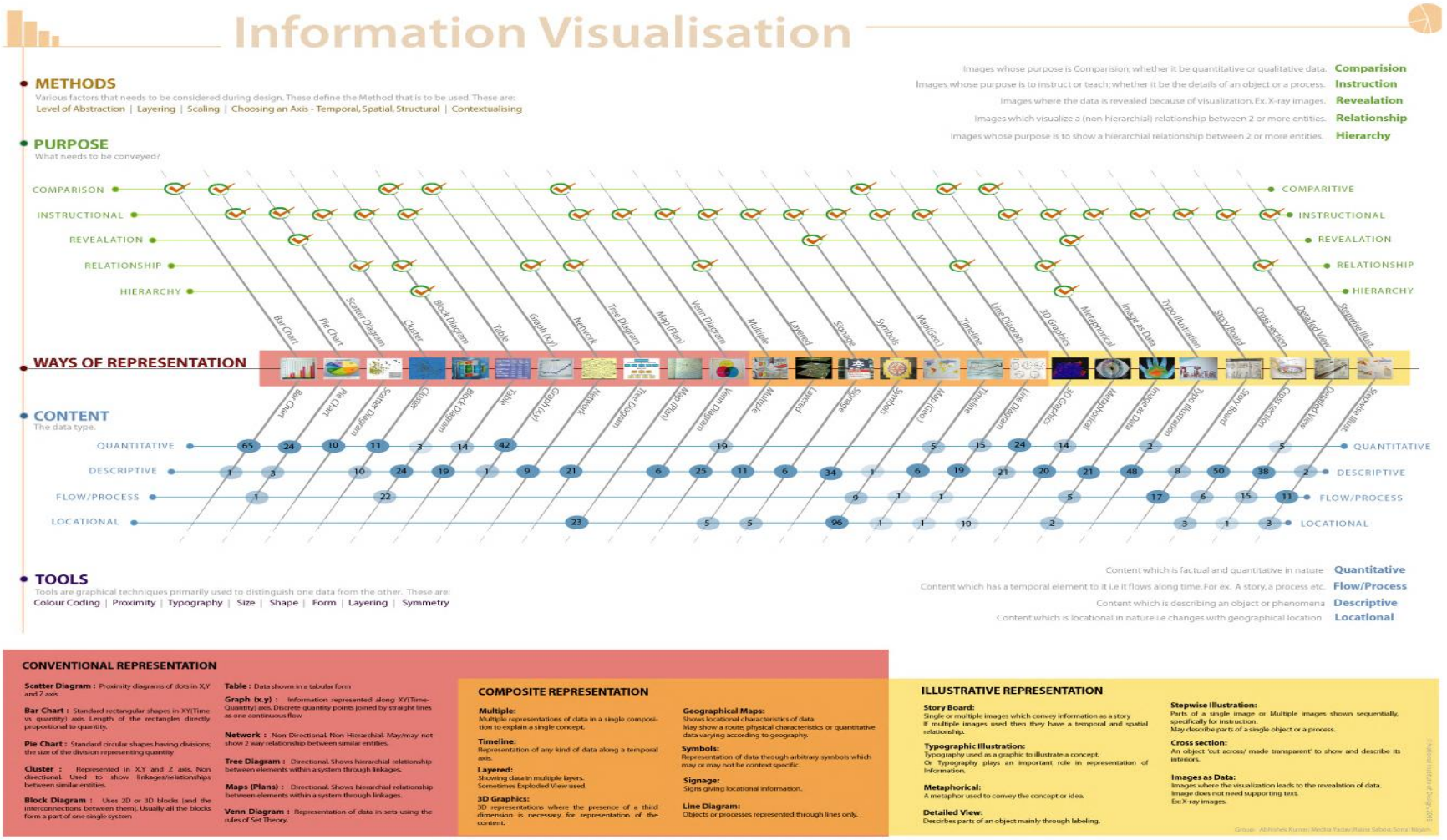

Figura 15. Resumo dos tipos de visualizações.

2.2.5 Padrões de desenho de visualizações

Na secção anterior foi possível constatar que existem diversos tipos de visualizações, em que cada uma possui uma perspetiva diferente sobre os dados. Esta Secção apresenta quais são os padrões de desenho das visualizações.

Fabricantes famosos de móveis, como James Krenov e Maloof Sam, acreditam que um bom desenho deve envolver o processo de fazer e utilizar o móvel. O mesmo se pode dizer sobre as visualizações [44].

Chen [44] faz a distinção entre os padrões de desenho de *software* e os padrões de desenho de visualizações. Por um lado, os padrões de desenho de visualização são utilizados pelos utilizadores dos sistemas de visualização para modelar, desenhar e executar tarefas de visualização. Por outro lado, os padrões de desenho de *software* são utilizados pelos desenvolvedores para desenhar e implementar um sistema, por exemplo, um sistema de visualização. Uma das coisas interessantes na aplicação de padrões de desenho de visualização é que estes têm um grande impacto no desenvolvimento de aplicações de visualização, tornando-se assim padrões de desenho de *software* especiais usados pelos desenvolvedores de aplicações de visualização. Chen [44] identificou nove padrões de desenho de visualização que respeitam os seguintes critérios:

- Existem na área das visualizações dinâmicas e analíticas.
- Resolvem um problema recorrente das tarefas de visualização e apresenta uma solução para este problema.
- Documentação existente, comprovada e prática de desenho comum e experiente.
- Identifica e especifica abstrações que estão acima do nível da prática especifica e experiencia.
- Fornecimento aos utilizadores e desenvolvedores de um vocabulário comum que forneça boas práticas e princípios.
- Fornecimento de um meio de documentação e comunicação de desenho de visualizações.

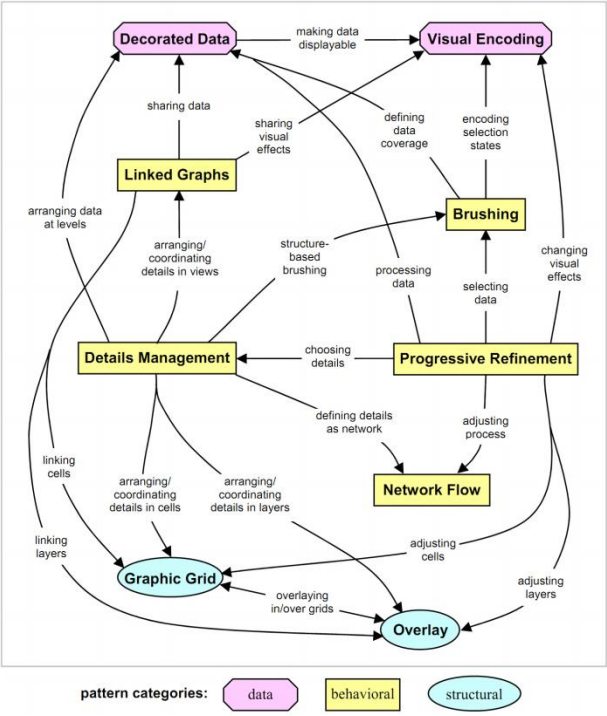

Figura 16. Padrões de desenho identificados.

A Figura 16 mostra os nove padrões de desenho identificados no trabalho realizado por Chen e as relações encontradas entre estes. Os padrões são classificados em três categorias: dados, estruturais e comportamentais. Estas categorias abordam a manipulação dos dados, a aparência dos gráficos e a interatividade [44].

2.2.5.1 Padrões de dados

As visualizações dinâmicas e analíticas tem como objetivo a exibição de dados abstratos, por exemplo, num monitor de um computador. Os padrões de dados focam-se em como organizar e representar visualmente os dados em uma visualização. Existem dois padrões de dados: o padrão *Decorated Data* aborda como organizar os dados brutos, meta-dados e os estados das visualizações, e o padrão *Visual Encoding* que descreve como os dados abstratos são mapeados em elementos visuais (ver anexo II, página 106) [44].

2.2.5.2 Padrões estruturais

Nas visualizações de dados analíticos, existem vários gráficos que são muitas vezes organizados de acordo com regras e princípios para facilitar a exploração de padrões e de tendências. Os padrões estruturais preocupam-se com a forma como os gráficos estão organizados num display. Existem dois padrões estruturais: o padrão *Graphic Grid* que define *layout* para organizar os gráficos em diferentes áreas num display e o padrão *Overlay* que organiza os gráficos num espaço partilhado do display (ver anexo II, página 107). A combinação destes dois padrões

permite resolver muitas das necessidades de um *layout* sofisticado para a visualização de dados analíticos [44].

2.2.5.3 Padrões comportamentais

Uma das características fundamentais na visualização de dados dinâmicos é a capacidade de exploração de dados de forma interativa. Os padrões comportamentais abordam como é que um ou mais gráficos podem ser manipulados interactivamente durante o processo de exploração de dados e como este processo pode ser gerido. Existem cinco padrões comportamentais: o padrão linked graphs, brushing, details management, network flow e progressive refinement (ver anexo II, página 108) [44].

2.2.6 Arquiteturas usadas nas visualizações

Esta secção apresenta as principais arquiteturas que são utilizadas nas visualizações, nomeadamente: Model-View-Controller, Model-View-ViewModel, e Model-View-Presenter.

2.2.6.1 *Model-View-Controller* (MVC)

A arquitetura *Model-View-Controller* (MVC) [45] foi descrita por Trygve Reenskaug em 1979 enquanto trabalhava no *Smalltalk* [46] na Xerox PARC [47]. O objetivo desta abordagem é a separação de uma aplicação em camadas distintas, cada uma com uma função específica. Este padrão divide uma aplicação interativa em três componentes, nomeadamente: o Modelo (*Model*), Vista (*View*) e o Controlo (*Controller*).

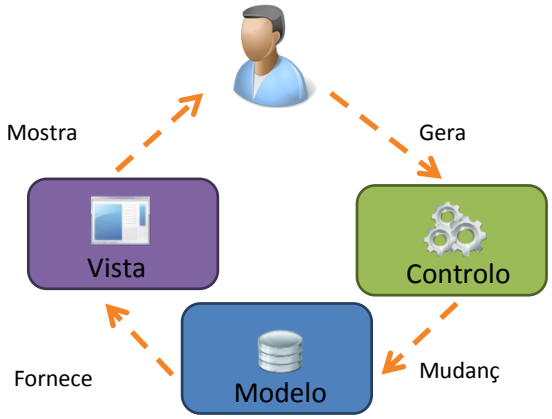

Figura 17. ***Model View Controler.***

A camada de modelo representa os dados da aplicação, a vista representa a interface gráfica dos modelos e finalmente a camada de controlo representa o código que efetua a gestão das ações do utilizador pela interface, aplicando alterações ao modelo e à vista quando necessário [48].

Esta abordagem permite a separação de responsabilidades por cada camada, o que aumenta a organização, modularidade, e reduz o acoplamento dos componentes. Por exemplo, a camada de vista pode ser alterada sem introduzir modificações na camada de modelo [48].

Os benefícios do MVC levaram à sua utilização em aplicações web, onde o modelo representa geralmente conteúdo guardado numa base de dados e regras de negócio associadas, a vista representa as páginas HTML geradas e o controlo é o código que processa os dados retornados pela vista atual. Este processamento pode incluir aplicar alterações ao modelo e geração da próxima vista [48].

2.2.6.2 *Model-View-ViewModel* (MVVM)

Para além do MVC, recentemente surgiu o padrão arquitetural *Model-View-ViewModel* (MVVM), Figura 18. Este padrão, a semelhança do MVC, isola a interface do utilizador da lógica de negócio subjacente, o que permite melhorar a capacidade de teste da aplicação e a evolução das interfaces do utilizador. O MVVM está dividido nas em três partes: o modelo (*Model*), a vista (*View*) e o modelo da vista (*ViewModel*) [49].

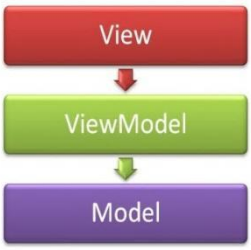

Figura 18. Model-View-ViewModel.

A vista é a interface do utilizador e não possui nenhum conhecimento sobre o modelo. Em alternativa, esta adquire tudo o que precisa através do modelo da vista. Por sua vez, o modelo da vista recupera os dados que se encontram no modelo e os manipula para o formato que é pedido pela vista, sendo este a ponte entre a vista e o modelo. As mudanças nas vistas são inteiramente irrelevantes para o modelo, o qual não possui nenhum conhecimento da vista. As mudanças no modelo são realizadas pelo modelo da vista modelo em resposta aos eventos que tenha sido acionados na vista [49].

2.2.6.3 *Model-View-Presenter* (MVP)

O *Model-View-Presenter* (MVP), Figura 19, basicamente separa a vista da sua lógica de apresentação com o intuito de permitir que a vista possa ser substituída de forma independente. No MVP, a vista torna-se um componente ultra fina cuja finalidade é fornecer uma apresentação ao utilizador. A vista captura e manipula os eventos gerados pelo utilizador, mas encaminha-os

diretamente para o apresentador que sabe como lidar com os eventos. O apresentador comunica com o modelo e coordena os controlos da vista diretamente para apresentar os dados [50].

A relação entre a vista e o apresentador segue mais ou menos o padrão de desenho *decorator*. O apresentador decora a vista com a lógica de apresentação, a fim de que todas as operações na vista sejam da responsabilidade do apresentador [50].

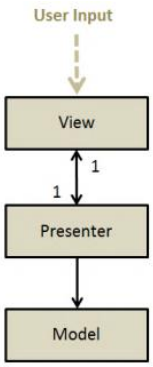

Figura 19. Model-View-Presenter.

Ao separar a vista da lógica da apresentação é simplificado a complexidade e manutenção das aplicações. Por outro lado, o MVP facilita também a realização de testes sem a necessidade do envolvimento de um utilizador e também promove a reutilização permitindo que diferentes vistas possam ser construídas por um apresentador simples [50].

2.2.7 Ferramentas que permitem a visualização de dados

Hoje em dia existem diversas ferramentas que ajudam a visualizar, analisar e validar os dados. Estas ferramentas facilitam o processo de conceção de visualizações através de assistentes ou *wizard* e caixas de diálogo. Muitas vezes, a abrangência ou compatibilidade destas ferramentas é limitada devido a falta de integração com os dados que se encontram em folhas de cálculo ou base de dados [16]. Nesta secção é apresentado um conjunto de ferramentas/aplicações que permitem visualizar os dados.

2.2.7.1 Many Eyes

O *Many Eyes* [51] é uma página pública da internet que é utilizado para criar e partilhar visualizações. Esta página é mais ou menos inspirada em páginas participativas como é o caso do *Flickr* [52] e *Youtube* [53]. As atividades principais que se podem encontrar na página são: a realização de carregamentos de dados, construção de visualização e realização comentários nos dados que foram carregados ou nas visualizações criadas. Este sítio da internet suporta 16 tipos de visualizações, tais como: gráficos, mapas, nuvens de *tags*, mapas de árvore, entre outras e estas visualizações são construídos recorrendo a mini aplicativos java (*applets*) [54].

De forma a poder utilizar as visualizações fornecidas pelo *Many Eyes*, é necessário carregar um ficheiro cujos dados estejam delimitados por vírgulas e depois deve ser selecionada uma visualização. As visualizações são criadas através do mapeamento dos elementos em atributos visuais [16].

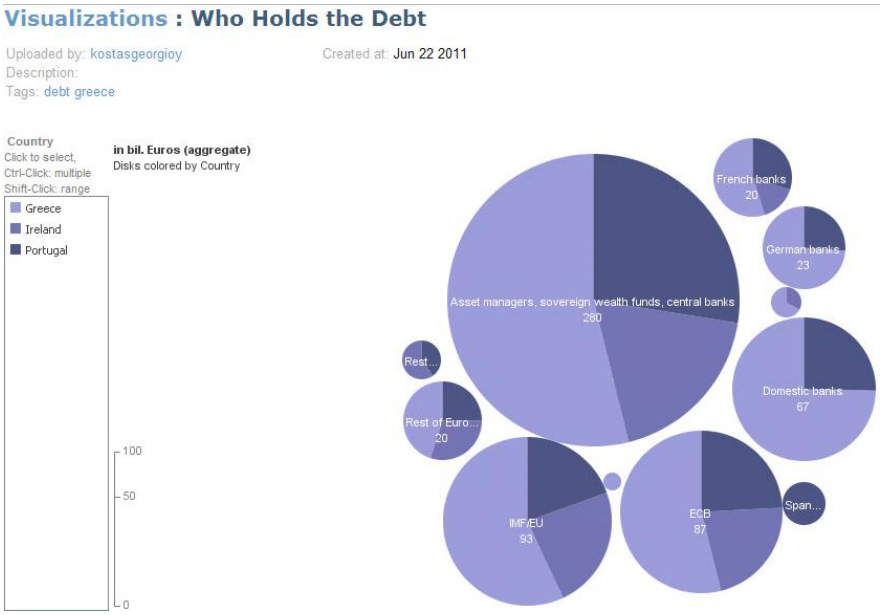

Figura 20. Exemplo de uma visualização do *Many Eye*.

Aa Figura 20 apresenta uma das visualizações realizada por um dos utilizadores do *Many Eyes*. Esta visualização é realizada com a utilização de um gráfico de Bolha (ver anexo I) e o que se pretende com ela é dar a conhecer quais são as entidades que detêm a divida dos três países mais referidos na Europa recentemente.

A utilização do *Many Eyes* para a criar visualizações é extremamente fácil, no entanto, a facilidade de customização é conseguida através das limitações impostas pelas opções disponíveis. O *Many Eyes*, fornece pouco apoio na exploração dos dados, sendo que todas as consultas e processamentos de dados devem ser realizadas por um outro programa. Além disto, uma vez que os dados são carregados, estes não podem ser modificados, limitando assim a evolução da própria visualização [16].

2.2.7.2 *Gephi*

O *Gephi* [55] é uma plataforma para a exploração e visualização interativa de grafos, Figura 21. À semelhança do *Photoshop* [56], mas neste caso orientado para dados, o utilizador interage com a representação, manipula as estruturas, formas e cores de forma a revelar propriedades ocultas. O objetivo é ajudar os analistas de dados a descobrir intuitivamente padrões, isolar singularidades da estrutura ou as falhas durante o fornecimento de dados. Dentro das principais funcionalidades do *Gephi* podem-se mencionar as seguintes [55]:

- Visualização em tempo real: através do motor gráfico de visualização utilizado para acelerar a visualização de dados é possível iterar através das visualizações usando filtros dinâmicos e ferramentas de manipulação de grafos. O *Gephi* possui um motor *OpenGL* para a visualização dos grafos.
- *Layout*: os algoritmos de *layout* permitem dar a forma ao grafo. O *Gephi* permite ao utilizador alterar as configurações de *layout* durante a execução e, portanto, aumentar o feedback do utilizador e a sua experiência com a aplicação.
- Métricas: O *Gephi* oferece métricas para a análise de redes sociais e redes de escala livre, entre elas: *Diameter*, *Average Path Length*, *Clustering Coefficient*, entre outras.
- Agrupamento e hierarquias de grafos: é possível explorar grafos multinível com *Gephi*, facilitando a exploração e edição de grandes grafos hierarquicamente estruturados, por exemplo, comunidades sociais, os caminhos bioquímicos ou grafos de tráfego de rede.
- Filtragem dinâmica: a filtragem permite selecionar nós e / ou arestas com base na estrutura de rede ou de dados usando para tal uma interface interativa para filtrar a rede em tempo real.

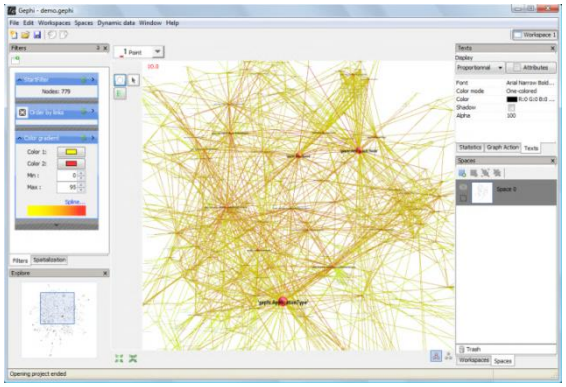

Figura 21. *Gephi.*

2.2.7.3 *Prefuse*

O *Prefuse* [57] é uma *framework* extensível que permite aos desenvolvedores a criação de aplicações de visualização interativa, usando como linguagem de programação, Java. Esta plataforma pode ser utilizada para criar aplicações independentes, componentes visuais incorporados em aplicações maiores e mini aplicações Web. O *Prefuse* pretende simplificar: os processos de representação, a entrega eficiente de dados, o mapeamento de dados em representações visuais nomeadamente: através da posição, tamanho, forma, cor, etc., e permitir também uma interação direta na manipulação dos dados visualizados. Algumas das características do *Prefuse* incluem: tabela, grafos e estruturas em árvores de dados que dão suporte a atributos arbitrários de dados, indexação de dados e consultas de seleção, entre outras características.

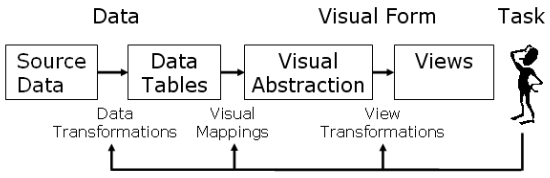

Figura 22. Modelo de referência da visualização.

A Figura 22 apresenta o modelo de referência da visualização no *Prefuse*. Em primeiro lugar, a fonte de dados é mapeada em tabelas. Estas tabelas são aquilo que se encontram pode detrás das visualizações. Em segundo lugar, é construída uma abstração visual a partir das tabelas de dados. Por último, a abstração visual é utilizada para criar visualizações interativas de dados, as quais podem afetar uma mudança em qualquer nível do modelo de referência da visualização.

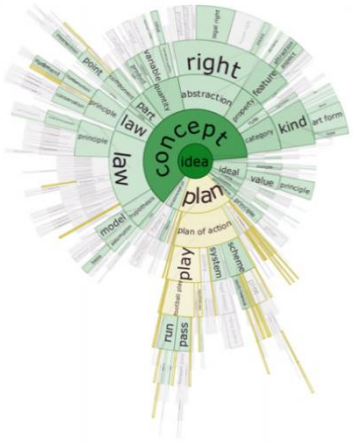

Figura 23. *DocuBurst*.

A Figura 23 exibe uma visualização criada com o auxílio do *Prefuse*, o *DocuBurst*. O *DocuBurst* permite a visualização do conteúdo do documento e tira proveito da estrutura humana criada em bases de dados lexicais [58].

2.3 A modelação na engenharia de *software*

O desenvolvimento de *software* é uma tarefa difícil devido a vários fatores, desde o planeamento, ao levantamento de requisitos, à escolha e implementação de padrões arquiteturais, de desenho e algoritmos, gestão de equipa, entre vários outros [59]. Para além destes fatores, nas grandes empresas de desenvolvimento de *software*, onde os projetos evoluem rapidamente, é difícil para os gestores de projeto manter o controlo das alterações que vão sendo realizadas no software durante o seu ciclo de vida [60]. Uma das formas para manter o controlo das alterações é através de uma boa documentação. Existem várias formas de realizar

documentação no desenvolvimento de software, nomeadamente através dos requisitos, planos do projeto, especificações de desenho, modelos, plano de testes, entre outros.

2.3.1 Modelos

A modelação é uma forma de simplificar o mundo real a fim de poder resolver os problemas que nos rodeiam. A modelação é utilizada em qualquer parte e em qualquer momento, sem que muitas vezes se tenha a noção de que os se está a usar, por exemplo: um calendário é um modelo de um mês, um mapa é um modelo das estradas de uma cidade. Portanto, as pessoas utilizam os modelos para resolver problemas do dia-a-dia, como por exemplo: "Qual o caminho mais curto?", "Quanto tempo falta até o meu aniversário?" [61].

O termo "modelo" derivada do latim *modulus*, que significa medida, regra, padrão, exemplo a ser seguido. A definição formal do modelo pode ser: Qualquer pessoa que utiliza um sistema A que não está nem direta nem indiretamente, interagindo com um sistema B, para obter informações sobre o sistema B, está a usar A como um modelo para B. Esta definição é bastante genérica. É possível descrever o conceito modelo de forma mais precisa, apresentando três critérios para um modelo. Stachowiak [62] descreve que um modelo precisa atender a três critérios essenciais, caso contrário não se trata de um modelo:

- **Mapeamento:** um modelo é baseado num original. O (sistema) original pode ser algo ainda a ser construído ou pode permanecer completamente imaginário.
- **Redução:** nem todas as propriedades do sujeito são mapeados para o modelo, mas o modelo é de certa forma reduzido. No entanto, um modelo deve refletir, pelo menos, algumas das propriedades do assunto.
- **Pragmática:** um modelo precisa de ser útil no lugar de um sujeito com relação a alguma finalidade.

Os modelos são tradicionalmente utilizados na engenharia de *software* para documentar aspetos do desenho e, em alguns casos, como base para a geração de parte ou a totalidade dos sistemas informáticos que descrevem. Portanto, em resumo um modelo é uma abstração de um sistema ou uma parte deste. Dependendo do tipo de modelo, este pode fornecer uma vista simples ou mais detalhada do sistema [63].

Os modelos são geralmente utilizados para:

- **Comunicar:** Os modelos são utilizados como meio de comunicação entre as partes envolvidas. É particularmente importante que todas as partes envolvidas possam

entender a terminologia que está a ser usada de forma a haver um acordo entre as partes e também reduzir as ambiguidades que podem resultar da linguagem natural [64].

- **Visualizar:** Existe a necessidade de apresentar todos os factos relevantes para que as partes envolvidas possam entendê-los. A utilização de diagramas é uma forma muito eficaz de comunicar e entender todos os factos relevantes em oposição à linguagem natural [64].
- **Verificação:** a utilização de modelos permitem que seja possível verificar os factos obtidos em termos de completude, consistência e exatidão [64].

2.3.2 Meta-modelo

Um modelo é uma abstração de um sistema ou parte deste. Um meta-modelo é novamente outra abstração, destacando as propriedades do modelo em si. A relação entre um modelo e o seu meta-modelo é a mesma que existe entre um programa e a gramática da linguagem de programação em que este está escrito [60]. Portanto, um meta-modelo é um modelo que descreve uma linguagem de modelação e como tal descreve os aspetos estruturais e semânticos dessa linguagem [11].

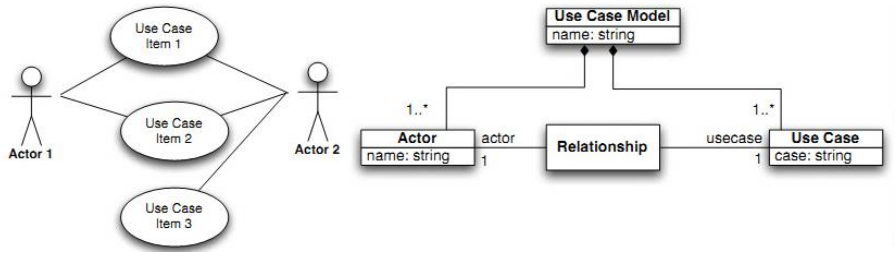

Figura 24. Exemplo de um meta-modelo.

A Figura 24 ilustra o exemplo de um meta-modelo. Do lado esquerdo pode-se observar o modelo de caso de utilização e no lado direito, o respetivo meta-modelo do modelo apresentado. No meta-modelo é possível observar que um modelo de caso de utilização possui um nome e está relacionado com um ator através de uma só relação. Por sua vez, um ator possui também um nome. A partir deste meta-modelo, é possível construir os modelos de caso de utilização que se encontra no lado esquerdo da Figura 24. De salientar que este meta-modelo é apenas uma pequena parte do meta-modelo completo que representa os casos de utilização e só foi utilizado neste contexto a título ilustrativo [65].

O meta-modelo descreve cada elemento do modelo e como estes estão relacionados. Este fornece informações sobre quais o atributos e os tipos que se encontram no modelo. Além disso, também é especificado a cardinalidade dos elementos [65].

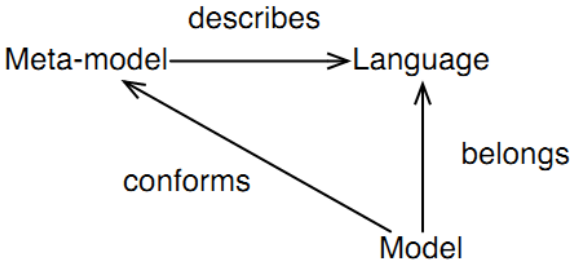

Figura 25. Relações entre os meta-modelos e modelos.

A Figura 25 apresenta as relações entre os meta-modelos e os modelos que é proposta por Jean-Marie Favre [66] e Jean Bézevin [67]

2.3.3 Model Driven

Para a maioria das pessoas o termo "model-driven" evoca a noção de automatização do desenvolvimento através da geração de código a partir de um modelo abstrato [68]. As abordagens *Model-driven* (MD) usadas no desenvolvimento de *software* tem vindo a melhorar a forma como é construído software. Estas abordagens aumentam a produtividade, diminuem o custo em termos de tempo e dinheiro, melhora a reutilização de *software* e permite uma melhor manutenção do software [69]. Existem várias abordagens no MD, entre as quais: o *Model Driven Development* (MDD), o *Model Driven Architecture* (MDA), o *Model Driven Engineerin*g (MDE), entre outras.

2.3.3.1 Model Driven Development (MDD)

O *Object Management Group* (OMG) foi fundado em 1989 e tem como objetivo o desenvolvimento de *standards* na indústria de software, entre os quais: *Unified Modeling Language* (UML), *Meta Object Facility* (MOF), *XML Metadata Interchange* (XMI), entre outros. Todos estes padrões contribuíram para tornar a ideia do *Model Driven Development* (MDD), uma realidade [63].

O MDD [63] é uma abordagem ao desenvolvimento de *software*, proposta pela OMG, que se baseia na utilização de uma única fonte de informação que contém todas as informações sobre o desenvolvimento de software. Os principais objetivos desta abordagem são a simplificação e a padronização das atividades que fazem parte do ciclo de vida do projeto.

Existem cinco aspetos que uma infraestrutura de apoio ao MDD deve definir [70]:

- Os conceitos que permitam a criação de modelos e regras claras que regulam a sua utilização.
- A notação a ser utilizada nos modelos representados.

- Deve ser claro como é que os elementos do modelo representam os elementos do mundo real e os artefactos de *software*.
- Os conceitos que facilitem a extensão dinâmica dos conceitos do modelo, a notação do modelo e os modelos que são criados a partir deste.
- Os conceitos que facilitem a troca dos conceitos e da notação do modelo, e os modelos que são criados a partir dos conceitos para facilitar os mapeamentos dos modelos para outros artefactos.

Quando é utilizado o MDD, é mais fácil comunicar com os diferentes intervenientes durante o ciclo de vida do desenvolvimento de *software*. Também é possível relacionar artefactos e utiliza-los como meio de comunicação entre os intervenientes do projeto. No entanto, a inter-relação entre os vários tipos de modelos e as diferentes linguagens de modelação torna mais difícil a tarefa de compreensão do sistema [60].

2.3.3.2 Model Driven Architecture (MDA)

Aproximadamente no ano de 2001, a OMG adotou uma nova *framework* a qual atribuiu o nome de *Model Driven Architecture* (MDA). Ao contrário dos outros padrões da OMG, o MDA oferece uma maneira de usar modelos, em vez do código fonte. A modelação é uma forma de pensar em questões que fazem parte do projeto antes de começar na codificação, porque esta permite pensar a um nível mais alto de abstração [71]. Este nível de abstração é suposto fazer com que o MDA seja mais fácil de usar e de entender. Também fornece um certo grau de independência da plataforma [63].

Existem quatro princípios que fundamentam a abordagem MDA [63]:

- Os modelos são expressos numa notação bem definida, o que é fundamental para a compreensão do sistema.
- A construção de sistemas pode ser organizada em torno de um conjunto de modelos através da imposição de uma série de transformações entre modelos.
- A sustentação formal para a descrição de modelos em conjunto de meta-modelos, facilita a integração e transformação entre os modelos, e é a base para a automatização por meio de ferramentas.
- A ampla aceitação e adoção desta abordagem baseada em modelos requer que existam normas industriais para proporcionar transparências aos consumidores e uma maior concorrência entre os fornecedores.

Os benefícios da utilização do MDA são os seguintes [63]:

- **Portabilidade:** aumenta a reutilização da aplicação, reduz o custo e a complexidade de desenvolvimento.
- **Interoperabilidade entre plataformas:** a utilização de métodos rigorosos permitem garantir que os padrões baseados em múltiplas tecnologias de implementação tenham funções idênticas.
- **Independência entre plataformas:** permite reduzir o tempo, o custo e a complexidade associados ao desenvolvimento de aplicações para plataformas diferentes.
- **Produtividade:** ao permitir que os vários intervenientes utilizem linguagens e conceitos que são comuns e com os quais se sentem confortáveis, permite uma comunicação continua e uma melhor integração das equipas, o que resulta num aumento da produtividade.

Os princípios básicos do MDA são melhor descritos pelo seu logotipo, Figura 26.

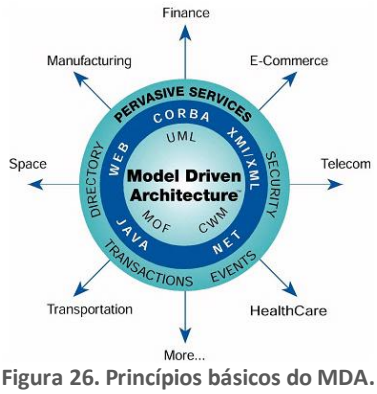

Figura 26. Princípios básicos do MDA.

Todos estes benefícios levam a redução do custo da aplicação, do tempo de desenvolvimento, melhoram a qualidade da aplicação e existe um maior retorno do investimento realizado. No entanto, existe um lado negativo no uso do MDA.

De acordo com os princípios do MDA, existem várias representações dos artefactos que fazem parte do processo de desenvolvimento de *software*. Estes artefactos são representados por diferentes vistas ou níveis de abstração. Quando os artefactos são criados manualmente, a repetição do trabalho e uma gestão de consistência são obrigatórios.

Outro problema associado a modelação é o problema do *round-trip*. Em sistemas complexos, é necessário um conjunto de modelos e artefactos em diferentes níveis de abstração o que aumenta a complexidade das relações entre eles. Assim, quando uma mudança precisa de ser realizada num artefacto que influencia outros artefactos e relacionamentos, em alguns casos, é impossível recorrer a uma automatização da mudança, sendo necessária uma intervenção manual, e isto é especialmente difícil se uma mudança é realizada nos níveis mais baixos dado ao facto de que na maioria dos modelos, os níveis mais baixos são gerados automaticamente.

Em alguns casos, não é possível gerar todo o código automaticamente, sendo necessário que os desenvolvedores editem manualmente o código gerado. Mas isto também traz a questão da otimização do código. O MDA também exige maior capacidade e especialização porque o desenvolvedor precisa de aprender uma ou mais linguagens que são utilizadas na modelação.

Uma vez que os artefactos resultantes de qualquer fase do ciclo de vida podem influenciar os artefactos produzidos em qualquer outra fase, é necessário existir conhecimento sobre as diferentes tecnologias e terminologias utilizadas no modelo. Para além disto, o MDA não oferece um sistema formal de linguagens padrão para a modelação e transformações específicas, apenas fornece sugestões. Isto reduz a interoperabilidade entre as ferramentas, pois diferentes fabricantes têm diferentes implementações e isto leva a que, em alguns casos, um projeto seja dependente das ferramentas que utiliza [71].

2.3.3.3 Model Driven Engineering (MDE)

O *Model Driven Engineering* (MDE) tem como objetivo aumentar o nível de abstração na especificação de um programa e aumentar a automatização no desenvolvimento do programa [72]. A ideia promovida pela MDE é a utilização de modelos em diferentes níveis de abstração para o desenvolvimento de sistemas, aumentando o nível de abstração na especificação do programa. Um aumento da automatização no desenvolvimento de programas é alcançado através de transformações do modelo executável. Os modelos de nível superior são transformados em modelos de nível mais baixo até que o modelo possa ser feito utilizando um programa de geração de código ou um interpretador de modelos [72].

O MDE é frequentemente confundido com MDA. Mas o MDA pode ser visto como a visão da OMG em MDE. O MDA concentra-se na variabilidade técnica do *software*, ou seja, como especificar o software de forma independente da plataforma.

O MDE pode ser comparado com a metade inferior da Figura 27. Basicamente esta figura demonstra que a partir de um conjunto de ideias que estão descritas no modelo de uma casa, e sendo este um modelo completo, se o modelo for colocado no gerador, o resultado final pode ser gerado automaticamente.

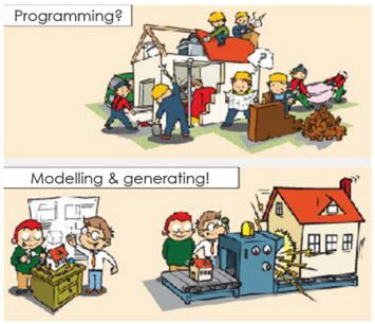

Figura 27. Processo do MDE.

Quando é utilizado uma abordagem MDE no desenvolvimento de *software* o que acontece é que o esforço da programação é transferido para a modelação. Com base nos requisitos funcionais, um modelo é desenvolvido. Este modelo pode ser convertido automaticamente em *software* funcional com uma ferramenta de geração de código a partir do modelo, ou pela interpretação do modelo com um ambiente de execução (também conhecido como máquina virtual). No anexo III o leitor poderá encontrar um glossário com os conceitos associados ao MDE.

2.3.4 Linguagens de modelação

Como foi dito anteriormente, os seres humanos possuem uma necessidade natural de comunicação, sendo que temos vindo a criar diferentes formas de comunicar e transmitir a informação. O ser humano também precisa de comunicar os modelos para que possa transmitir a informação que se encontra neles e ter uma melhor compreensão. A fim de comunicar os modelos são utilizadas linguagens de modelação.

Uma linguagem de modelação [73] é uma linguagem que é utilizada para expressar conhecimento ou informação através de uma estrutura que é definida por um conjunto consistente de regras. Existem dois tipos de linguagens de modelação, as gráficas e as textuais. Por um lado, as linguagens de modelação gráficas são representadas tipicamente por diagramas, utilizando símbolos que representam conceitos. Por outro lado, as linguagens de modelação textual são representadas utilizando palavras-chaves.

Existem várias linguagens de modelação, entre as quais: *Unified Modeling Language* (UML), *Business Process Model and Notation* (BPMN), *Human Activity Modeling* (HAM) e Domain Specific Languages (DSL's) [74].

2.3.4.1 Unified Modeling Language (UML)

O UML é utilizado para especificar, visualizar, construir e documentar os artefactos de um sistema de *software*. Desde os artefactos conceituais tais como: os processos de negócio, as componentes dos sistemas e as atividades, passando pelos artefactos concretos tais como: os esquemas das bases de dados, gráficos de estados e modelos de atividade, o UML combina as práticas que provem dos conceitos de modelação de dados utilizados em todos os processos, em todo o ciclo de vida do desenvolvimento de software, através da de diferentes tecnologias de implementação. Esta linguagem de modelação pretende ser uma linguagem padrão que permita modelar todos os sistemas e seja amplamente utilizada pela comunidade de desenvolvimento de *software* [65].

Um sistema pode ser modelado através da utilização de diferentes abstrações. Cada abstração exigirá um diagrama do modelo que o descreva. Um diagrama é uma representação gráfica parcial do modelo de um sistema. Estes representam três vistas diferentes de um modelo de um sistema, nomeadamente [65]:

- **Vista estática (estrutural):** é composta pelos diagramas de classes e os diagramas de estrutura composta. Estes enfatizam a estrutura estática do sistema, utilizando objetos, atributos, operações e relacionamento.
- **Vista dinâmica (comportamental):** é composta pelos diagramas de atividades, diagramas de sequência e diagramas de estado. Estes diagramas demostram a colaboração entre os objetos e as mudanças internas dos objetos.
- **Vista interativa (interativo):** é composta pelos diagramas de comunicação, diagramas da visão global de interação e os diagramas de temporização. Estes diagramas modelam o fluxo de controlo e de dados entre os elementos do sistema.

Apesar de ser considerado como *Unified Modeling Language*, o UML é frequentemente criticado pelas seguintes razões [65], [75]:

- **Dimensão da linguagem:** o UML contém muitos modelos que são redundantes ou não são utilizados.
- **Vistas Fracas:** os diagramas gráficos são muitos semelhantes, sendo que, o mesmo conector de relacionamento têm diferentes significados de modelo para modelo.
- **Mau uso da simbologia:** as ferramentas CASE normalmente fornecem símbolos e não realizam qualquer tipo de validação sobre a sua utilização correta.
- **Falta de suporte na sincronização de código:** se o código for gerado a partir do modelo UML, este continua a evoluir deixando o modelo ultrapassado.
- **Inconsistência:** Embora existam defensores que argumentam que os símbolos possuem a mesma aparência, a fim de fornecer modelos uniformes, a utilização dos mesmos símbolos com significados diferentes levam a interpretações erradas.
- **Perfis e estereótipos como meio de extensibilidade:** com o objetivo de ser uma linguagem de modelação de uso geral, esta tenta ser compatível com todas as linguagens necessárias, e mesmo assim, ainda existe falta de apoio consistente para estas linguagens.
- **Suporte para o formato de intercâmbio:** a definição de um modelo UML 2.0 numa ferramenta e, em seguida a sua importação para outra ferramenta, tipicamente leva à perda de informações. A especificação do formato de intercâmbio de diagramas carece de detalhes suficientes a fim de facilitar o mesmo entre as ferramentas de modelação.

2.3.4.2 Business Process Model and Notation (BPMN)

O BPMN foi lançado inicialmente em maio de 2004 pela associação internacional BPMI (*Business Process Management Initiative*) e é uma notação cujo objetivo é o fornecimentos de instrumentos que permitam mapear todos os processos de negócios da organização. O BPMN destina-se a ser compreendido pelos analistas de negócios, técnicos, utilizadores e eventualmente sistemas. Em 2005 este padrão passou a fazer parte dos padrões/especificações da OMG [76].

Em termos de características, é possível mencionar as seguintes:

- **Simples:** O BPMN pode começar a ser utilizado com os elementos básicos dos fluxogramas e evoluir para elementos mais complexos [76].
- **Flexível:** O BPMN deve ser capaz de fazer o mapeamento dos processos internos e externos da organização [76].
- **Não-técnico:** Os analistas de BPMN não precisam ser necessariamente profissionais técnicos [76].
- **Expansível:** A organização deve poder expandir o modelo de acordo com regras e interesses próprios, criando novos instrumentos de modelação sem prejudicar a especificação já existente [76].

2.3.4.3 Human Activity Modeling (HAM)

O *Human Activity Modeling* [77] é uma abordagem sistemática para organizar e representar os aspetos contextuais do uso de ferramentas que estejam bem fundamentadas em uma *framework* teórica aceite e incorporado dentro de um método de desenho comprovado. A teoria da atividade fornece um vocabulário e uma estrutura conceitual para a compreensão do uso de ferramentas e outros artefactos por parte dos seres humanos.

2.3.4.4 Domain Specific Languages (DSL's)

As DSL's são utilizadas para aumentar a flexibilidade, qualidade e entrega dos sistemas de *software*, aproveitando as propriedades específicas do domínio de uma aplicação particular. Como é possível ter abordagens genéricas e específicas para resolver um problema, as DSL's procuram alcançar uma solução que seja ótima para um problema específico [74], [65].

A utilização das DSL's trás vantagens e desvantagens. As principais vantagens na utilização das DSL's são as seguintes [65]:

- Permitem que as soluções sejam expressas numa linguagem e nível de abstração do domínio do problema, permitindo aos especialistas do domínio poderem ter uma melhor compreensão, validação e modificação de modelos.
- Aumenta a produtividade, confiabilidade, manutenção e portabilidade.
- Incorpora conhecimento do domínio, a fim de que seja possível a conservação e reutilização deste conhecimento.
- Permite a validação e otimização ao nível do domínio.
- Melhora a capacidade de teste se continuar no mesmo domínio e num conjunto de problemas.

As desvantagens na utilização das DSL's são as seguintes [65]:

- Os custos de conceção, implementação e manutenção de um DSL.
- Dificuldade na aprendizagem de um DSL para um novo membro da equipa de desenvolvimento.
- Disponibilidade limitada do DSL.
- Dificuldade no equilíbrio entre as construções do domínio específico e o uso geral da linguagem.

2.4 A visualização no contexto do *Model-driven*

Nesta secção pretende-se dar a conhecer o estado da arte da visualização no contexto do *model-driven*.

Entender grandes sistemas de *software* é uma tarefa difícil. Nos últimos 15 anos, muitas ferramentas de visualização de *software* tem sido propostas para ajudar aos desenvolvedores de software a compreender os sistemas de grande porte. Ferramentas como o *Rigi* [78] e *SHriMP* [79] têm sido usadas com sucesso para navegar nos sistemas de *software* que possuem mais de um milhão de LOC (*lines-of-code*), em parte devido à sua capacidade de gerar vistas interativas. No entanto, estas ferramentas possuem falta de adaptabilidade e flexibilidade quando são aplicadas em ambientes industriais. Para além disso, muitas vezes as ferramentas não são capazes de sincronizar-se com múltiplas fontes de informação que existem nos sistemas de *software*, tais como: base de dados, informações sobre requisitos, repositórios de documentação, entre outros [80].

Hoje em dia, os engenheiros de *software* estão começando a perceber o potencial da visualização de informação. Este potencial cada vez mais está presente em muitos sistemas de *software*. Por sua vez, o número crescente de sistemas de *software* construídos com recurso a modelos e técnicas de modelação faz com que exista a necessidade de integração das técnicas de

visualização com o MDE. Esta desconexão entre a visualização da informação e o resto do sistema requer que os desenvolvedores tenham que gerir múltiplos paradigmas de desenvolvimento o que leva a uma falta de continuidade do sistema. Assim, muitos dos benefícios do MDE não podem ser estendidos para as visualizações sem um modelo formal [16].

O *Model Driven Visualization* (MDV) [16] é uma abordagem para a conceção e geração de visualizações usando o meta-modelo e as transformações de modelos. Esta abordagem utiliza técnicas estabelecidas pelo MDE, o que permite suportar o rápido desenvolvimento de ferramentas de visualização de informação.

O objetivo final do MDV é apoiar a criação de vistas de uma forma compatível com os objetivos do MDE. O MDV contribui em diversas áreas tais como: a engenharia baseada em modelos, visualização de informação e engenharia de *software* [16].

Entre os benefícios oferecidos pelo MDV, é possível mencionar os seguintes [16]:

- Melhor documentação no ciclo de desenvolvimento de *software*.
- Geração automática de adaptadores de modelos.
- Formato conciso para a especificação de vistas simples sendo que as vistas mais complexas recorrem a uma linguagem imperativa para as interações.

Todavia, existem limitações no MDV, entre elas [16]:

- Todo tem que ser modelado, ou seja, um modelo formal deve existir para todas as vistas que sejam criadas e é preciso ter um modelo para os dados.
- As linguagens de transformação declarativa são bastantes diferentes das tradicionais linguagens de programação. A projeção de soluções através de uma série de transformações é algo que pouco provavelmente todos os engenheiros de software estão confortáveis a fazê-lo.

2.5 *Object Constraint Language* (OCL)

Existe um conjunto de abordagens que utilizam os modelos como ponto central do desenvolvimento e existe uma abordagem que utiliza as visualizações no processo de desenvolvimento conjuntamente com os modelos, o MDV. No entanto, como foi referido anteriormente, um dos passos na criação das visualizações é a recolha dos dados. Esta secção descrever a forma como é recolhida a informação contida nos modelos a fim de alimentar as visualizações.

O *Object Constraint Language* (OCL) foi desenvolvido por Jos Warmer na IBM e foi inicialmente utilizada como linguagem de modelação de negócio. Em 1997 a IBM em parceria com *ObjecTime* apresentaram uma proposta a OMG onde se inseria a linguagem OCL. Algumas das partes desta proposta foram posteriormente incluídas na versão 1.1 do UML [11].

O OCL surge como resposta a um problema muito particular, o facto de a linguagem UML não ter os artefactos necessários para que seja possível modelar os aspetos semânticos de um problema [11].

O OCL pode ser definido [81] como sendo linguagem de expressões que permitem especificar restrições nos modelos orientados a objetos ou outros artefactos da linguagem UML. O OCL é uma linguagem precisa, textual e formal, sendo que uma das suas principais características é que o seu uso não exige grandes conhecimentos matemáticos a fim de obter exatidão na sua manipulação [81]. O OCL suporta a declaração de invariantes, pré/pós condições que permitem ao desenvolvedor especificar restrições e detalhar o comportamento dos modelos sem necessidade de entrar em detalhes de implementação [82].

Outra das características do OCL é ser uma linguagem declarativa indicando o "quê" e não o "como", é fortemente tipificada, sendo que cada expressão tem um tipo associado que pode ser primitivo (*Boolean*, *Integer*, *Real*, ou *String*), coleções (*Set*, *OrderedSet*, *Bag*, ou *Sequence*) ou outro tipo definido no seu contexto e não permite a comparação direta entre valores de diferentes tipos. Por outro lado, uma expressão OCL é atómica, ou seja, esta não pode ser subdividida no momento da avaliação. Quando é avaliada uma expressão OCL, o modelo permanece inalterado, ou seja, a avalização das expressões não tem qualquer efeito sobre o modelos e apenas devolvem um valor. No entanto, existe um tipo expressão OCL que permite que seja possível especificar uma mudança no estado do sistema, as pós-condições [11].

No contexto do UML, o OCL possui duas funções. Em primeiro lugar, o OCL é utilizado para formalizar a semântica da própria linguagem UML. Em segundo lugar, o OCL permite expressar, de forma precisa, as restrições sobre a estrutura dos modelos definidos através do UML [11].

Ainda no contexto do UML, o OCL possui um vasto conjunto de aplicações, nomeadamente [11]:

- Como linguagem para consultas.
- Especificação de invariante de classes e tipos no modelo de classes.
- Especificação de tipo de invariantes para *Stereotypes*.
- Descrição de pré e pós-condições em operações e métodos.

- Descrição de guardas.
- Especificação de mensagens e ações.
- Especificação de restrições nas operações.
- Especificação de regras de derivação para atributos.

O OCL é cada vez mais usado nos sistemas como uma forma de navegação entre os modelos. O recente aumento das DSL's tem contribuído para este processo [83]. Por outro lado, também podem-se usar as expressões OCL como um meio para realizar consultas nos modelos. Estas consultas podem ser úteis para extrair informação dos modelos a fim de pode validar os mesmos nas várias etapas do ciclo de desenvolvimento. O valor de retorno das consultas não é um valor booleano mas sim valores de um tipo específico do OCL [81]. No anexo IV, o leitor poderá encontrar uma descrição mais detalhada sobre o OCL como exemplos ilustrativos.

Contudo, e embora o OCL tenha sido desenhado para evitar a complexidades das linguagens formais e a ambiguidade das linguagens naturais, determinadas restrições podem ser muito complexas, sendo que seria preferível descrevê-las através de linguagem natural. De ressaltar que o OCL não é perfeito e é importante simplificar ao máximo as restrições a modelar e as consultas a serem executadas [11].

2.5.1 *Imperative* OCL

O QVT é uma linguagem imperativa que suporta a criação de transformações de modelos [84]. Esta linguagem define duas abordagens para expressar as transformações dos modelos: uma abordagem declarativa e uma abordagem operacional. A abordagem declarativa é a linguagem relacional onde as transformações entre os modelos são especificadas como um conjunto de relações que devem ser mantidas para que a transformação seja bem-sucedida. Por outro lado, a abordagem operacional permite que seja possível definir transformações utilizando uma abordagem completamente imperativa. O QVT introduz o *Imperative OCL* [85].

O *Imperative OCL* é uma extensão do OCL com todas os construtores de programação que são necessários para escrever transformações complexas de uma forma confortável e também estende a hierarquia de tipos do OCL, introduzindo, por exemplo, os dicionários [84]. Basicamente o Imperative OCL é utilizado no QVT para especificar transformações operacionalmente.

2.6 Conclusão

Este estado da arte procurou estudar, de forma sucinta, o estado atual das visualizações, o estado atual do processo de desenvolvimento de *software* com recurso a utilização de modelos e como é que as visualizações podem se tornar mais uma ferramenta valiosa, juntamente como os modelos, no ciclo de desenvolvimento de software.

O conceito de visualização está presente em muitos dos artefactos, ferramentas de desenvolvimento e aplicações de *software* devido a sua importância. É mais fácil observar um conjunto de visualizações que representam uma amostra ou um conjuntos de dados, do que analisar diretamente os dados e retirar alguma conclusão destes. Por outro lado, foi possível observar que diferentes visualizações possibilitam ter diferentes perspetivas sobre os mesmos dados e, consequentemente descobrir novos padrões e melhorar a compreensão sobre os dados.

Para além das visualizações, foi possível observar o trabalho que está a ser realizado na área do desenvolvimento de *software* dirigido por modelos. Existem diversas abordagens que utilizam os modelos como ponto central do ciclo de desenvolvimentos e cada uma destas abordagens apresenta vantagens e desvantagens específicas à sua utilização. No entanto, todas estas abordagens possuem algumas desvantagens comuns associadas ao uso de modelos, nomeadamente, os modelos são vistas parciais do sistema que está a ser desenvolvido, o que não permite ter uma vista geral de todo o sistema e dificulta a sua compreensão.

Para além disto, existe a necessidade de integrar as técnicas de visualização com os modelos, com o intuito de melhorar a compreensão dos sistemas e do ciclo de desenvolvimento. Uma das abordagens de desenvolvimento que permitiu a integração destes dois aspetos mencionados anteriormente é o *Model Driven Visualization* (MDV), mas esta abordagem possui também desvantagens, nomeadamente, a necessidade de existir um modelo formal para cada uma das vistas que sejam criadas.

Outro aspeto observado é o facto de ser preciso também recolher/extrair a informação contida nos modelos. Uma das abordagens para a extração de informação dos modelos é através da utilização do OCL. O OCL é uma linguagem de expressões que, para além de permitir definir restrições nos modelos, também permite realizar consultas, sendo que o resultado dessas consultas são valores de um tipo específico do OCL.

A utilização das visualizações em conjunto com a linguagem de consulta OCL fornece a oportunidade melhorar a compreensão dos sistemas de *software* que estão a ser desenvolvidos. Esta oportunidade bem a satisfazer a necessidade de existir uma ferramenta que permita obter,

através de consultas, informações específicas que se encontram nos modelos e visualizar os resultados das consultas melhorando assim a compreensão e validação dos modelos.

3 Proposta

3.1 Introdução

Neste capítulo é feita a descrição de uma proposta para uma nova ferramenta que permita visualizar as informações contidas nos modelos, a fim de melhorar a compreensão e validação dos modelos. A construção desta ferramenta toma em consideração o trabalho já realizado nesta área e que foi mencionado no estado da arte. Esta ferramenta requer um grande esforço de desenvolvimento devido a complexidade associada a criação dinâmica de visualizações e os algoritmos envolvidos.

Este capítulo está estruturado da seguinte forma: em primeiro lugar é justificado a necessidade desta ferramenta para visualizar os dados contidos nos modelos. Em segundo lugar, dá a conhecer a abordagem que foi utilizada na construção da ferramenta. Por último, descrevem-se as características da ferramenta a ser desenvolvida.

3.2 Fundamentação

Como foi descrito no estado da arte, existe a necessidade de combinar técnicas de visualização com os modelos que são utilizados no desenvolvimento de *software*, com o intuito de melhorar a compreensão dos sistemas que estão a ser desenvolvidos. Esta necessidade tornou-se evidente no trabalho desenvolvido no projeto de modelação de processos de uma secção do governo regional [6], através das notações gráficas *Human Activity Modeling* (HAM) e *Business Process Modeling Notation* (BPMN).

Por outro lado, também existe a necessidade de extração de dados que se encontram nos modelos. Estes dados são importantes na elaboração das visualizações. No entanto, também estes dados contribuem para a tomada de decisões sobre um determinado aspeto do modelo. Por exemplo: supondo que se possui o modelo de uma empresa descrita através da notação gráfica Business HAM [7]. Pretende-se querer saber qual o número de papéis de um determinado ator ou o número de atividades que um ator realiza. A partir dos dados obtidos, por exemplo das consultas anteriores realizadas no modelo, é possível saber se um determinado ator possui uma sobrecarga de trabalho ou fundamentar o porque é que é preciso um novo empregado na empresa.

Com a combinação das visualizações, os dados obtidos nas consultas anteriores ganham uma nova forma, na medida em que, é possível inferir sobre aquilo que está a ser visualizado e tomar melhores decisões de uma forma rápida e simples.

3.3 Abordagem

No desenvolvimento da ferramenta não foi seguida a abordagem que foi utilizada no MDV devido as suas desvantagens e a falta de extração das informações contidas nos modelos.

Em primeiro lugar, pretende-se que a nossa ferramenta permita realizar qualquer tipo de visualização sem a necessidade de definir, para cada visualização, o seu modelo formal. Isto permite maior flexibilidade ao nível das visualizações da ferramenta, possibilitando que os seus utilizadores não precisem de saber como especificar o modelo formal das visualizações existentes.

Em segundo lugar, pretende-se que a ferramenta seja extensível e flexível, isto é, com pouco esforço consegue-se adicionar ou remover visualizações na ferramenta. Para tal, irá ser utilizada uma abordagem baseada em *plugins*, em que cada plugin representa uma determinada componente. Estas componentes podem ser utilizadas várias vezes na aplicação o que possibilita ter diferentes tipos de visualizações no mesmo ambiente de construção.

Por último, irá ser utilizado, como linguagem de consulta, a linguagem OCL. Esta linguagem tem demonstrado ser muito útil quando se pretende navegar através dos modelos, definir restrições e obter informação que se encontra nos modelos através de consultas.

3.4 Características

A fim de facilitar a compreensão do *software*, um software de visualização deve fornecer um conjunto de características de forma a satisfazer um conjunto de requisitos:

- Fornecer uma linguagem de consulta sobre múltiplos modelos.
- Ser independente do formato e estrutura dos modelos.
- Permitir adicionar facilmente novos modelos.
- Suportar diferentes formatos de modelos.
- Fornecer todos os recursos a fim de realizar consultas.
- Permitir a realização de consultas simples.
- Possibilitar adicionar novas visualizações.
- Consentir a combinação das visualizações existentes.
- Ser intuitiva e fácil de usar.

Para além destas características, a ferramenta a ser desenvolvida, terá como linguagem de programação C# [9] e irá utilizar o sistema gráfico *Windows Presentation Foundation* (WPF)

[10], com o intuito de ser integrada na *suite* de ferramentas já desenvolvidas no âmbito da ferramenta Meta Sketch [86].

3.5 Conclusão

A partir do capítulo de estado de arte, foi possível caracterizar qual é o cenário atual do desenvolvimento de *software* com recurso a utilização de modelos. Também foi possível perceber que existe uma necessidade crescente de possuir uma ferramenta que permita a visualização de informações contidas nos modelos, a fim de melhorar a compreensão e validação sobre os mesmos, já que cada vez mais os modelos estão assumindo grandes dimensões e se estão tornando cada vez mais complexos.

A proposta aqui apresentada, em primeiro lugar fundamentou a sua razão a partir da necessidade descrita anteriormente. Em segundo lugar, foi delineada uma abordagem a fim de ser adotada no desenvolvimento da ferramenta, nomeadamente, a construção de uma aplicação que permita realizar consultas e visualizar os resultados dessas consultas através de diferentes visualizações. Finalmente foram descritas as características que esta ferramenta deverá possuir, a fim de satisfazer as necessidades dos desenvolvedores.

É com base nesta proposta que nos próximos capítulos irá ser dado a conhecer a especificação e implementação desta ferramenta.

4 Especificação

4.1 Introdução

A partir da proposta do capítulo anterior, foram desenvolvidas abordagens ao desenvolvimento de uma aplicação de visualização que espera satisfazer a necessidade de combinar as visualizações com o desenvolvimento dirigido por modelos e os requisitos que foram identificados neste trabalho.

Neste capítulo, descreve-se os detalhes estruturais e comportamentais da aplicação a desenvolver recorrendo a alguns diagramas *standards* que fazem parte do UML.

Este capítulo está organizado da seguinte forma: em primeiro lugar é apresentado o diagrama de casos de utilização da aplicação e o diagrama de atividades. Em seguida será indicada a arquitetura geral da aplicação. Por último é apresentado o *wireframe* que servirá de base para a construção da interface gráfica da aplicação.

4.2 Diagrama de casos de utilização

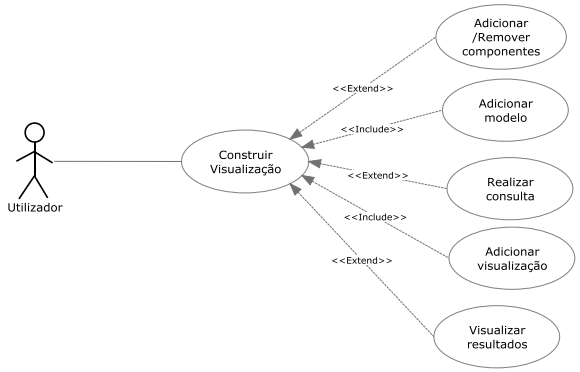

Figura 28. Diagrama de casos de utilização.

Como é possível observar no diagrama de casos de utilização, Figura 28, existe um ator genérico cujo nome é "Utilizador" e existem seis casos de utilização, nomeadamente:

- "Adicionar/ remover componentes": uma componente é uma funcionalidade, como por exemplo: uma nova visualização, um novo método de consulta, que o utilizador adiciona ou remova da aplicação.
- "Construir visualização": consiste na combinação das componentes adicionadas e que se encontram disponíveis na ferramenta com o intuito de obter uma visualização.
- "Adicionar modelos": consiste na adição do modelo a fim de ser possível ter os dados necessários para realizar consultas e obter visualizações.

- “Realizar consulta”: consiste na realização de uma consulta sobre o modelo a fim de obter os dados necessários para uma determinada visualização.
- “Realizar consulta”: consiste na realização de uma consulta sobre o modelo a fim de obter os dados necessários para uma determinada visualização.
- “Adicionar visualização”: consiste na adição de uma determinada visualização.
- “Visualizar resultados”: a partir dos dados obtidos através de uma consulta, é utilizado uma determinada componente para poder visualizar os dados.

4.3 Diagramas de atividades

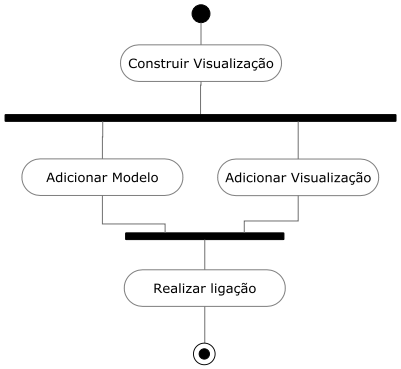

Figura 29. Diagrama de atividade do caso de utilização "Construir Visualização".

A Figura 29 ilustra o diagrama de atividade do caso de utilização “construir visualização”. Neste diagrama é possível constatar que para construir uma visualização é necessário adicionar o modelo, que contém os dados que irão ser utilizados para alimentar a visualização e adicionar a visualização que se pretende. No fim é necessário realizar a ligação entre o modelo e a visualização para que seja possível concluir a visualização. De salientar, que é possível adicionar uma componente de consulta a fim de obter dados específicos do modelo.

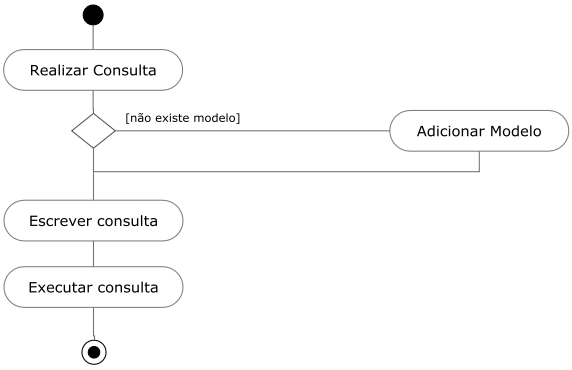

Figura 30. Diagrama de atividade do caso de utilização "Realizar Consulta".

Por outro lado, a Figura 30 apresenta o diagrama de atividade do caso de utilização “Realizar Consulta”. Neste diagrama é possível observar que para realizar uma consulta, é necessário adicionar o modelo caso ainda não o se esteja adicionado, escrever e executar a consulta que se

pretende realizar. Os restantes diagramas de atividades não foram incluídos na dissertação por se considerar serem simples.

4.4 Arquitetura Geral

A fim de garantir a extensibilidade da aplicação foi adotado um conjunto de estilos arquiteturais que possibilitam, de forma fácil e rápida, adicionar novas funcionalidades e facilitar as alterações na aplicação.

Em primeiro lugar, irá ser utilizado o estilo arquitetural *plugins*. Este estilo permite adicionar facilmente novas funcionalidades sem recorrer a adição manual de código na aplicação, permitindo assim aos desenvolvedores a capacidade de adicionar funcionalidades a ferramenta consoante as suas necessidades.

Em segundo lugar, irá ser utilizado o estilo arquitetural *pipes and filters*. O motivo para a sua utilização é dar a possibilidade de ter diferentes componentes interligados. Cada componente possui um conjunto de funcionalidades. Estes irão ser ligados através de portos de entrada e de saída com o intuito dos componentes terem acesso à informação que precisam. Os componentes podem ou não ser transparentes, isto é, podem não alterar a informação que possui sendo que, o que se encontra no porto de entrada está também no porto de saída sem nenhuma alteração.

Por último, irá ser utilizado o estilo arquitetural MVVM. Como foi mencionado no estado de arte, o MVVM é um estilo arquitetural que isola a interface do utilizador da lógica de negócio subjacente, o que permite melhorar a capacidade de teste da aplicação e a evolução das interfaces do utilizador.

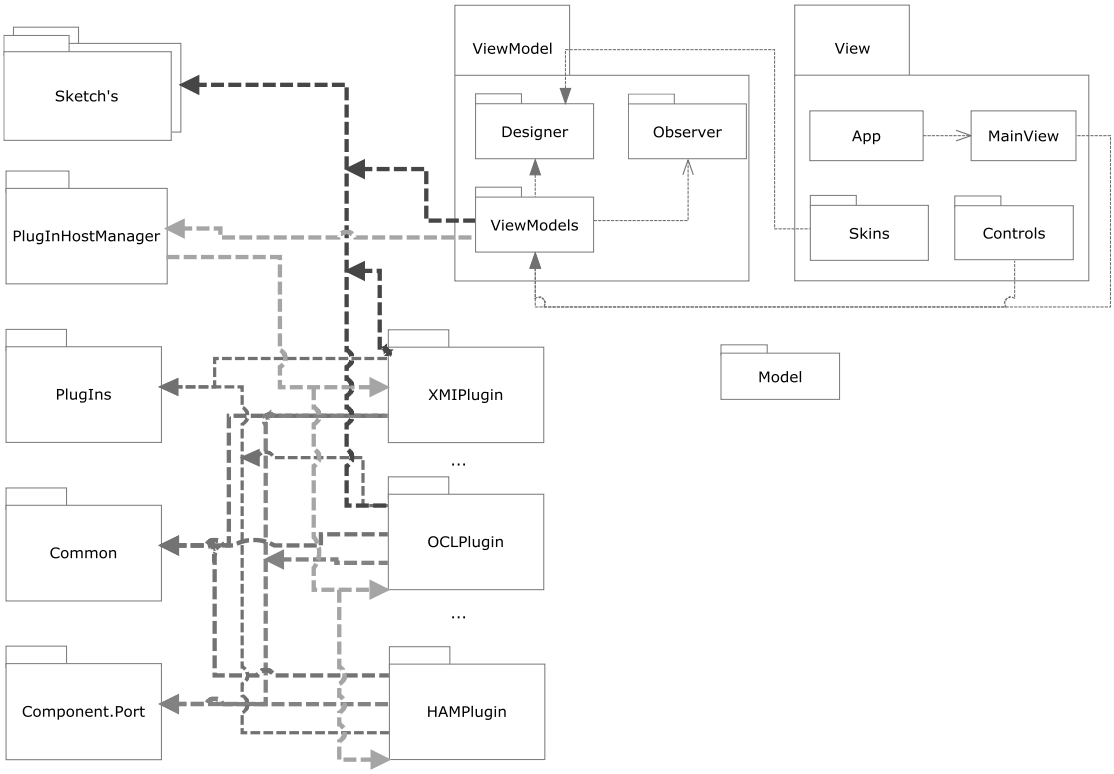

Figura 31. Componentes da aplicação.

A Figura 31 mostra a relação entre os diferentes elementos necessários à implementação da aplicação. De destacar, que não existe somente três *plugins* na aplicação, apenas foram representados três nesta figura a título ilustrativo. Também é importante mencionar que a estrutura dos próprios *plugins* segue uma arquitetura MVVM a fim de facilitar a evolução da interface gráfica do plugin.

4.5 Wireframe

A Figura 32 apresenta o *wireframe* que irá ser utilizado como base para a criação da interface gráfica da aplicação. Em primeiro lugar, na parte esquerda do *wireframe*, onde diz "Tools", é onde irá encontrar-se todas as componentes que foram carregadas através dos plugins. Estas componentes podem desempenhar diferentes funcionalidades, sendo que podem existir componentes cuja única finalidade é a visualização de gráficos, ou realização de consulta, ou abertura dos modelos, etc.

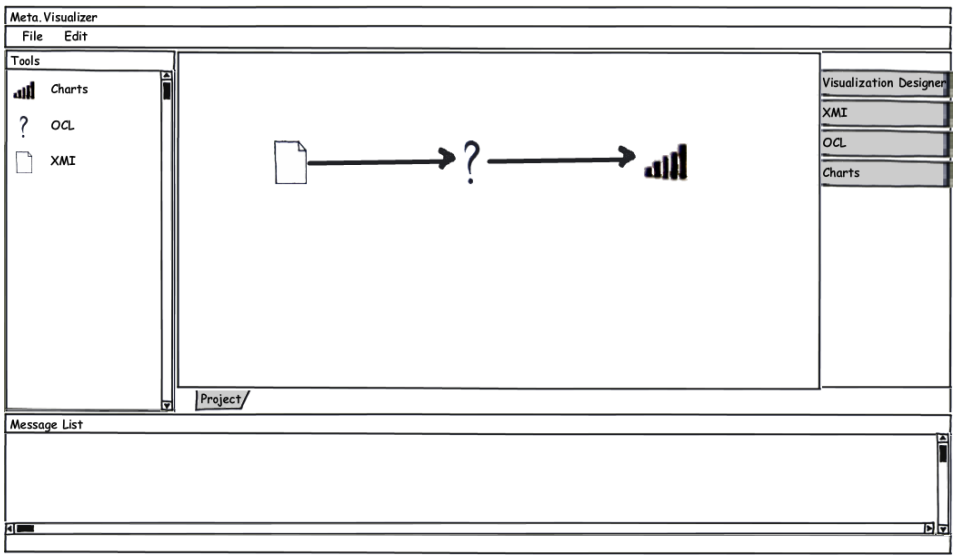

Figura 32. *Wireframe* da aplicação.

Em segundo lugar, na parte inferior do *wireframe*, onde diz "Message List", é onde o utilizador irá ser notificado de possíveis erros, alertas ou mensagens, a fim de não perturbar o utilizador com o aparecimento de janelas de notificação.

Por último, na parte central da aplicação é onde se desenvolve a construção das visualizações e a sua respetiva visualização. É possível observar um separador, na parte inferior do centro do *wireframe*, denominado "Project". A finalidade destes separadores é permitir ter projetos abertos simultaneamente com o intuito de poder comparar projetos e visualizações. Para além destes separadores, existe um conjunto de separadores no lado direito do *wireframe* cuja finalidade é permitir visualizar o conteúdo de um determinado componente que esteja em utilização. O separador denominado "Visualization Designer" está sempre presente em todos os projetos que são criados pelo utilizador e a sua finalidade é fornecer um espaço onde o utilizador possa construir a visualização através da adição de componentes ao espaço de construção e ligação dos portos das componentes. Os restantes separadores permitem aceder aos dados/visualizações das componentes que foram adicionadas.

4.6 Conclusões

Este capítulo definiu várias especificações que irão ser a base para o desenvolvimento da aplicação. A partir destas especificações foi possível delinear a arquitetura sem entrar muito em detalhe sobre a implementação, as linguagens de programação e bibliotecas a serem utilizadas. Com a utilização dos diagramas apresentados, foi conseguido de uma forma muito simples comunicar quais são as funcionalidades que a nossa ferramenta irá implementar e quais são os passos necessários para realizar uma determinada atividade.

5 Implementação

5.1 Introdução

O objetivo da aplicação é satisfazer o conjunto de especificações e requisitos que foram delineados nos capítulos anteriores. Este capítulo descreve o processo de desenvolvimento da aplicação apresentando as componentes relevantes e as suas funcionalidades. A aplicação foi desenvolvida com recurso a linguagem C# e o sistema gráfico *Windows Presentation Foundation* (WPF). Entre as vantagens da utilização desta linguagem é possível mencionar: ser orientada a objetos, robusta e familiar a outras linguagens como Java.

O capítulo começa com uma breve referência aos requisitos e funcionalidades, logo de seguida descreve-se qual foi a tecnologia utilizada, a estrutura geral do projeto e das componentes, as bibliotecas auxiliares que foram utilizadas, quais são as componentes que se encontram disponíveis e por último as dificuldades encontradas na implementação da aplicação.

5.2 Tenologia utilizada

Como foi mencionado na introdução, a tecnologia que irá ser utilizar para o desenvolvimento da aplicação, é a linguagem C#. O motivo para tal é que esta ferramenta irá ser integrada posteriormente numa *suite* de ferramentas desenvolvidas no âmbito do projeto "Meta Sketch". O C# é uma linguagem orientada a objetos, robusta e muito familiar com outras linguagens.

Para a parte gráfica da aplicação foi utilizado o *Windows Presentation Foundation* (WPF). Esta tecnologia possui grande flexibilidade em termos de interfaces já que permite ter duas interfaces completamente diferentes mas que partilham o mesmo comportamento. Também incorpora todas as funções da biblioteca ".NET" acrescentando às interfaces novos recursos, tais como: 3D, animações, gráficos vetoriais, reconhecimento de voz, entre outros. Esta tecnologia incorpora o conceito existente na web da separação entre a interfaces gráficas e o código, o que permite que a interface seja desenvolvida por um *designer* e o código seja feito por um programador especializado de forma independente. Outra característica do WPF é que utiliza os recursos do sistema operativo, a fim de otimizar a performance da interface do utilizador através de *hardware*. Possui também grande flexibilidade, já que permite aos desenvolvedores personalizar os controlos disponíveis, por exemplo, consegue-se criar qualquer tipo de botão com animações em 3D, sem necessidade de escrever código para tal [87].

Os programas em WPF são normalmente compostos por duas partes: um ficheiro XML com características especiais denominado *eXtended Aplication Markup Language* (XAML) e um

ficheiro com código para ".NET". Os ficheiros XAML contêm as diretrizes para a interface e os ficheiros com o código, contêm o tratamento de eventos e em alguns casos as funcionalidades da aplicação [87]. A Figura 33 exibe as principais funcionalidades do WPF.

Figura 33. Principais funcionalidades do WPF.

O WPF tem uma característica muito poderosa de "data binding" que fornece uma forma unidirecional ou bidirecional de sincronização de propriedades. Esta característica permite a utilização do estilo arquitetural MVVM que foi mencionado no capítulo "Estado da arte" e que é uma dos estilos arquiteturais a ser utilizado neste projeto.

No entanto, o WPF também possui problemas, nomeadamente a sua curva de aprendizagem. Em alguns casos e dependendo do nível de experiencia do desenvolvedor, a aprendizagem do WPF pode levar cerca de dois meses. Todavia, os desenvolvedores que já trabalharam com *Windows Forms* sentem mais dificuldades devido a que o WPF é completamente diferente desta tecnologia [88].

5.3 Estrutura geral do Projeto

Em termos de estrutura geral do projeto, Figura 34, foi seguido uma estrutura que venha acomodar o MVVM na aplicação. Para tal, existem três projetos em C#, nomeadamente: "Model", "ViewModel" e "View". O nome dos três projetos advém da separação que é feita pelo MVVM a fim de desvincular a lógica de negócio da interface do utilizador. Por outro lado, a separação em três projetos, permite que seja possível substituir os projetos em qualquer altura do ciclo de desenvolvimento ou durante a manutenção da aplicação. Assim, se o *designer* quiser melhorar a parte da interface gráfica só irá precisar do projeto denominado "View".

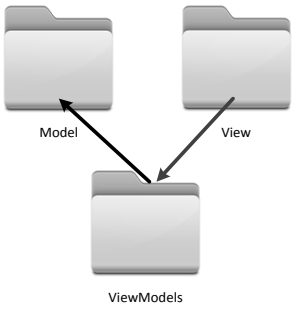

Figura 34. Estrutura geral do projeto.

De salientar que a utilização do MVVM implica que o projeto "view" tenha conhecimento apenas do projeto "ViewModels" para poder fazer o *data binding*, ou seja o processo que estabelece uma ligação entre UI e a logica de negócio [89]. Por sua vez, o projeto "ViewModels" tem apenas conhecimento do projeto "Model" e, por último, o projeto "Model" não tem conhecimento dos dois outros projetos. Isto permite uma abstração a fim de melhorar a manutenção e reduzir o *coupiling* entre os projetos.

Em termos de conteúdo de cada um dos projetos, em primeiro lugar, o projeto "Model" não possui qualquer tipo de conteúdo, isto porque as classes que representam os dados de cada funcionalidade encontram-se em cada um dos componentes e não é necessário armazenar estas classes na aplicação. A razão da existência do projeto "Model" é simplesmente teórica, ou seja, pretende-se preservar a estrutura do MVVM a fim de evitar mal entendidos.

Em segundo lugar, o projeto "ViewModels", é aquele que, como o próprio nome indica, possui as *ViewModels* da aplicação. A Figura 35 apresenta a estrutura interna deste projeto.

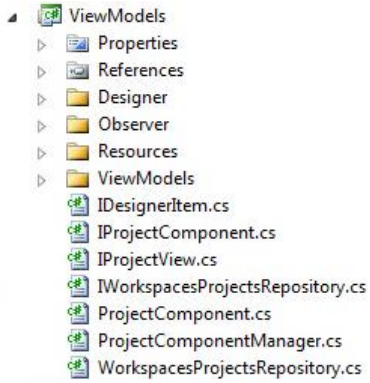

Figura 35. Estrutura interna do projeto "ViewModels".

O projeto "ViewModels" possui quatro pastas. A pasta "Designer" possui todas as classes necessárias para a parte de desenho/construção das visualizações no "Visualization Designer". O "Visualization Designer" é a parte da aplicação onde o utilizador "arrasta" uma ou mais componentes a fim de construir uma visualização. Para a implementação desta parte foi necessário recorrer ao padrão de desenho *Decorator* para permitir realizar o "drag and drop" e

para a parte do borde que cada componente possui quando está selecionado. Também foi necessário recorrer a utilização de *Adorners*. Estes são um tipo especial de FrameworkElement, que é utilizado para fornecer vestígios visuais para o utilizador. Também é possível usar os *Adorners* para adicionar funcionalidades aos elementos e fornecer informações sobre o estado dos mesmos.

Por sua vez a pasta "Observer" possui as interfaces que irão ser utilizadas para implementar o padrão *Observer* nas *ViewModel*. Por outro lado, a pasta "Resources" apenas contém imagens que irão ser utilizadas em alguns dos nos menus de contexto das componentes. Por último, a pasta "ViewModels" possui todas as *ViewModel* que irão ser utilizadas pelo projeto "View". A Figura 36 apresenta, de uma forma bastante resumida, o diagrama de classes do projeto "ViewModels".

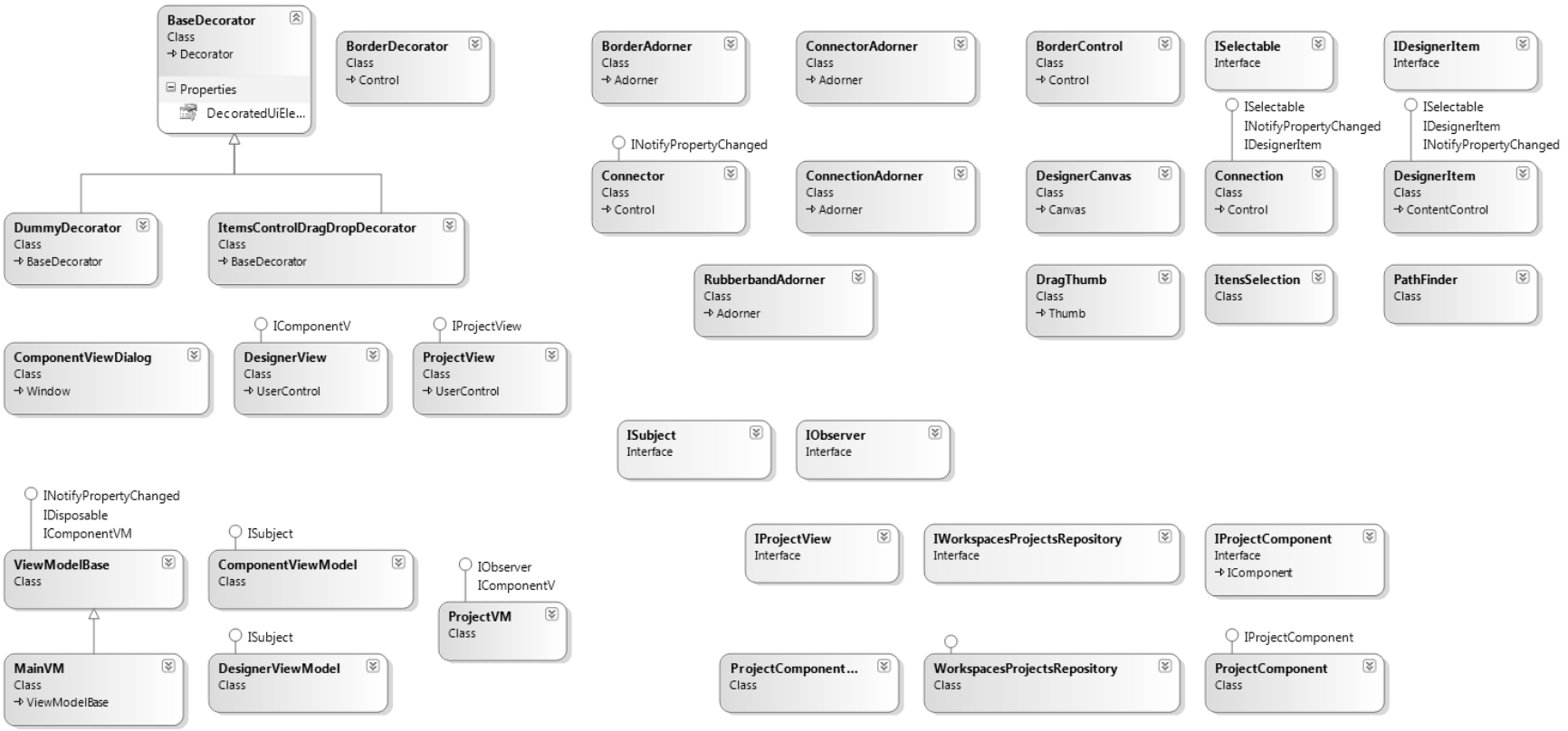

Figura 36. Diagrama de Classes do projeto "ViewModels".

Por último, o projeto "Views" é aquele que possui as vistas da aplicação. Neste projeto são definidos os controlos e os *layouts* das interfaces gráficas que irão ser utilizadas na aplicação. A Figura 37 mostra a estrutura interna deste projeto.

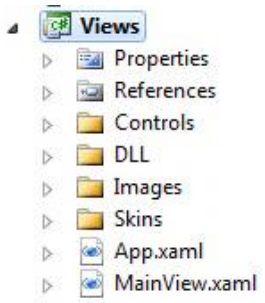

Figura 37. Estrutura interna do projeto "Views".

O projeto "View" possui quatro pastas. A primeira pasta, denominada "Controls", que contém todos os controlos que foram criados. Em WPF, o termo "control" é utilizado identificar qualquer classe que represente um objeto visível numa aplicação WPF. É importante salientar que uma classe que representa um controlo, não precisa de herdar da classe "Control" para tornar-se num objeto visível numa aplicação WPF [90].

Com a utilização da classe "Control" é possível alterar todo o desenho de um controlo durante o tempo de desenho (*design time*) e/ou durante o tempo de execução (*runtime*), permitindo desta forma personalizar os controlos existentes sem reinventar a roda. Nesta pasta também encontram-se os conversores (*Converters*). Estes são utilizados sempre que os dados que estão no *binding* diferem do tipo de dados que o controlo recebe, sendo necessária uma conversão.

Também é possível encontrar nesta pasta, os diálogos, que são utilizados para notificar informações ou receber dados, como por exemplo: o diálogo de "abrir ficheiro" (open file). Os "explorers", que são utilizados apresentar informações e as ferramentas, também se encontram nesta pasta, assim como o "Ribbon" que é utilizado para a barra de ferramentas e as "Views" que contém todas as vistas que a aplicação irá utilizar com a exceção da vista principal que se encontra na raiz do projeto.

A segunda pasta deste projeto é a pasta "DLL". Está pasta contém todas as *Dynamic-link library* (DLL) que o projeto irá utilizar. A terceira pasta "Images" possui todas as imagens que são utilizadas por este projeto. De salientar que as Imagens/ícones específicos das componentes que são adicionadas se encontram no seu respetivo projeto.

Por último, tem a pasta "Skins". Esta pasta possui todas as *ResourceDictionary* que são necessárias para a aplicação. Uma *ResourceDictionary* é um conceito que é suportado pela

classe *ResourceDictionary* que faz parte da tecnologia WPF/Silverlight. Basicamente uma *ResourceDictionary* é um dicionário que possui chaves (*Keys*) que podem ser utilizados tanto em XAML como em código. No XAML, o uso mais comum das *ResourceDictionary* é para definir os *Styles*, animações, transformações, Templates, *DataTemplates*, entre outros, dos controlos [91]. Neste projeto foi definido uma *ResourceDictionary* geral que possui todos os *Styles* dos controlos utilizados, isto permite que, eventualmente no futuro, seja possível alterar dinamicamente os *Styles* dos controlos na aplicação.

A Figura 38 apresenta, de uma forma bastante resumida, o diagrama de classes do projeto "Views".

Figura 38. Diagrama de Classes do projeto "Views".

Portanto, a aplicação principal está dividida nos três projetos anteriormente apresentados. No entanto, falta ainda referir onde está a parte dos *plugins* e o seu funcionamento, assim como outras bibliotecas auxiliares que foram utilizados na aplicação.

Uma das biblioteca auxiliares que foram criadas para aplicação é a biblioteca "PlugIns". Esta biblioteca é utilizada por todos as componentes que pretendem ser utilizadas como *plugins*.

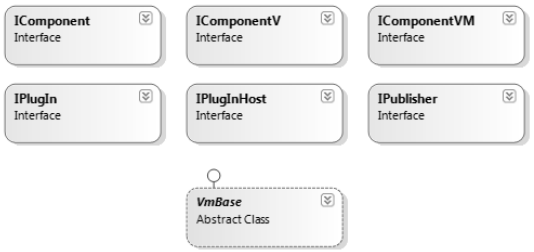

Figura 39. Diagrama de Classes do projeto "PlugIns".

Como é possível observar na Figura 39, esta biblioteca possui seis interfaces e uma classe abstrata. Em primeiro lugar, a interface "IComponent" é utilizada para definir uma componente.

Cada componente possui uma interface "IComponentV", que representa a vista da componente e uma interface "IComponentVM" que representa a *ViewModel* da componente. De salientar que, como foi referido anteriormente no capítulo de especificação, as componentes que são criadas seguem também o estilo arquitetural MVVM. Em segundo lugar, existe também a interface "IPlugin" que possui um conjunto de métodos e propriedades que caracterizam o plugin, tais como: nome, descrição, autor, entre outros. Um plugin possui um hospedeiro (*host*) que é a entidade que "publica" o plugin, dai a existência das interfaces "IPlugInHost" e "IPublisher".

Por último, para além das interfaces referidas anteriormente, existe a classe abstrata "VmBase". Esta classe é utilizada com o intuito de permitir que as classes que representam as *ViewModels* nas componentes possam ter uma superclasse, ou seja, as classes *ViewModels* herdam as propriedades e métodos que estão contidos na classe abstrata "VmBase" e implementam os métodos e propriedades abstratos desta classe. Isto permite apagar código, no sentido em que, a maior parte dos métodos e propriedades que são comuns a todas as componentes já se encontram implementadas nesta classe abstratas.

Outra biblioteca que é utilizada pela aplicação é a biblioteca "PlugInHostManager". Esta biblioteca possui duas classes: "PlugInManager" e "PlugIn". A classe "PlugInManager", como o próprio nome indica, gere todos *plugins* que são adicionados a aplicação e esta encarregue do carregamento dos plugins quando a aplicação é executada. Por sua vez, a classe "PlugIn" possui duas propriedades: a instância do plugin que representa e o caminho onde se encontra o plugin. Esta biblioteca é utilizada no projeto "ViewModels", apresentado anteriormente, pela classe "MainVM".

Por sua vez, a biblioteca "Component.Port" possui todas as interfaces e implementações necessárias para os portos de entrada e saída dos componentes. Um componente pode possuir portos de entrada e/ou de saída. A finalidade destes portos é permitir que a informação que se encontra num porto de saída de uma determinada componente possa estar disponível em outra componente através da ligação do respetivo porto de saída a um ou mais portos de entrada. De salientar que cada porto de entrada/ saída possui um nome e um tipo, sendo que a maior parte dos portos que foram implementadas são do tipo "IData" que irão ser explicados mais a frente.

Para além destes portos que podem já estar predefinidos, também é possível que o utilizador adicione mais portos de entrada ou de saída consoante a componente. A adição destes portos é feita através de botões especiais, nomeadamente: "Add Input Port" e "Add Outpu Port", que se encontram nas vistas das componentes. O desenvolvedor das componentes não é obrigado a que

estas opções estejam disponíveis e também pode limitar o número de portos de entrada ou saída que o utilizador pode adicionar.

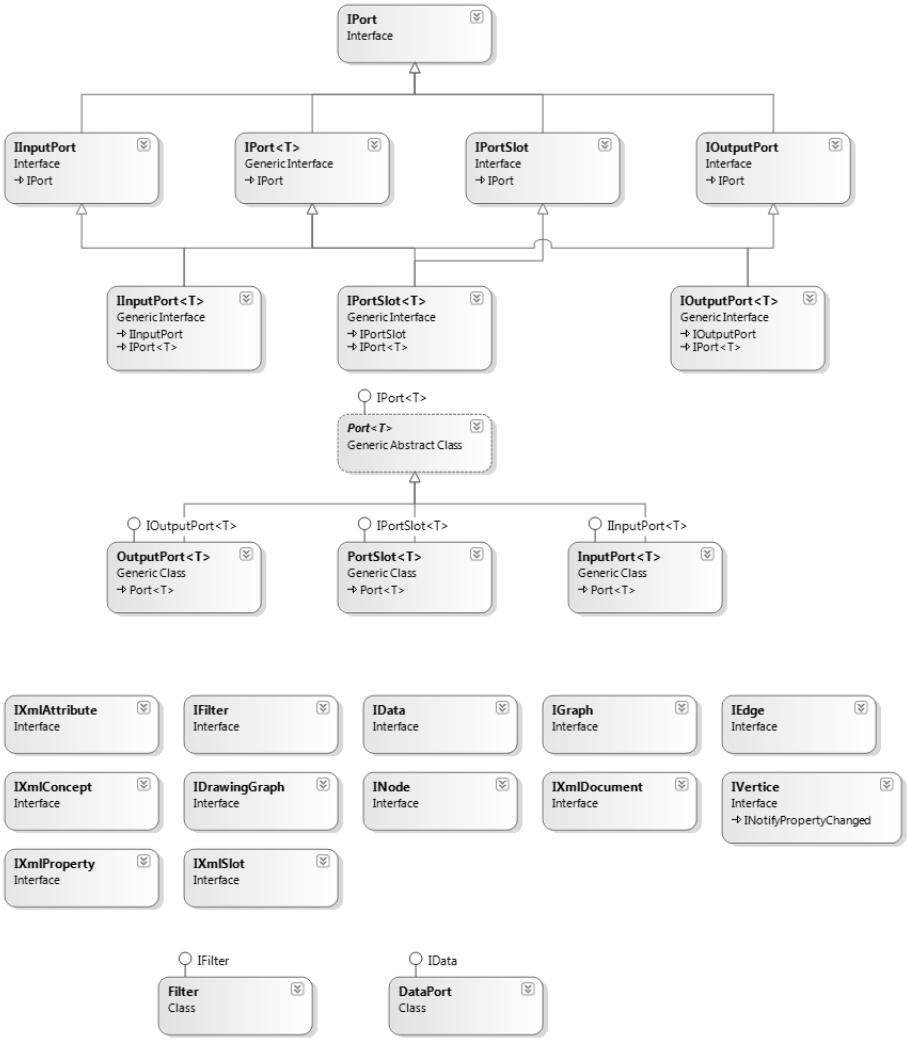

Figura 40. Diagrama de Classes da biblioteca "Component.Port".

A Figura 40 mostra o diagrama de classes biblioteca “Component.Port”. É possível observar que existem três tipos de portas diferentes, designadamente: “IInputPort”, “IOutputPort” e “IPortSlot”. As portas “IInputPort” e “IOutputPort” são as portas de entrada e de saída, respetivamente, que já foram mencionadas anteriormente, e que permitem que a informação que se encontra num porto de saída de uma determinada componente possa estar disponível em outra componente. Por outro lado, o tipo de porta “IPortSlot” é utilizado para colocar a informação na porta de saída.

A fim de compreender melhor a necessidade desta porta, observe-se o seguinte exemplo: suponha-se que temos duas componentes, a componente “X” e “Y”. A componente “X” é uma componente que permite abrir ficheiros XML e possui um porto de saída cuja informação provem do conteúdo de um ficheiro XML e a componente “Y” permite visualizar os dados da

componente “X” em uma tabela, possui um porto de entrada que alimenta a tabela e possui uma porta de saída que representa a célula que está atualmente selecionada. Ora, no caso da componente “X” uma porta do tipo “IPortSlot” pode ser utilizada para colocar os dados na porta de saída desta componente. No caso da componente “Y” se pode utilizar uma porta do tipo “IPortSlot” para colocar o conteúdo da célula que esta atualmente selecionada. Portanto, as portas do tipo “IPortSlot” são úteis quando se pretende colocar dados que são “gerados” pela componente numa porta de saída. Existem casos em que as portas deste tipo não são necessárias, nomeadamente, o caso em que a componente é passiva, ou seja, não altera os dados ou não existe seleção de dados que tenham que ser colocados na porta de saída da componente.

Para além dos portos, a Figura 40 também apresenta um conjunto de interfaces. Estas interfaces são utilizadas pelos portos e representam o tipo de dados que um porto recebe ou envia. Como foi referido anteriormente, a maior parte dos portos são do tipo “IData”, isto porque o tipo “IData” é um tipo “genérico” que encapsula vários tipos de dados através das suas propriedades. As propriedades deste tipo são as seguintes: “Data” que recebe e retorna um dado do tipo “object”, “ListData” que recebe e retorna um enumerado do tipo “IEnumerable <object> ”, “Nodes” que recebe e retorna dados do tipo “INode”, “Graph” que recebe e retorna dados do tipo “IGraph” e ListTuple que recebe e retorna uma lista de tuplos do tipo “IList<Tuple<object, object>>”.

Finalmente tem a biblioteca “Common” que é utilizada por todos os projetos mencionados anteriormente e pelos componentes que são criadas. A razão da existência desta biblioteca deve se ao facto de existir determinadas classes que tanto as componentes como os projetos da aplicação utilizam. Esta biblioteca possui quatro classes:

- “Mediator”: esta classe implementa o padrão comportamental *mediator*. Este padrão é utilizado para enviar mensagens entre os diferentes componentes e projetos da aplicação.
- “RelayCommand”: esta classe implementa a interface *ICommand* e é utlizada quando se pretende a utilização de comandos no *ViewModel*.
- “TextSearchFilter”: esta classe permite que seja possível procurar determinados termos nas listas que são utilizadas pelas *Views* das componentes e da aplicação.
- “TraceEntry”: esta classe é utilizada para permitir que as componentes possam mostrar mensagens de erro, de atenção ou de informação ao utilizador. Se Pode observar o funcionamento desta classe na parte da aplicação onde diz “Message List”.

5.4 Estrutura geral das componentes

Uma componente é uma pequena peça de *software* que foi construída com o objetivo de satisfazer uma necessidade que não é resolvida pela aplicação ou pelas componentes já existentes.

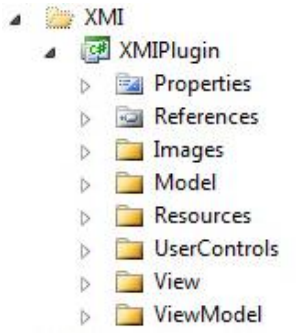

Figura 41. Estrutura de uma componente.

A Figura 41 exibe a estrutura geral de uma componente, neste caso da componente "XMIPlugin". De salientar que as componentes também seguem o estilo arquitetural MVVM, mas que ao contrário de termos um projeto para cada uma das partes que compõe o MVVM, nas componentes temos pastas que fazem essa separação com o intuito de reduzir a complexidade na criação das componentes.

Uma componente normalmente possui seis pastas: "Images" que possui todas as imagens que a componente ira utilizar, "Model" que possui as classes que representam os dados da componente, "Resources" que possui os controlos que foram criados especificamente para a componente e também possui as *ResourceDictionary* necessárias, "View" que possuir todas as vistas que são utilizadas pela componente e "ViewModel" que possui as *ViewModel* das vistas.

O utilizador pode construir a suas próprias componentes e adicionar-lhas a aplicação. Para tal, é necessário que a componente que se pretende construir implemente as seguintes interface:

- "IPlugin": esta interface é implementada na classe "Plugin". Nesta classe podem-se encontrar as seguintes propriedade: nome, descrição, autor, versão, *host* e ícone da componente e os seguintes métodos "CreateComponent()" e "CreateComponent(XmlNodeList aChildNodes)" que são utilizados para criar uma nova componente "vazia" e para criar uma nova componente tendo em conta um conjunto de propriedades que se encontra na variável "aChildNodes".
- "IComponent": esta interface é implementada na classe "Component". Nesta classe podem-se encontrar as seguintes propriedades: "GetView" que retorna a vista da componente, "GetQuickView" que retorna a vista rápida da componente e "GetVM"

que retorna a ViewModel da componente, e os seguintes métodos: “Component()” e “Component(XmlNodeList aChildNodes)” que são utilizados para criar uma nova componente “vazia” e para criar uma nova componente tendo em conta um conjunto de propriedades que se encontra na variável “aChildNodes”

Para além destas interfaces é necessário que as *Views* implementem a interface “IComponentV” a fim de poderem ser utilizadas na aplicação como *Views* das componentes. Da mesma forma, as classes que representam as *ViewModels* têm que implementar a classe abstrata “VmBase”.

De salientar que não é obrigatório implementar uma *View* para a *QuickView*. Uma *QuickView* é basicamente uma vista rápida que pode ser adicionada a interface gráfica de outra componente sem ser necessário realizar as ligações necessárias. Isto permite visualizar de forma rápida os dados que estão na componente através de diferentes visualizações.

5.5 Bibliotecas auxiliares

Para além das bibliotecas anteriormente descritas foram necessárias outro conjunto de bibliotecas cuja implementação já estava finalizada. Uma das bibliotecas necessárias para ser possível ler os modelos e realizar consultas sobre estes foram as Biblioteca “Sketch.MOF” e “Sketch.OCL”. Estas Biblioteca foram utilizadas nas componentes “XMIPlugin” e “OCLPlugin” a fim de ser possível abrir os modelos e realizar consultas sobre os modelos utilizando o OCL. Existem também outras bibliotecas tais como “Sketch.Logging” e “Sketch.CodeGen” que são utilizadas pelas bibliotecas “Sketch.MOF” e “Sketch.OCL” anteriormente descritas. Todavia foi necessário adaptar a biblioteca “Workbench” que faz parte do projeto “MetaSketch” que foi desenvolvido pelo orientador para uma versão muito mais “leve” a qual foi denominado “WorkbenchLight”.

Outras bibliotecas que foram necessárias para a parte visualizações tem a ver com as bibliotecas que permitem gerarem gráficos. Existem diversas bibliotecas, entre as quais: *Telerik*, *ComponentArt*, *Infragistics* e *Visifire*. Todas estas bibliotecas foram escopo de uma análise, que pode ser consultada no anexo V, a fim de apurar qual era a melhor biblioteca que se ajustava aos seguintes requisitos:

- Suporte para diferentes tipos de gráficos em 2D e 3D.
- Suporte para interação nos gráficos e a aparência dos mesmos.
- Performance.
- Facilidades de *Binding*.
- Preço e descontos.
- Licenciamento.

- Controlos adicionais incluídos na biblioteca.

A conclusão a que se chegou com esta análise foi que sempre que se faz um produto de *software* é necessário pensar se irão ser utilizadas as bibliotecas já existentes que permitem poupar trabalho, ou se irão ser desenvolvidas as bibliotecas. No caso de as bibliotecas serem gratuitas, e ter acesso ao código fonte e serem de boa qualidade, quase sempre, não é pensado a possibilidade de fazer as bibliotecas. Mas devido a que encontrar esse tipo de bibliotecas é muito raro, por vezes é necessário decidir entre comprar ou produzir as componentes.

No caso do C# isto é uma decisão que acontece muitas vezes devido a que a maior parte das bibliotecas não são gratuitas e por isso é necessário analisar várias bibliotecas de forma a encontrar a aquela que tenha a melhor relação preço qualidade. Aquela que apresenta a melhor relação preço qualidade é a biblioteca *Visifire*.

Em termos de características da biblioteca *Visifire* é possível enunciar as seguintes:

- Permite criar gráficos agradáveis em Silverlight e WPF em poucos minutos.
- Uma única API para ambas as tecnologias Silverlight e WPF.
- Os controlos do *Visifire* são multi-targeting.
- Pode ser utilizado para aplicações Desktop, Web ou móveis.
- É Compatível com *Microsoft Expression Blend*, permitindo que todos os gráficos sejam editados com auxilio desta ferramenta.
- Permite a geração de gráficos em tempo real e permite que eles sejam atualizados em tempo real, sendo que as propriedades dos gráficos *Visifire* podem ser atualizados em tempo real usando código .Net ou JavaScript.
- É independente da tecnologia do lado do servidor.
- Em termos de gráficos, esta biblioteca suporta 19 gráficos em 2D e o mesmo número em 3D.

Para além destas características também foi possível testar a biblioteca e ver quais dos requisitos anteriormente enunciados eram satisfeitos, Tabela 5.

Tabela 5. Requisitos satisfeitos.

Requisitos	
Gráficos 2D	✓ (19 tipos)
Gráficos 3D	✓ (19 tipos)
Interação	✓
Performance	✓
Aparência	Alta
Binding	✓
Preço	$399.00 Sem suporte

Desconto	20%
Licenciamento	Todas as licenças do produto são permanentes e livres.
Controlos adicionais	✓ (2 controlos adicionais)

Portanto devido as características desta biblioteca e ao seu preço esta é a biblioteca que é a melhor para aplicação que foi desenvolvida.

5.6 Funcionamento da aplicação

Em termos de funcionamento da aplicação, esta permite que o utilizador possa construir visualizações através de componentes. Uma componente, como já foi referido neste capítulo, é uma pequena peça de *software* que foi construída com o objetivo de satisfazer uma necessidade que não é resolvida pela aplicação ou pelas componentes já existentes. Por outro lado, uma componente também pode ser vista como o resultado da agregação de várias componentes num projeto. O utilizador pode construir a suas próprias componentes e adicionar-lhas a aplicação desde que implemente um conjunto de interfaces que foram mencionadas na secção "Estrutura geral das componentes", ou então desde que o abra um projeto como uma componente.

Quando o utilizador pretende adicionar uma componente que não exista na aplicação, primeiro tem que que ir ao painel "Tool" e carregar na pastinha que se encontra lá (Figura 42, A). Depois de carregado irá aparece um diálogo onde o utilizador pode navegar ate ao diretório onde se encontra o plugin. Caso o plugin estiver correto, este irá aparecer no painel "Tool". De salientar que este painel possui dois separadores, o primeiro separador, "Plugins", contém todas as componentes que foram carregadas por meio de um plugin, e o segundo separador, "Component", contém todas as componentes que foram criadas a partir de projetos já realizados. Basicamente, quando se guarda um projeto este mesmo projeto pode ser utilizado como uma componente para outro projeto mais complexo.

Um exemplo da utilização deste tipo de componentes pode ser dado quando é construído um projeto que já possua visualizações e as consultas já estejam definidas faltando somente ligar ao modelo. Assim o utilizador pode reutilizar projetos sem a necessidade de estar a construir todas as visualizações, bastando somente adicionar o respetivo projeto como componente. De salientar que uma componente proveniente de um plugin é diferente de uma componente que provem de um projeto. A diferença está na forma como são construídas e como são tratadas pela aplicação.

Após ter a componente, consegue-se arrastar essa componente para a área do separador "Visualization Designer" (Figura 42, B). Neste separador é onde é construída a visualização através da adição das componentes e ligação dos seus portos de saída e de entrada. De destacar que por cada componente que é adiciona na área de desenho surge um novo separador cujo nome é referente ao nome da componente adicionada (Figura 42, C).

Quando é carregado num separador, é possível ter acesso a informação que a componente possui. Esta informação pode estar sobre a forma gráfica ou textual dependendo da função da componente. Existem componente nas quais são possíveis adicionar "QuickView". Estas vistas têm como objetivo permitir ter diferentes formas de visualizar os dados que se encontram na componente sem recorrer a ligações entre componentes de visualização.

Por último existe uma secção na aplicação (Figura 42, D) na qual o utilizador é notificado sobre possíveis erros, alertas ou notificações resultantes das ações que este realiza na aplicação.

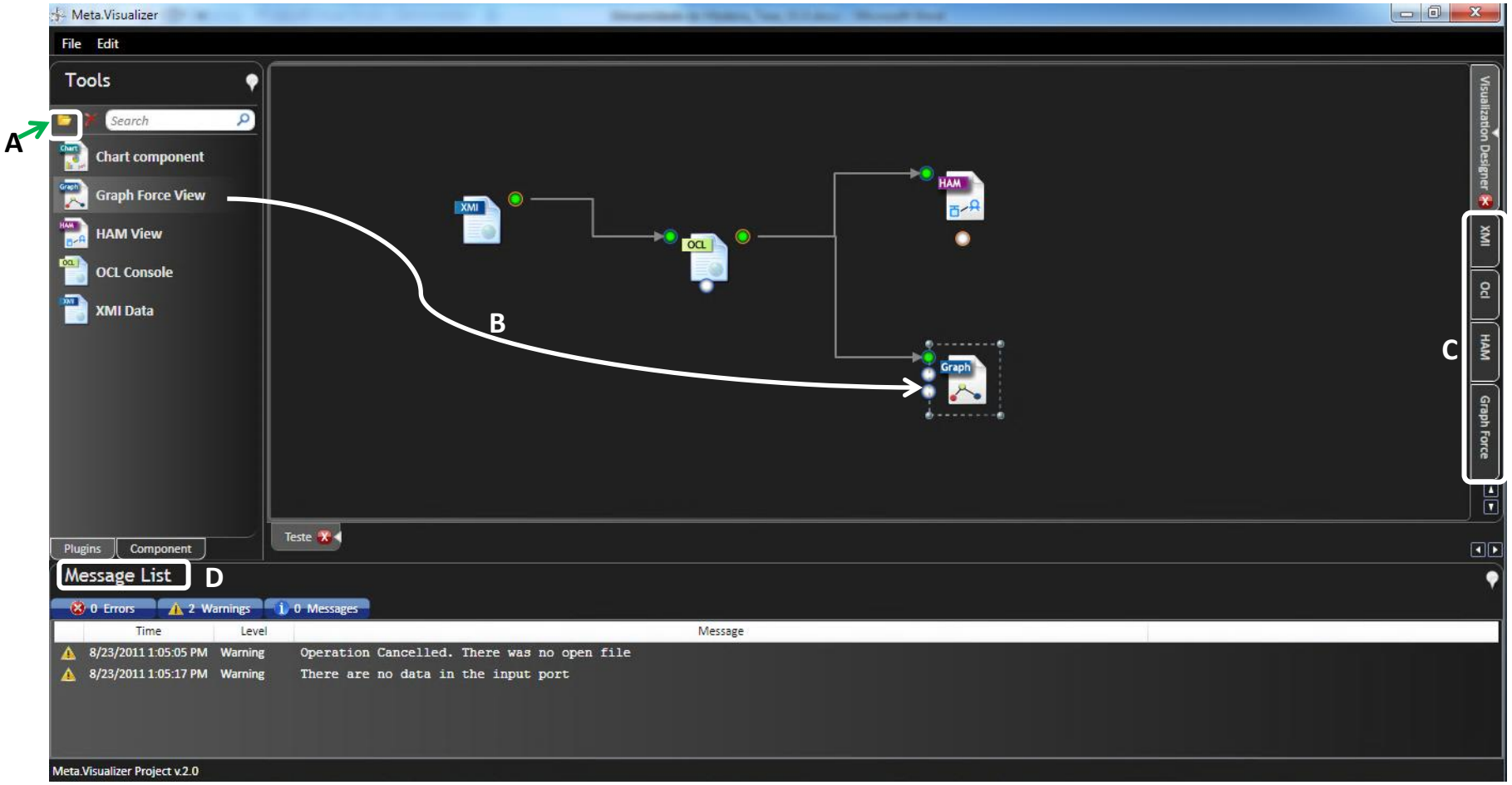

Figura 42. Aplicação, Meta-Visualizer.

5.7 Componentes implementadas

Em termos de componentes implementadas se podem enumerar os seguintes:

- **Componente "Chart":** este componente permite visualizar vários tipos de gráficos através da utilização da biblioteca *Visifire* que foi referida neste capítulo. Permite também adicionar vistas rápidas e pode ser utilizada como uma vista rápida, ou seja, ao arrastar a componente para a parte onde diz "Quick View" ela atua como uma vista rápida cujos dados são os dados que se encontram na componente onde foi adicionada. Nesta componente é possível mudar o tipo de gráfico, o tamanho do mesmo, o título, o tema, e o tipo de vista.
- **Componente "Graph":** este componente permite visualizar os dados através de grafos. Esta componente permite adicionar vistas rápidas e visualizar as propriedades do elemento que esta selecionado, nomeadamente, o nome e o tipo.
- **Componente "HAM":** este componente foi especialmente criada para poder visualizar os resultados das consultas com os ícones que fazem parte do HAM. Para além disto, os elementos que compõem o resultado são desenhados recorrendo a um algoritmo bastante simples de grafos. Ao clicar num dos elementos com o botão direito do rato, consegue-se ter acesso a consultas pré-definidas que podem ser executados sem a necessidade de escrever consultas em OCL. Por outro lado, quando é selecionado um elemento consegue-se visualizar as propriedades deste elemento, nomeadamente, o nome e o tipo. Também é possível adicionar vistas rápidas a fim de visualizar dados quantitativos como por exemplos o número de atores que existem na visualização.
- **Componente "Heat Map":** este componente permite visualizar os dados através de um *heat map*. A componente permite adicionar vistas rápidas e também pode ser utilizada como uma vista rápida, ou seja, ao arrastar a componente para a parte onde diz "Quick View" ela atua como uma vista rápida cujos dados são os dados que se encontram na componente onde foi adicionada.
- **Componente "Literal":** este componente é uma componente de teste em que o utilizador pode usar a fim de introduzir valores na porta de entrada ao qual a porta de saída desta componente está ligada.
- **Componente "Merge Tuple":** como o próprio nome indica, este componente encarrega-se de fundir duas listas, com o mesmo número de elementos, em uma única lista composta por tuplos binários.
- **Componente "OCL":** este componente permite realizar consultas com recurso à utilização do OCL. Nesta componente consegue-se selecionar o contexto no qual se

pretende fazer a consulta e o tipo de dado que se pretende que seja devolvido após a execução da consulta. Também é possível adicionar quatro portas de entrada cujos dados são: o meta-modelo, parâmetro, instância do modelo e por último o classificador de uma instância.

- **Componente "Table":** este componente permite visualizar os dados em tabelas. De salientar que este componente é uma componente de teste da aplicação. Eventualmente no futuro irá possuir maior utilidade.
- **Componente "XMI":** Este componente permite abrir ficheiros cuja extensão é ".xmi", ".cmof", ".uml" e ".skt". Estes ficheiros contêm os modelos. Para além de abrir os modelos, esta componente possui três tipos de vistas, nomeadamente, a vista do modelo onde é visualizado o modelo, a vista do meta-modelo em que se pode visualizar o respetivo meta-modelo e por último a vista XML que permite visualizar o conteúdo do ficheiro. Por outro lado, consegue-se adicionar vistas rápidas (*Quick Views*) a esta componente a fim de visualizar rapidamente os dados que se encontram no modelo. Por exemplo, é possível adicionar uma vista rápida de gráficos a esta componente e se for aberto o modelo do MOF consegue-se visualizar o número de classes, interfaces entre outras coisas, que este modelo possui através de um gráfico. Este componente permite adicionar três portas de saída cujos dados são: o meta-modelo, instância do modelo que esta atualmente selecionada e por último o classificador da instância que esta selecionada.
- **Componente "XML":** este componente permite abrir e visualizar ficheiros cuja extensão é ".xml".

5.8 Dificuldades

Diversas dificuldades foram encontradas no decorrer deste trabalho. Em primeiro lugar, a maior dificuldade encontrada foi a imposta pela tecnologia. O tempo de aprendizagem do WPF é aproximadamente de dois meses, mas no caso deste projeto foi mais tempo devido a pouca experiencia nesta tecnologia, cerca de três a quatro meses. Na opinião do autor, foi complicado elaborar pequenos projetos que permitissem avançar com o projeto em geral. Muitos dos problemas encontrados foram resolvidos após muitas tentativas, a maior parte destes problemas eram problemas de "ausência" de informação nas partes da aplicação onde a informação deveria aparecer. Alguns destes problemas foram resolvidos com auxílio de conversores ou utilização de outros tipos de dados que fossem compatíveis com *binding* dos controlos.

Em segundo lugar, existiram dificuldades em implementar o estilo arquitetural MVVM. Na elaboração do projeto foi difícil perceber este padrão porque existiam diversos autores que tinham diversas implementações deste padrão e cada autor resolvia um problema específico

numa abordagem MVVM que o outro autor não descrevia como se podia resolver. No entanto, depois de muitas tentativas, foi possível encontrar um pequeno projeto em MVVM elaborado pelos engenheiros da *Microsoft* que permitiu desvendar e esclarecer muitas das dúvidas que existiram acerca da implementação deste estilo. Todavia, foi necessário uma pequena modificação do MVVM a fim de ser possível a implementação da aplicação. Esta modificação passa por existirem algumas vistas inseridas no projeto "ViewModels" a fim de poderem ser injetadas num controlo dinamicamente através do *binding*.

Em terceiro lugar, outra dificuldade encontrada foi a de tentar elaborar uma biblioteca que gera-se gráficos. Esta biblioteca precisaria de aproximadamente três há quatro meses de desenvolvimento (partindo do zero) devido a complexidade dos algoritmos de *layout*. Esta complexidade esteve presente quando foi elaborada uma pequena biblioteca que permitisse gerar grafos. Esta biblioteca pode ser encontrada na componente "GraphForce".

Por último, foi encontrado no final do projeto uma dificuldade que não estava prevista. Esta dificuldade tem origem nas consultas que se realizam com recurso ao OCL. Por um lado, quando a biblioteca começou a ser testada para realizar consultas em um modelo real, foi detetados muitos problemas de implementação, nomeadamente, determinados "comandos" das consultas não funcionarem.

Por outro lado, e apás resolver alguns destes problemas relacionados com os "comandos", para realizar aquilo que pode ser uma consulta simples como por exemplo: saber o número de papéis que um determinado ator possui num modelo HAM, tornou-se uma consulta complexa e que requer um bom conhecimento, não só do OCL mas também do meta-modelo. Este tipo de consulta originou aquilo a que foi chamado de "comboios OCL" devido ao comprimento da consulta OCL que tinha que ser realizada. No entanto, existe um outro projeto que está a ser desenvolvido cujo objetivo é a implementação de uma biblioteca para o *Imperative OCL* [92].

5.9 Conclusões

Com esta aplicação pretende-se dar um contributo para a definição de um novo conjunto de ferramentas que tiram partido da tecnologia MDE, possibilitando novas formas de lidar e usar modelos no desenvolvimento de *software*. Com esta ferramenta, os desenvolvedores podem extrair de uma forma mais fácil as informações que estão contidas nos modelos através de consultas OCL.

Embora tenham sido apresentadas neste capítulo um conjunto de dificuldades, quase todas elas já foram ultrapassadas durante o desenvolvimento do projeto. Num futuro muito próximo,

pretendesse substituir a biblioteca do OCL pela biblioteca do *Imperative OCL* permitindo assim reduzir a complexidade das consultas.

Acima de todo com esta aplicação fica criada a possibilidade de obter uma melhor compreensão e validação dos modelos através da utilização de diferentes técnicas de visualização aqui apresentadas.

6 Caso de estudo

6.1 Introdução

De forma a por em prática a proposta aqui apresentada, foi desenvolvida uma aplicação que permita retirar a informação que se encontra nos modelos com o intuito de melhorar a compreensão e validação dos modelos. Neste capítulo pretendesse realizar um caso de estudo que permita perceber se esta ferramenta ajuda ou não aos objetivos propostos na dissertação.

Este capítulo descreve quais são os objetivos do caso de estudo, qual foi a abordagem seguida para realizar este caso de estudo e culmina com uma análise aos resultados obtidos.

6.2 Objetivos

Foram proposto como objetivos a analisar neste caso de estudo os seguintes objetivos:

- Permitir extrair informação dos modelos.
- Utilizar uma linguagem de consulta, como o OCL.
- Visualizar a informação obtida através da consulta utilizando diversas visualizações.
- Navegar pelo modelo, utilizando a visualização, com o intuito de encontrar informação relevante para o utilizador.

6.3 Abordagem

A fim de atingir os objetivos aqui enunciados, é necessário de um modelo. O modelo a ser utilizado é o modelo da superestrutura UML.

O UML é uma linguagem que abrange um conjunto amplo e diversificado de domínios de aplicação, sendo que nem todas as capacidades de modelação desta linguagem são necessárias em todos os domínios ou aplicações. Devido a este facto, foram criados quatro níveis de conformidade [93]:

- **Nível 0 (L0):** este nível de conformidade é formalmente definido na infraestrutura do UML. Este nível contém uma única linguagem de modelação que prevê os tipos de modelação baseados em classes encontradas na maior parte das linguagens de programação orientadas a objetos.
- **Nível 1 (L1):** este nível acrescenta novas linguagens e amplia as capacidades fornecidas pelo nível 0. Especificamente, este nível acrescenta as linguagens para casos de utilização, interações, estruturas ações e atividades.

- **Nível 2 (L2):** este nível estende as linguagens existentes no nível 1 e acrescenta novas linguagens para o desenvolvimento, modelação de máquinas de estado e perfis.
- **Nível 3 (L3):** este nível representa todo o UML. Estende as linguagens que se encontram no nível 2 e acrescenta novas linguagens para a modelação dos fluxos de informação, *templates* e pacotes de modelos.

Para o caso de estudo irá ser utilizado o nível 2 na sua versão 2.1.2, Figura 43.

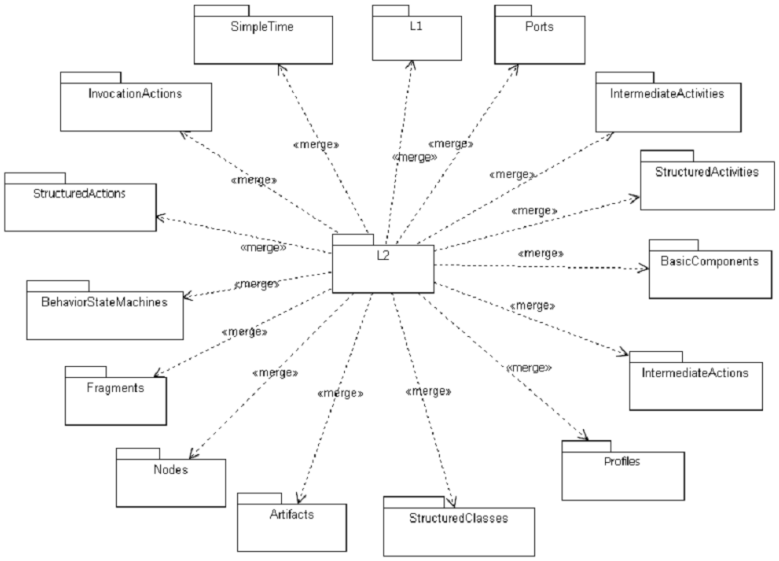

Figura 43. Nível 2 do UML.

Como é apresentado na Figura 43 existem muitos pacotes (packages) que o nível 2 possui. Isto torna difícil a tarefa de compreender este nível em um curto espaço de tempo. A melhor forma de perceber o conteúdo de nível é ver a especificação que a OMG oferece. No entanto, esta especificação é longa, cerca de 750 páginas, e com varias terminologias. O que se pretende é utilizar a aplicação a fim de ajudar ao utilizador a compreender este modelo.

De forma a ser possível extrair as informações que se encontram neste modelo, em primeiro lugar irá ser utilizada uma componente que permita carregar o modelo. A componente que permite isto é a componente "XMI". Logo depois de estar aberto o modelo que é pretendido, é possível adicionar uma vista rápida, por exemplo, uma que permita visualizar gráficos, a fim de saber quais os elementos contidos no modelo e a quantidade desses elementos.

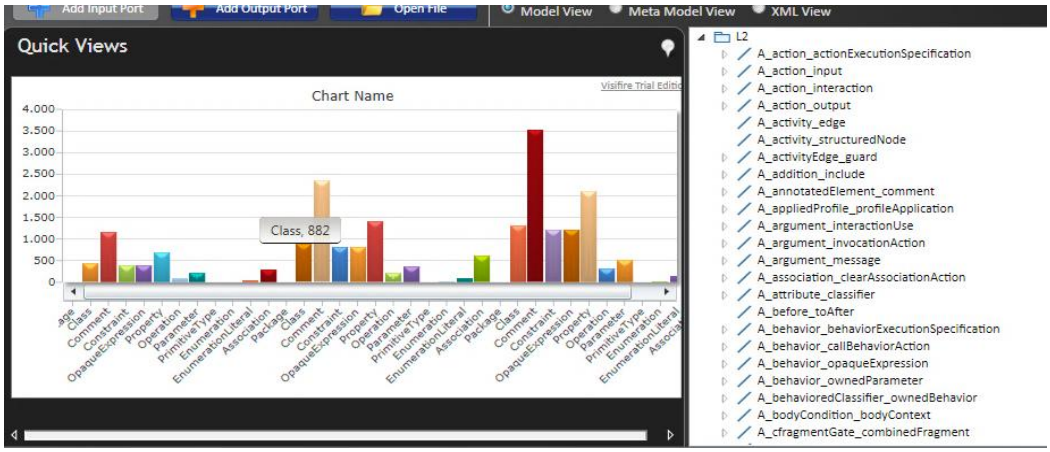

Figura 44. 1ª Situação - visualização rápida.

A Figura 44 apresenta o resultado das ações que foram descritas anteriormente. Em primeiro lugar, este modelo possui muitos itens sendo necessário modificar o comprimento da vista rápida. Em segundo lugar, com a utilização de uma vista rápida é possível observar que o item que possui mais elementos é o "comment". Portanto, a partir do carregamento de um modelo e com a utilização de uma vista simples consegue-se contabilizar o número de itens que o modelo possui e quantos elementos pertencem a cada item.

Assim, consegue-se ter uma base para formular uma consulta, por exemplo, quais são as classes que existem no modelo, ou quantas classes possuem instâncias de uma determinada classe. Com estas consultas é possível saber, por exemplo, qual o impacto de uma determinada alteração de uma classe no modelo. Por último, é possível alterar o tipo de gráfico ou adicionar outra visualização que permita obter uma outra perspetiva, possibilitando uma melhor compreensão do modelo que está sendo analisado.

Portanto, já é possível presenciar a uma parte da potencialidade da ferramenta. Todavia, não foi satisfeito todos os objetivos aqui propostos. A fim de obter dados mais detalhados sobre um aspeto particular de um modelo é necessário realizar consultas sobre o modelo. A componente que permite realizar consultas é a componente "OCL". Como já foi descrito no capítulo anterior, esta componente permite realizar consultas nos modelos através da linguagem OCL.

A fim de realizar consultas é necessário ligar a componente "XMI" a componente "OCL" através dos seus portos a fim da componente "OCL" possa ter acesso ao modelo. A primeira consulta que irá ser realizada é saber quais são as classes que existem no modelo, para tal escreve-se a seguinte consulta:

```
Class.allInstances ()->collect( c | c.name)
```

Depois de executar esta consulta obtém-se o seguinte resultado:

```
"{ Comment, DirectedRelationship, LiteralSpecification,
LiteralInteger, LiteralString, LiteralBoolean, LiteralNull,
Constraint, ElementImport, TypedElement, Feature,
RedefinableElement, Generalization, Parameter, StructuralFeature,
Slot, PackageImp...}"
```

O resultado obtido é bastante extenso sendo que o cumprimento da lista que é retornada é de 205 elementos. Mais uma vez é complicado analisar todos estes elementos um a um sendo que é necessário algo que permita visualizar estes elementos.

Suponha-se então que pretende-se saber quantos atributos possui cada classe. Ora se for analisado "a mão" as 205 classes que foram retornadas pela consulta anterior irá ser despendido muito tempo. Uma forma de evitar esta análise manual é através da adição de mais uma componente "OCL" a fim de realizar mais uma consulta. De salientar que é necessário ligar a componente "OCL" que foi adicionada à componente "XMI".

```
Class.allInstances()->collect(c |c.oclAsType(Class).ownedAttribute-
>size())
```

Ao executar esta consulta se obtém o seguinte resultado:

```
{2, 2, 0, 1, 1, 1, 0, 3, 4, 1, 2, 3, 3, 4, 1, 3, 3, 2, 1, 1, 0, 5,
0, 1, 2, 1, 1, …}
```

No entanto, se pode observar que é muito confuso realizar o mapeamento dos atributos de cada classe, visto que são muitos. Ora a melhor forma de fazer este mapeamento e compreender melhor o que se está a observar é através de uma visualização. É possível então adicionar uma componente que permita gerar gráficos, mas primeiro é necessário adicionar uma componente que permita "mapear" os dois resultados obtidos, para tal, é adicionado a componente "Merge Tuple" e realizadas as devidas conexões. A Figura 45 ilustra como é que fica a construção da visualização no separador "Visualization Designer".

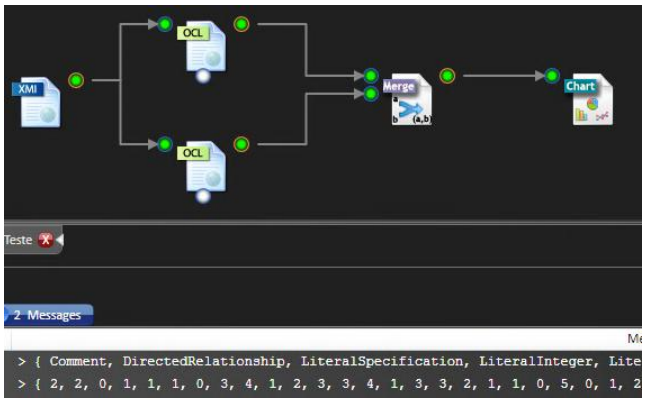

Figura 45. 2ª Situação – construção da visualização.

De salientar que com a utilização da componente "Merge Tuple" foi realizado um mapeamento entre os dois resultados. No entanto, a compreensão dos resultados que se encontram neste mapeamento ainda leva tempo, dai a necessidade de uma visualização.

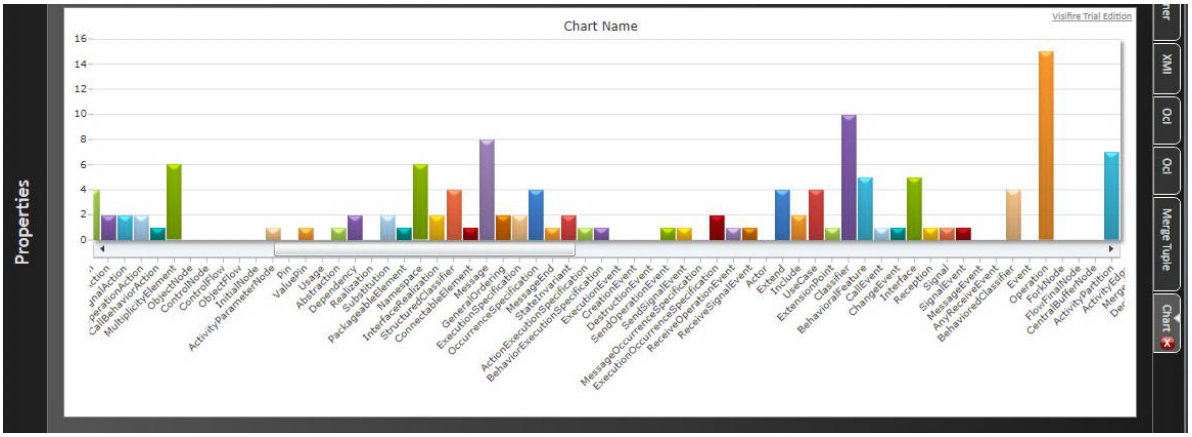

Figura 46. 2ª Situação - visualização do resultado através de gráficos.

A Figura 46 exibe o resultado obtido a partir das duas consultas anteriores, é com muita facilidade que consegue-se ver que a classe "Operation" é uma das classes que possui mais atributos, um total de 15 atributos. Com a utilização de consultas semelhantes é possível explorar o porquê de uma determinada classe possuir tantos atributos, ou então, em um outro modelo, por exemplo, um modelo HAM, porque é que um determinado autor possui tantos papéis.

Para além deste tipo de visualizações, é possível adicionar uma componente "Heat Map" e ter uma outra perspetiva. A

Figura 47 exibe a utilização de um *Heat Map* com os dados que resultaram das duas consultas anteriores. Em primeiro lugar, as "bolinhas" que possuem um vermelho mais "vivo" são as classes que possuem mais atributos. Em segundo lugar, as "bolinhas" que estão a preto são as classes que não têm atributos. Por último, o *Heat Map* aqui apresentado já se encontra ordenado, o que facilita ainda mais a sua compreensão.

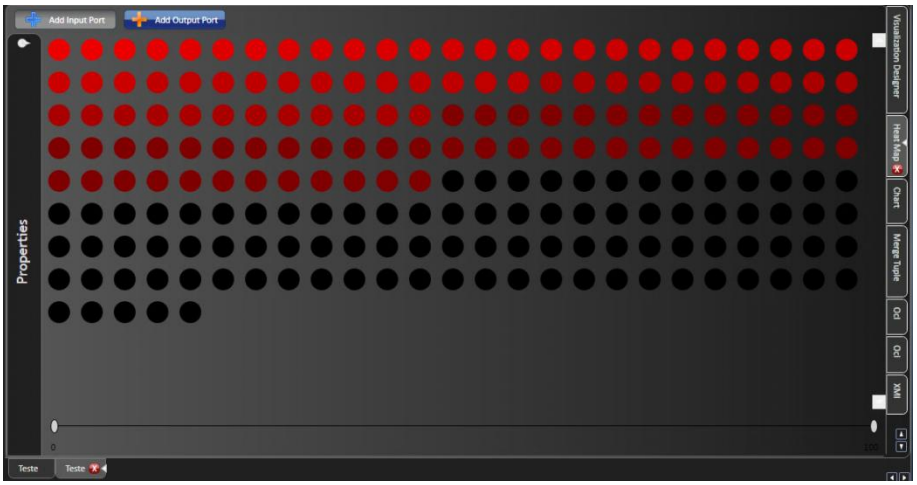

Figura 47. 2ª Situação - visualização do resultado através de Heat Map.

Por último, um dos objetivos propostos neste caso de estudo é o de permitir a navegação pelo modelo, utilizando a visualização, com o intuito de encontrar informação relevante para o utilizador. Este objetivo encontra-se parcialmente implementado na ferramenta. Para demonstrar a sua utilidade utilizou-se um outro modelo, o modelo que foi desenvolvido no âmbito projeto de modelação de processos de uma secção do governo regional e que motivou a elaboração deste projeto.

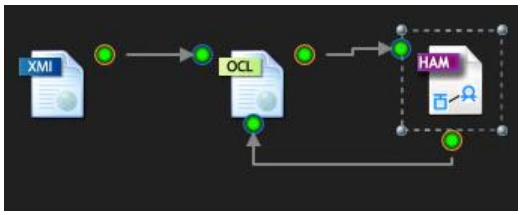

Figura 48. 3ª Situação – construção da visualização.

A Figura 48 ilustra a construção da visualização. Em primeiro lugar, são precisas três componentes, nomeadamente: "XMI", "OCL" e "HAM". As duas primeiras componentes já foram referidas, sendo que a terceira componente permite visualizar os resultados utilizando as figuras que fazem parte de HAM.

Em segundo lugar, é necessário ligar os portos de entrada e de saída para que as componentes possuam os dados de que precisam. De salientar que a componente HAM possui uma porta de saída, a qual é ligada a uma porta de entrada da componente "OCL". Esta porta destina-se a ser utilizada como uma porta onde são enviados consultas OCL já pré-definidas. A ideia é que o utilizador não tenha que saber OCL para poder navegar pelo modelo. Apenas existe um conjunto de consultas predefinidas que o utilizador pode executar. No entanto, se não existir a consulta que se pretende nas consultas pré-definidas, o utilizador sempre pode ir a componente "OCL" escrevê-la e executá-la. Por último, é necessário executar a seguinte consulta:

```
Actor.allInstances()->collect( c | c.name)
```

Esta consulta devolve todos os atores que estão contidos no modelo e como já está ligada a componente de visualização é possível observar os resultados obtidos na Figura 49.

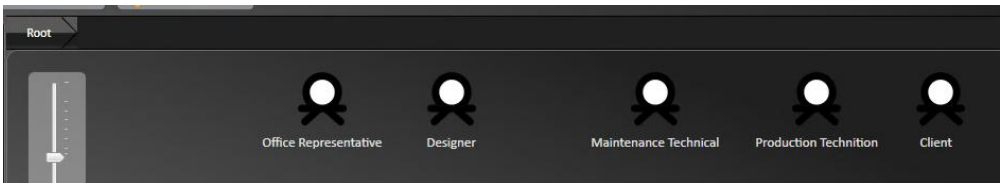

Figura 49. 3ª Situação - visualização do resultado através da componente "HAM".

Se for realizado duplo clique, por exemplo, no ator "Designer" consegue-se navegar e visualizar todas as relações existentes com este ator.

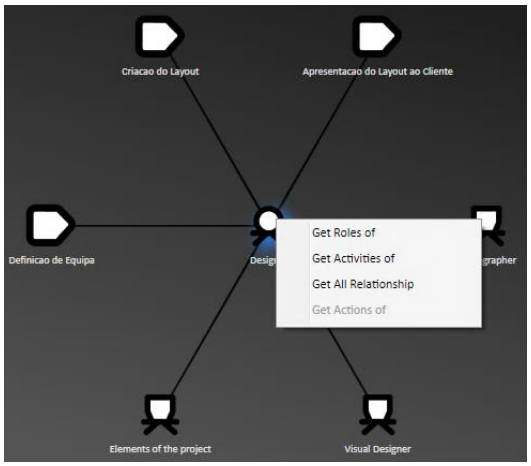

Figura 50. 3ª Situação - navegação na componente "HAM".

A Figura 50 apresenta todas as relações do ator "Designer". Também é possível querer saber todos os papéis, atividades ou relações deste ator através de um menu de contexto. Assim, é possível navegar no modelo através das visualizações.

No entanto, como foi referido anteriormente, o objetivo da navegação encontra-se parcialmente implementado, porque não utiliza consultas OCL para gerar a navegação. Esta sendo realizado trabalhos com o intuído de no futuro próximo seja possível fazer estas navegações com recurso a consultas OCL.

6.4 Conclusões

Como foi possível observar neste capítulo o caso de estudo permitiu perceber até que ponto a ferramenta satisfaz os objetivos propostos. De facto, a ferramenta permite melhorar a compreensão de grandes modelos como o que foram utilizados nos exemplos e a utilização de

diferentes visualizações permite ter diferentes perspetivas dos dados que estão ser analisados. Por sua vez, a utilização de consultas nos modelos vem ajudar em muito a compreender aquilo que se encontra no modelos, no entanto, sem recurso a visualização torna-se muito difícil perceber os resultados obtidos, caso estes são extensos.

Com a utilização da visualização e das consultas foi possível contribuir em muito para a compreensão dos modelos. Aliado a estas duas vantagens, a navegação através dos resultados que se encontram nas visualizações permite contribuir para o melhoramento da compreensão e validação dos modelos. No entanto, a parte da navegação tem que ser melhorada a fim de poder utilizar a vantagem das consultas em OCL. Outra parte que tem que ser melhorada/acrescentada são as visualizações em grafos e novas visualizações.

Todavia, foram realizados teste de utilização para saber quais as dificuldades encontradas pelos utilizadores, sendo que os resultados destes testes foram muito positivos permitindo melhorar a funcionalidade da ferramenta. Acima de tudo, esta ferramenta atingiu quase na sua totalidade todos os objetivos proposto sendo necessário apenas algumas melhorias que irão acontecer com o amadurecimento e utilização da ferramenta.

7 Conclusões e Perspetivas Futuras

7.1 Conclusões

Ultimamente, se tem vindo a observar um aumento das aplicações de *software* que estão a ser desenvolvidas. Cada vez mais, estas aplicações assumem uma complexidade tremenda e em muitos casos até as aplicações de médio porte são difíceis de compreender. Ter uma boa documentação permite acompanhar o ciclo de desenvolvimento de *software* e melhorar a compreensão dos sistemas que estão a ser desenvolvidos.

A utilização de modelos tem vindo aumentar nos últimos anos, não somente porque são utilizados para documentar os aspetos do desenho, mas também porque servem como base para a geração de parte ou a totalidade dos sistemas de informação. No entanto, um modelo é uma vista parcial de todo o sistema que representa e isto dificulta a extração e compreensão das informações do sistema de forma rápida e simples, o que consequentemente dificulta a própria validação e compreensão dos modelos.

Com esta dissertação foi possível constatar que, com o auxílio de técnicas de visualização é possível melhorar a compreensão dos sistemas que estão ou já foram desenvolvidos. A ferramenta que foi desenvolvida, contribui para a definição de um conjunto novo de ferramentas que tiram partido da tecnologia MDE, possibilitando formas inovadoras que permitem lidar e utilizar modelos no desenvolvimento de *software*. Em concreto, a ferramenta desenvolvida, permite extrair, de uma forma mais fácil, as informações sobre os modelos, utilizando para tal consultas em OCL e utilizar o resultado dessas consultas para criar as visualizações que o utilizador pretende. Com a utilização das consultas OCL foi possível apurar o verdadeiro poder do OCL como linguagem de consulta. No entanto, o OCL em si oferece muitas dificuldades, entre as quais a complexidade de aprendizagem da linguagem e a complexidade na realização de consultas. Estas duas dificuldades são claramente um entrave para uma maior utilização e disseminação do OCL.

Por outro lado, a elaboração desta dissertação permitiu aprofundar muitos dos conceitos que são abordados durante a licenciatura e mestrado de engenharia de informática. Também permitiu desenvolver novas competências no âmbito da programação orientada a objetos, o que é uma mais-valia na experiencia profissional do autor. Muito trabalho foi realizado durante este ano para concretizar esta dissertação e foi necessário ultrapassar muitos obstáculos, nomeadamente: a dificuldade de aprendizagem do WPF, problemas que surgiram no desenvolvimento da aplicação associados ao WPF, mau funcionamento da biblioteca OCL que foi utilizada no projeto, entre outros. No entanto, com a ajuda do Professor Doutor Leonel Nóbrega e da equipa

da *ZON Service Engineering* foi possível ultrapassar muitos dos obstáculos e também melhorar a aplicação que foi desenvolvida.

Acima de tudo, os objetivos aqui propostos foram cumpridos ficando a possibilidade de obter uma melhor compreensão e validação dos modelos através da utilização de diferentes técnicas de visualizações.

7.2 Perspetivas Futuras

A realização de qualquer trabalho deixa sempre possibilidades de evolução no futuro, em particular, no futuro pretende-se:

- Adicionar o *Imperative OCL* a fim de simplificar e potenciar as consultas realizadas nos modelos.
- Criar a possibilidade de utilizar uma linguagem de consulta visual a fim de facilitar a construções das consultas em OCL.
- Estender as visualizações, adicionando novas visualizações e novas funcionalidades a ferramenta.
- Dar suporte ao *Query/View/Transformation.*
- Melhorar as componentes a fim de ser possível que qualquer utilizador crie a sua própria componente na aplicação.

Bibliografia

[1] G. B. J. R. Ivar Jacobson, The Unified Software Development Process, Addison-Wesley, 1999.

[2] OMG, “OMG Unified Modeling LanguageTM (OMG UML), Infrastructure,” Maio 2010. [Online]. Available: http://www.omg.org/spec/UML/2.3/Infrastructure. [Acedido em 12 08 2011].

[3] S. W. Liddle, “Model-Driven Software Development”.

[4] J. V. Igor Sacevski, “Introduction to Model Driven Architecture (MDA),” 2007.

[5] D. C. Schmidt, “Model-Driven Engineering,” *IEEE computer,* vol. 39, pp. 25-31, 2006.

[6] S. Serrão, “Modelação de Processos Centrada em Actividades,” 2010.

[7] L. L. Constantine, “Human Activity Modeling,” *Engineering,* 2008.

[8] S. A. White, “Introduction to BPMN,” 2006.

[9] Microsoft, “msdn,” [Online]. Available: http://msdn.microsoft.com/en-us/vcsharp/aa336706. [Acedido em 14 Agosto 2011].

[10] A. Nathan, “Windows Presentation Foundation,” 2007.

[11] J. M. D. Matos, “MetaSketch OCL Interpreter,” Funchal, 2008.

[12] K. McGarry, O contexto dinâmico da informação, Brasilia: Briquet de Lemos, 1999.

[13] S. Furgeri, Representação de informação e conhecimento: estudo das diferentes abordagens entre a ciência da informação e a ciência da computação, Campinas, 2006.

[14] “Visualization (computer graphics),” 11 Septembro 2010. [Online]. Available: http://en.wikipedia.org/wiki/Visualization_(computer_graphics). [Acedido em 05 Outubro 2010].

[15] Z. Kaidi, “Data visualization,” National University of Singapore.

[16] R. I. Bull, *Model Driven Visualization: Towards a Model Driven Engineering Approach for Information Visualization,* University of Victoria, 2008.

[17] S. Few, “Data Visualization Past, Present, And Future,” Perceptual Edge, 2007.

[18] M. Friendly, “A Brief History of Data Visualization,” em *Handbook of Computational Statistics: Data Visualization*, Toronto, Springer-Verlag, 2006.

[19] D. J. D. Michael Friendly, “Milestones in the History of Thematic Cartography Statistical Graphics, and Data Visualization,” [Online]. Available: http://www.datavis.ca/milestones/. [Acedido em 22 Maio 2011].

[20] M. Friendly, “Milestones in the history of thematic cartography, statistical,” 2009.

[21] “Information visualization,” 14 Septembre 2010. [Online]. Available: http://en.wikipedia.org/wiki/Information_visualization. [Acedido em 23 Septembre 2010].

[22] “Solid Source,” [Online]. Available: http://www.solidsourceit.com/products/SolidSX-source-code-dependency-analysis.html. [Acedido em 18 Maio 2011].

[23] “Scientific Computing and Imaging Institute,” [Online]. Available: http://www.sci.utah.edu/research/visualization.html. [Acedido em 18 Maio 2011].

[24] “wikipedia,” 2010. [Online]. Available: http://en.wikipedia.org/wiki/Scientific_visualization. [Acedido em 23 Novembro 2010].

[25] “educause,” Outubre 2007. [Online]. Available: http://net.educause.edu/ir/library/pdf/ELI7030.pdf. [Acedido em 05 Outubre 2010].

[26] C. D. Hansen, The visualization Handbook, United States of America: Elsevier, 2005.

[27] W. H. a. A. U. Chun-houh Chen, Handbook of Data Visualization, Berlin: Springer, 2008.

[28] N. I. Julie Steele, Beautilful Visualization, O'Reilly, 2010.

[29] U. o. Huddersfield. [Online]. Available: http://hospitality.hud.ac.uk/studyskills/usingData/AnalysingData/typesData.htm. [Acedido em 28 Agosto 2011].

[30] N. S. G. J. Andreia Hall. [Online]. Available: http://www2.mat.ua.pt/pessoais/ahall/me/files/1Introdu%C3%A7%C3%A3o.pdf. [Acedido em 22 Maio 2011].

[31] S. K. M. J. D. &. S. Card, Readings in information visualization, Morgan Kaufmann, 1999.

[32] Y. Park, “Design Elements & Principles,” University of Texas - Austin.

[33] J. Stout, “Design, Exploring the Elements & Principles,” Iowa State University, Iowa, 2000.

[34] C. Jirousek, “Art, Design and Visual thinking,” 1995. [Online]. Available: http://char.txa.cornell.edu/language/principl/principl.htm. [Acedido em 23 Janeiro 2011].

[35] P. K. Cindy Kovalik, “Visual Literacy,” [Online]. Available: http://www.educ.kent.edu/community/VLO/design/principles/perspective/index.html.

[Acedido em 28 Agosto 2011].

[36] L. A. Cary Jensen, Reference, Harvard Graphics 3: The Complete, McGraw-Hill Osborne Media, 1991.

[37] “types of graphs,” 2011. [Online]. Available: http://www.typesofgraphs.com/. [Acedido em 08 07 2011].

[38] “Microsoft Office,” [Online]. Available: http://office.microsoft.com/pt-pt/excel-help/tipos-de-graficos-disponiveis-HA001233737.aspx?CTT=1#BMbarcharts. [Acedido em 07 07 2011].

[39] “Many Eyes,” [Online]. Available: http://www-958.ibm.com/software/data/cognos/manyeyes/. [Acedido em 08 07 2011].

[40] P. Wied, “heatmap.js,” 2011. [Online]. Available: http://www.patrick-wied.at/static/heatmapjs/. [Acedido em 8 7 2011].

[41] T. d. Grafos, “José de Oliveira Guimarães”.

[42] A. A. Cardozo, “Hyperbolic Tree,” Universidade Católica de Pelotas, 2011.

[43] [Online]. Available: https://158317233402888923 6-a-1802744773732722657-s-sites.googlegroups.com/site/abhishekiitr/InfoViz2.png?attachauth=ANoY7cqxvhonDoOkipiu6Ukqcst_VI9B-6pc6ACt5msMIoT896kUMQoU0fbqbj0LLy7-MXCMyL_b2n5-WAC6zU9eGONdDmdiKsm42tqG2ei0q058r-Z-k6KKez1aypMbRT8ukUeJNh. [Acedido em 2010 Septembro 28].

[44] H. Chen, “Toward design patterns for dynamic analytical data visualization,” Proceedings of SPIE, 2004.

[45] T. M. H. Reenskaug, “MVC,” 12 Junho 2010. [Online]. Available: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html. [Acedido em 13 Agosto 2011].

[46] Smalltalk. [Online]. Available: www.smalltalk.org.

[47] “PARC - Palo Alto Research Center,” [Online]. Available: http://www.parc.com/.

[48] G. G. R. Gomes, “Tábula : uma framework para o desenvolvimento de aplicações REST,” Portugal, 2008.

[49] G. M. Hall, Pro WPF and Silverlight MVVM, Apress, 2010.

[50] M. Hunter, “Tidying the House: The MVPC Software Design Pattern,” 2006.

[51] IBM, “Many Eyes,” [Online]. Available: http://www-

958.ibm.com/software/data/cognos/manyeyes/.

[52] Yahoo, “flickr,” [Online]. Available: http://www.flickr.com/.

[53] Google, “youtube,” [Online]. Available: http://www.youtube.com/.

[54] M. W. F. v. H. J. K. M. M. Fernanda B. Viégas, “Many Eyes: A Site for Visualization at Internet Scale,” IEEE Computer Society, 2007.

[55] “Gephi:Wiki,” [Online]. Available: http://wiki.gephi.org. [Acedido em 2010 10 23].

[56] Adobe, “photoshop,” [Online]. Available: http://www.photoshop.com/.

[57] “Prefuse Documentation,” 23 Agosto 2007. [Online]. Available: http://prefuse.org/doc/manual/. [Acedido em 26 Outubro 2010].

[58] C. M. Collins, 7 Júlio 2009. [Online]. Available: http://faculty.uoit.ca/collins/research/docuburst/index.html. [Acedido em 03 08 2011].

[59] Addison-Wesley, “The Unified Software Development Process,” 1999.

[60] D. Ferreira, “Meta.Tracer - MOF with traceability,” Funchal - Portugal, 2009.

[61] A. A. o. Science, “science,” 2008. [Online]. Available: http://www.science.org.au/nova/016/016box02.htm. [Acedido em 2011 Agosto 19].

[62] H. Stachowiak, Allgemeine Modelltheorie, Springer, 1973.

[63] J. V. Igor Sacevski, “Introduction to Model Driven Architecture (MDA),” Salzburg, 2007.

[64] H. B. P. B. Patrick Graessle, UML 2.0 in Action, PacktPub, 2005.

[65] D. Ferreira, “Meta.Tracer - MOF with traceability,” Portugal, 2009.

[66] J.-M. Favre, “Megamodelling and Etymology - A story of Words: from MED to MDE via MODEL in five millenniums”.

[67] J. Bézivin, “On the unification power of models,” *Software & Systems Modeling,* vol. 4, pp. 171-188, 2005.

[68] D. Frankel, “What Does "Model-Driven" Mean?,” *MDA Journal,* 2006.

[69] S. W. Liddle, “Model-Driven Software Development”.

[70] T. K. Colin Atkinson, “Model-Driven Development: A Metamodeling Foundation,” *IEEE Software,* pp. 36 - 41, 2003.

[71] M. Gally, “What is MDD/MDA and where will it lead the software development in the future?,” 2007.

[72] J. d. Haan, “MDE - Model Driven Engineering - reference guide,” 15 Janeiro 2009. [Online]. Available: http://www.theenterprisearchitect.eu/archive/2009/01/15/mde---model-driven-engineering----reference-guide. [Acedido em 5 Dezembro 2010].

[73] wikipedia, “Modeling language,” 7 Junho 2011. [Online]. Available: http://en.wikipedia.org/wiki/Modeling_language. [Acedido em 13 Agosto 2011].

[74] P. K. J. V. Arie van Deursen, “Domain-Specific Languages”.

[75] B. Meyer, “The positive spin.,” *American Programmer,* 1997.

[76] R. Bortolini, “Business Process Modeling Notation,” 2006.

[77] L. L. Constantine, “Human Activity Modeling: Toward A Pragmatic Integration of Activity Theory and Usage-Centered Design,” *HUMAN-CENTERED SOFTWARE ENGINEERING,* vol. 1, n.º Human-Computer Interaction Series, pp. 27-51, 2009.

[78] K. K. Hausi A. Muller, “Rigi A System for Programming-in-the-large,” *Proceedings 1989 11th International Conference on Software Engineering,* pp. 80-86, 1989.

[79] H. A. M. M A D Storey, “Manipulating and documenting software structures using SHriMP views,” *Proceedings of the International Conference on Software Maintenance,* p. 275, 1995.

[80] J.-M. F. R. Ian Bull, “Visualization in the context of model driven engineering,” em *International Workshop on Model Driven Development of Advanced User Interfaces, MDDAUI 2005@ MODELS*.

[81] R. M. Daniel Henrique Alves Lima, “Entendendo OCL. Apresentação e utilização da Object Constraint Language”.

[82] J. H. S. K. Ali Hamie, “Interpreting the Object Constraint Language,” *Software Engineering Conference 1998 Proceedings 1998 Asia Pacific,* pp. 288-295, 1998.

[83] W. K.-M. D.H.Akehurst, “UML/OCL – Detaching the Standard Library,” 2010.

[84] T. L. Tamás Vajk, “Imperative OCL Compiler Support for Model Transformations,” *7th International Symposium of Hungarian Researchers on Computational Intelligence,* p. 166–178.

[85] M. K. Fabian Buttner, “Problems and Enhancements of the Embedding of OCL into QVT ImperativeOCL,” *Proceedings of the 8th International Workshop on OCL Concepts and Tools (OCL 2008) at MoDELS 2008,* vol. 15, 2008.

[86] N. J. N. H. C. G. C. C. P. G. S. Leonel Nóbrega, “The Meta Sketch Editor: a Reflexive Modeling Editor,” *Springer-Verlag,* pp. 199-212, 2006.

[87] B. S. e. R. Sonnino, “Introdução ao WPF,” Microsoft, 9 outubro 2006. [Online]. Available: http://msdn.microsoft.com/pt-br/library/cc564903.aspx. [Acedido em 2011 Agosto 18].

[88] K. Januszewski, “Hitting the Curve: On WPF and Productivity,” 2006 Abril 5. [Online]. Available: http://blogs.msdn.com/b/karstenj/archive/2006/04/05/curve.aspx. [Acedido em 19 Agosto 2011].

[89] microsoft, “msdn,” [Online]. Available: http://msdn.microsoft.com/en-us/library/ms752347.aspx. [Acedido em 22 Agosto 2011].

[90] microsoft, “msdn,” 2011. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb613548.aspx. [Acedido em 21 Agosto 2011].

[91] Microsoft, “msdn,” [Online]. Available: http://msdn.microsoft.com/en-us/library/cc903952(v=vs.95).aspx. [Acedido em 22 Agosto 2011].

[92] L. Telo, “TOWARDS STANDARD MODEL-TO-MODEL TRANSFORMATION,” 2011.

[93] OMG, “OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2,” 2007.

[94] OMG, “Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification,” 2008.

[95] J. d. Haan, “the enterprise architect,” 15 Janeiro 2009. [Online]. Available: http://www.theenterprisearchitect.eu/archive/2009/01/15/mde---model-driven-engineering----reference-guide. [Acedido em 8 Dezembro 2010].

[96] oracle, “oracle,” [Online]. Available: http://www.oracle.com/technetwork/java/index.html. [Acedido em 14 Agosto 2011].

[97] R. I. Bull, Model Driven Visualization: Towards a Model Driven Engineering Approach for Information Visualization, University of Waterloo, 2002.

[98] W. Aigner, Interactive Visualization of Time-Oriented Treatment Plans and Patient Data, Viena: Instituto de Tecnologia de Software e Sistemas Interactivos na Universidade Técnica de Viena, 2003.

[99] A. Kumar, “interfacewithdesign,” 2009. [Online]. Available: http://sites.google.com/site/interfacewithdesign/portfolio/infomation-visualization-patterns. [Acedido em 10 Outubro 2010].

[100] karl, “.NET Developer Guidance,” 14 Novembro 2010. [Online]. Available: http://blogs.msdn.com/b/kashiffl/archive/2010/11/14/mvvm-technical-description.aspx. [Acedido em 02 08 2011].

[10 D. B. Tekinerdoğan, “Model-Driven Software Development,” [Online]. Available:
1] http://www.cs.bilkent.edu.tr/~bedir/CS587-MDSD/. [Acedido em 4 Agosto 2011].

[10 D. S. Frankel, “What Does "Model-Driven" Mean?,” *MDA Journal,* 2006.
2]

[10 T. Kühne, “Matters of (Meta-) Modeling,” *Software Systems Modeling,* vol. 5, pp. 369-385,
3] 2006.

Anexo I - Estudo das visualizações

Neste anexo é apresentado algumas das visualizações que foram estudadas no contexto da dissertação.

1.1 Gráfico de Linhas – Variações

Os gráficos de linhas são úteis para mostrar tendências ao longo do tempo ou categorias ordenadas.

Gráfico de linhas com marcadores: esta variação, Figura 51, é muito útil, principalmente quando existem vários pontos de dados e a respetiva ordem de apresentação é importante. No entanto, se existirem várias categorias ou os valores forem aproximados, recomenda-se a utilização de um gráfico de linhas sem marcadores [38].

Gráfico de linhas empilhadas e linhas empilhadas com marcadores: este gráfico, Figura 51, pode ser utilizado para mostrar a tendência da contribuição de cada valor ao longo do tempo ou categorias ordenadas, sendo que este também pode incluir marcadores. No entanto, não é recomendado o uso deste quando existem muitas linhas, já que este fator irá dificultar a visualização e compreensão do gráfico [38].

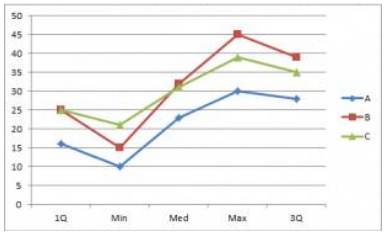

Figura 51.Gráfico de linhas empilhadas com marcadores.

Gráfico de linhas em 3D: este gráfico, Figura 52, mostra cada linha ou coluna de dados como uma fita em três dimensões. Este gráfico possui um eixo horizontal, vertical e de profundidade que podem ser modificados [38].

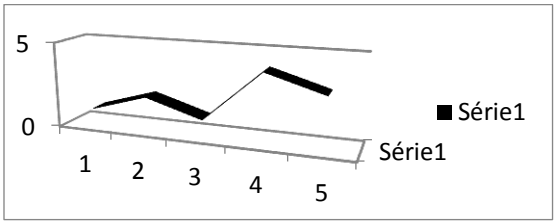

Figura 52. Gráfico de linha em 3D.

1.2 Gráficos de Barras – Variações

Os gráficos de barras ilustram comparações entre itens individuais. Todas as variações aqui apresentadas são respetivas aos gráficos de barras horizontais, não entanto, as mesmas variações podem ser encontradas nos gráficos de barras verticais, ou gráficos de colunas [38].

Gráfico de barras agrupadas e barras agrupada em 3D: esta variação, Figura 53, permite comparar valores entre categorias. Num gráfico de barras agrupadas, as categorias são normalmente organizadas ao longo do eixo vertical e os valores ao longo do eixo horizontal. Uma barra agrupada num gráfico 3D mostra os retângulos horizontais no formato 3-D e não apresenta os dados em três eixos [38].

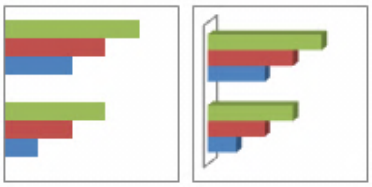

Figura 53. Gráfico de barras agrupadas e barras agrupadas em 3D.

Barras empilhadas e barras empilhadas em 3D: esta variação, Figura 54, mostra a relação dos itens individuais com o total. Uma barra empilhada num gráfico 3-D apresenta os retângulos horizontais no formato 3-D e não apresenta os dados nos três eixos [38].

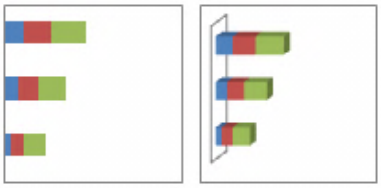

Figura 54. Gráfico de barras empilhadas e barras empilhadas em 3D.

Barras empilhadas a 100% e barras empilhadas a 100% em 3D: este tipo de gráfico, Figura 55, compara a percentagem com a qual cada valor contribui para o total das categorias. Uma barra 100% empilhada num gráfico 3-D apresenta os retângulos horizontais no formato 3-D e não apresenta os dados em três eixos [38].

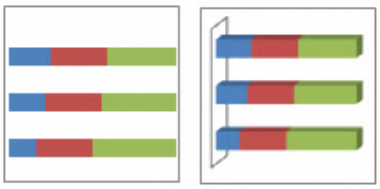

Figura 55. Barras empilhadas a 100% e barras empilhadas a 100% em 3D.

Cilindros, cones e pirâmides horizontais: estes gráficos, Figura 56, mostram e comparam os dados exatamente da mesma forma. A única diferença reside no facto de estes tipos de gráfico apresentarem formas cilíndricas, cónicas e piramidais em vez de retângulos horizontais [38].

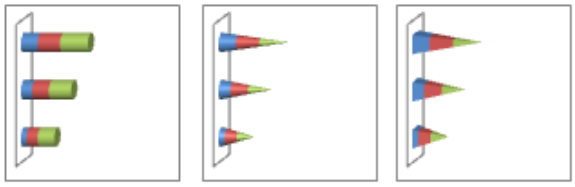

Figura 56. Cilindros, cones e pirâmides horizontais.

1.3 Gráficos circulares – Variações

Os gráficos circulares mostram o tamanho dos itens numa serie de dados, proporcional à soma dos itens, sendo que os pontos de dados de um gráfico circular são mostrados com uma percentagem do total do círculo.

Circular de circular e barra circular: estas variações, Figura 57, mostram os valores definidos pelo utilizador e que são extraídos do gráfico circular principal e combinados em um gráfico circular secundário ou num gráfico de barras empilhadas. Estes gráficos são úteis quando se pretende facilitar a distinção de sectores pequenos do círculo principal [38].

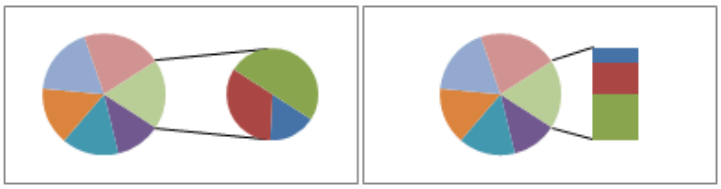

Figura 57. Circular de circular e barra circular.

Gráfico circular destacado e gráfico circular destacado em 3-D: estes gráficos circulares, Figura 58, mostram a contribuição de cada valor relativamente ao total, realçando os valores individuais. Os gráficos circulares destacados podem ser apresentados no formato 3-D [38].

Figura 58. Gráfico circular destacado e gráfico circular destacado em 3-D.

1.4 Gráficos de Área

Os gráficos de área, Figura 59, realçam a magnitude das alterações ao longo do tempo, e podem ser utilizados para chamar a atenção para o valor total ao longo de uma tendência. Por exemplo, os dados que representem lucros ao longo do tempo podem ser representados num gráfico de área para realçar o lucro total [38].

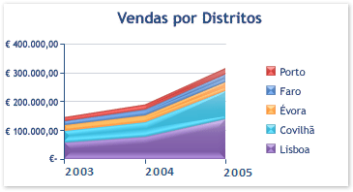

Figura 59. Gráfico de área.

Os gráficos de área dividem-se nos seguintes subtipos:

Gráficos de Área 2-D e área 3-D: estas duas variações dos gráficos de área, Figura 60, apresentam a tendência dos valores ao longo do tempo ou outros dados de categorias [38].

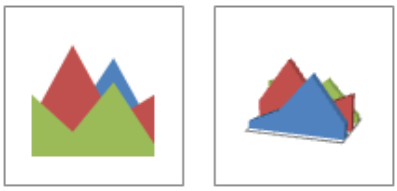

Figura 60. Área 2-D e área 3-D.

Gráficos de Área empilhada e área empilhada em 3-D: estas variações, Figura 61, apresentam a tendência da contribuição de cada valor ao longo do tempo ou outros dados de categoria [38].

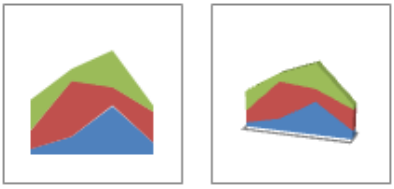

Figura 61. Área empilhada e área empilhada em 3-D.

Gráficos de Área 100% empilhada e área 100% empilhada em 3-D: estas variações, Figura 62, apresentam a tendência da percentagem com que cada valor contribui ao longo do tempo [38].

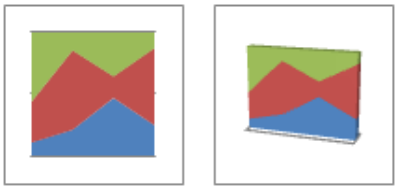

Figura 62. Área 100% empilhada e área 100% empilhada em 3-D.

1.5 Gráficos de dispersão (XY)

Os gráficos de dispersão, Figura 63, mostram as relações entre os valores numéricos de várias séries de dados ou representam dois grupos de números como uma série de coordenadas XY.

Estes gráficos têm dois eixos de valores, mostrando um conjunto de dados numéricos ao longo do eixo horizontal (eixo dos x) e outro ao longo do eixo vertical (eixo dos y). Normalmente, os gráficos de dispersão são utilizados para apresentar e comparar valores numéricos, tal como dados científicos, estatísticos e de engenharia [38].

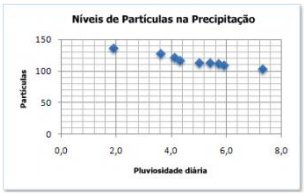

Figura 63. Gráficos de dispersão (XY).

Os gráficos de dispersão dividem-se nos seguintes subtipos:

Dispersão apenas com marcadores: este tipo de gráfico, Figura 64, compara pares de valores. É recomendado usar um gráfico de dispersão com marcadores de dados, mas sem linhas, já que a ligação dos pontos dificulta a leitura dos dados [38].

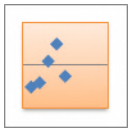

Figura 64. Dispersão apenas com marcadores.

Dispersão com linhas suaves e dispersão com linhas suaves e marcadores: este tipo de gráfico, Figura 65, apresenta uma linha suave que liga os pontos de dados. As linhas suaves podem ser apresentadas com ou sem marcadores [38].

Figura 65. Dispersão com linhas suaves e dispersão com linhas suaves e marcadores.

Dispersão com linhas retas e dispersão com linhas retas e marcadores: este tipo de gráfico, Figura 66, apresenta linhas de conexão retas entre os pontos de dados [38].

Figura 66. Dispersão com linhas retas e dispersão com linhas retas e marcadores.

1.6 Gráficos de cotações

Como o nome indica, os gráficos de cotações são frequentemente utilizados para ilustrar as flutuações dos valores das ações. Contudo, estes gráficos também podem ser utilizados para dados científicos. Por exemplo, é possível utilizar um gráfico de cotações para indicar as flutuações das temperaturas diárias ou anuais [38].

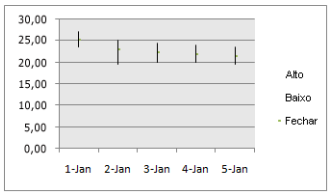

Figura 67. Gráficos de cotações.

Os gráficos de cotações dividem-se nos seguintes subtipos:

Máximo-mínimo-fecho: o gráfico de cotações de máximo-mínimo-fecho, Figura 68, é utilizado frequentemente para ilustrar preços de cotações. Requer três séries de valores na seguinte ordem: máximo, mínimo e, em seguida, fecho [38].

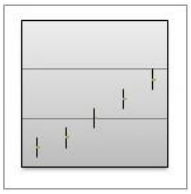

Figura 68. Gráfico Máximo-mínimo-fecho.

Abertura-máximo-mínimo-fecho: este tipo de gráfico de cotações, Figura 69, requer quatro séries de valores pela ordem correta (abertura, máximo, mínimo e, em seguida, fecho) [38].

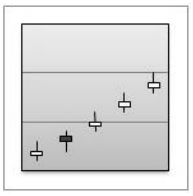

Figura 69. Gráfico Abertura-máximo-mínimo-fecho.

Volume-máximo-mínimo-fecho: este tipo de gráfico de cotações, Figura 70, requer quatro séries de valores pela ordem correta (volume, máximo, mínimo e, em seguida, fecho). Mede o volume utilizando dois eixos de valores: um para as colunas que medem o volume e o outro para os preços das ações [38].

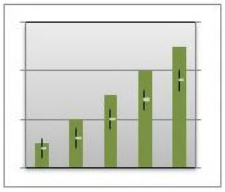

Figura 70. Gráfico Volume-máximo-mínimo-fecho.

Volume-abertura-máximo-mínimo-fecho: este tipo de gráfico de ações requer cinco séries de valores pela ordem correta (volume, abertura, máximo, mínimo e, em seguida, fecho) [38].

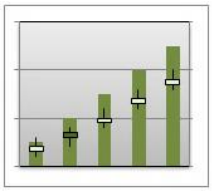

Figura 71. Gráfico Volume-abertura-máximo-mínimo-fecho.

1.7 Gráficos de superfície

Um gráfico de superfície, Figura 72, é útil quando se pretende encontrar combinações ótimas entre dois conjuntos de dados. Como num mapa topográfico, as cores e padrões indicam áreas no mesmo intervalo de valores [38].

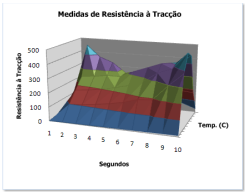

Figura 72. Gráficos de superfície.

Os gráficos de superfície dividem-se nos seguintes subtipos:

Superfícies em 3D: os gráficos de superfície 3-D, Figura 73, mostram as tendências em valores ao longo de duas dimensões numa curva contínua. As faixas a cores num gráfico de superfície não representam as séries de dados mas sim a distinção entre os valores. Este gráfico é normalmente utilizado para mostrar relações entre grandes quantidades de dados que de outro modo seria difícil de ver [38].

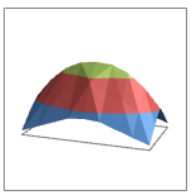

Figura 73. Superfícies em 3D.

Superfície de esboço em 3D: quando apresentado sem cor na superfície, um gráfico de superfície 3-D, Figura 74, é denominado gráfico de superfície vetorial 3-D. Este gráfico mostra apenas as linhas. Um gráfico de superfície 3-D apresentado sem faixas a cores em qualquer superfície é denominado gráfico de superfície vetorial 3-D. Este gráfico mostra apenas as linhas [38].

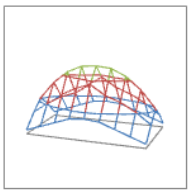

Figura 74. Superfície de esboço em 3D.

Níveis: os gráficos de níveis são gráficos de superfície vistos de cima, semelhantes a mapas topográficos 2-D. Num gráfico de nível, as faixas de cores representam intervalos de valores específicos. As linhas num gráfico de níveis ligam pontos interpolados de igual valor [38].

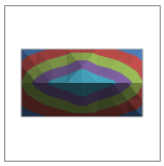

Figura 75. Gráficos de Níveis.

1.8 Gráficos em anel

Como no gráfico circular, um gráfico em anel, Figura 76, mostra a relação das partes com o todo, mas pode conter mais que uma série de dados.

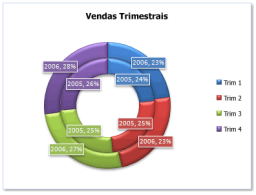

Figura 76. Gráficos em anel.

Os gráficos em anel dividem-se nos seguintes subtipos:

Anéis Destacados: à semelhança do gráfico circular destacado, o gráfico em anel destacado, Figura 77, mostra a contribuição de cada valor para um total, realçando os valores individuais, mas pode conter mais do que uma série de dados [38].

Figura 77. Gráficos de Anéis Destacados.

1.9 Gráficos de radar

Os gráficos de radar, Figura 78, comparam os valores agregados de várias séries de dados.

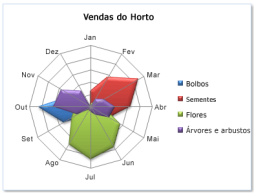

Figura 78. Gráficos de Radar.

Os gráficos de radar dividem-se nos seguintes subtipos:

Radar e radar com marcadores: com ou sem marcadores para pontos de dados individuais, Figura 79, o gráfico de radar mostra as alterações dos valores relativamente a um ponto central [38].

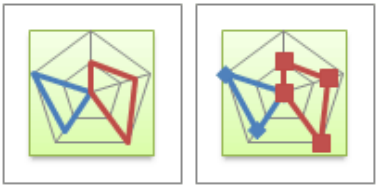

Figura 79. Gráfico de radar e radar com marcadores.

Radar preenchido: num gráfico de radar preenchido, a área coberta por uma série de dados é preenchida por uma cor [38].

Figura 80. Gráfico de Radar preenchido.

1.10 Gráfico de bolhas

Um gráfico de bolhas, Figura 81, exibe um conjunto de valores numéricos como círculos. É especialmente útil para conjuntos de dados com dezenas a centenas de valores, ou com valores que diferem em várias ordens de magnitude [39].

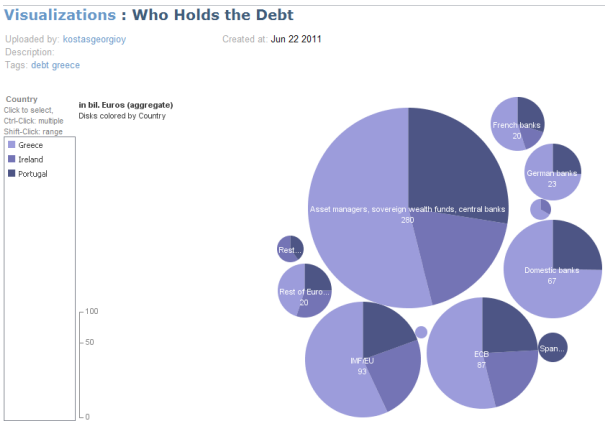

Figura 81. Gráfico de bolhas.

Os círculos em um gráfico de bolhas representam valores de diferentes dados, em que a área de um círculo correspondente ao valor. As posições das bolhas não significam nada.

1.11 Matriz de gráficos

Uma matriz de gráficos, Figura 82, resume um conjunto de dados multidimensional em uma matriz. As linhas representam os valores em uma coluna de texto, por exemplo: linha de produtos, e as colunas representam uma outra coluna de texto, por exemplo: país. Cada célula da matriz mostra um círculo ou uma barra que representa o valor da combinação linha/coluna. As barras são uteis quando se pretende realizar comparações exatas e estas permitem ter mais espaço para mais colunas. Por sua vez, os círculos, são bons para mostra valores não negativos que variam muito [39].

Figura 82. Matriz de gráficos.

Anexo II - Padrões de visualização

Este anexo aprofunda cada um dos padrões de visualização em encontrados no contexto do trabalho realizado no artigo *Toward design patterns for dynamic analytical data visualization* [44].

2.1 Padrões de dados

Existem dois tipos de padrões de dados: o padrão decorador e o padrão de codificação visual [44].

- Padrão *Decorated Data* [44]:
  - **Contexto:** necessidade de lidar com vários elementos nas visualizações dinâmicas e analíticas.
  - **Problema:** como organizar e processar os vários elementos de forma eficaz.
  - **Forças:** os elementos podem incluir dados em bruto, meta-dados, relacionamentos, dados derivados e ate mesmo codificações visuais. Diferentes tipos de fontes de dados podem ser utilizados.
  - **Solução:** separação dos elementos relacionados em uma parte principal e em outra parte de "decoração", para que as diferentes partes possam ser criadas e manipuladas dinamicamente.
  - **Exemplo:** a Figura 83 apresenta uma vista de um micro vector unidirecional de uma aplicação. Um mapa de calor, etiquetas, uma árvore e componentes dos eixos são utilizados nesta visualização, juntamente com os três modelos de dados. Estes modelos são usados para os dados em bruto, os dados hierárquicos e os dados resultantes da união destes dois. Qualquer alteração num dos modelos desencadeia alterações nos modelos relacionados.

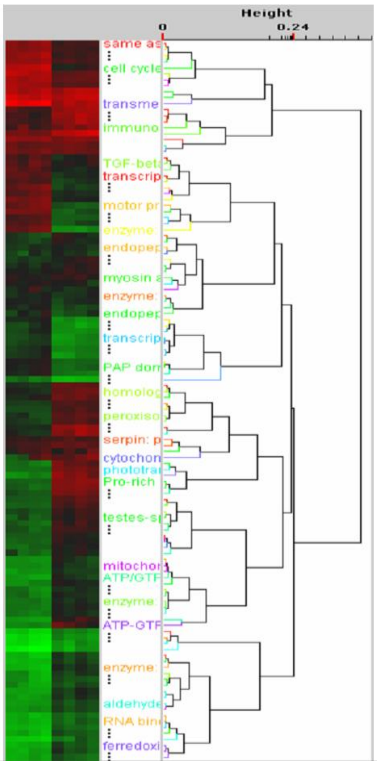

Figura 83. Vista do micro vector.

- Padrão *Visual Encoding* [44]:
  - **Contexto:** visualização de dados abstratos.
  - **Problema:** como converter os dados abstratos em algo apresentável.
  - **Forças:** a informação a ser exibida nas visualizações pode incluir dados e meta-dados que são derivados ou calculados e também pode conter informações sobre os estados (seleções, exclusões, etiquetas, etc.).
  - **Solução:** codificação das informações visuais tais como: posição, cor, forma, tamanho, orientação, saturação, textura, entre outras.
  - **Exemplo:** a Figura 84 mostra o uso da codificação num gráfico de dispersão. Os tipos inteiros são utilizados para codifica as cores e as formas. Por sua vez, os tipos booleanos são utilizados para os estados de seleção.

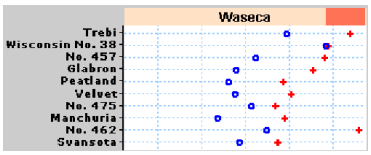

Figura 84. Codificação num gráfico de dispersão.

2.2 Padrões estruturais

- Padrão *Graphic Grid* [44]:
  - **Contexto:** existe a necessidade de ter diferentes gráficos que permitam visualizar diferentes aspetos dos dados numa área limitada.
  - **Problema:** como organizar os gráficos no espaço disponível a fim de a visualização seja fácil de usar e entender.
  - **Forças:** Frequentemente os gráficos são utilizados para explorar dados e padrões.
  - **Solução:** organizar os gráficos e outros componentes visuais em um *layout* cujo aspeto seja parecido com o de uma tabela, a fim de facilitar a descoberta de padrões e tendências nos dados.
  - **Exemplo:** a Figura 85 permite visualizar vários gráficos no mesmo espaço.

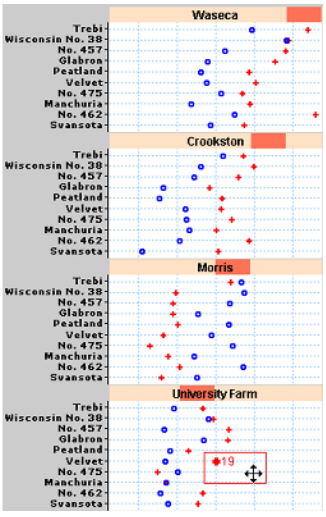

Figura 85. Exemplo da utilização do padrão *Graphic Grid*.

- Padrão *Overlay* [44]:
  - **Contexto:** existem um conjunto de gráficos relativamente simples especializados para executarem tarefas específicas. É necessário fornecer um gráfico para apoiar várias destas tarefas.
  - **Problema:** Como construir um novo gráfico complexo a partir dos gráficos existentes.
  - **Forças:** estes gráficos simples partilham parcial ou totalmente a mês área de desenho. Também estes gráficos parecem estar no mesmo sistema de coordenadas. Se necessário, os gráficos podem ter comportamentos interativos, tais como: *zoom*, seleção entre outros.
  - **Solução:** sobreposição das diferentes camadas dos gráficos simples para construir um gráfico complexo.
  - **Exemplo:** a Figura 86 exibe um exemplo da utilização deste padrão de visualização.

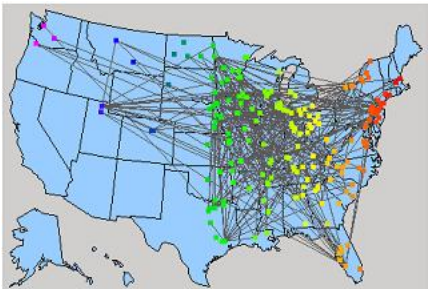

Figura 86. Exemplo da utilização do padrão *Overlay*.

2.3 Padrões comportamentais

- Padrão *Linked Graphs* [44]:
  - **Contexto:** utilização de vários gráficos para explorar dados de forma dinâmica.
  - **Problema:** como visualizar os dados e associações visuais entre os diferentes gráficos dinâmicos.
  - **Forças:** as associações de dados podem ser valores relacionados, estruturas relacionadas ou estados relacionados.
  - **Solução:** ligar os gráficos através de modelos compartilhados. Os modelos capturam os dados e/ou estados visuais e associações. Os gráficos interpretam o conteúdo dos mesmos modelos de forma consistente.
  - **Exemplo:** a Figura 86 exibe um exemplo da utilização deste padrão de visualização.

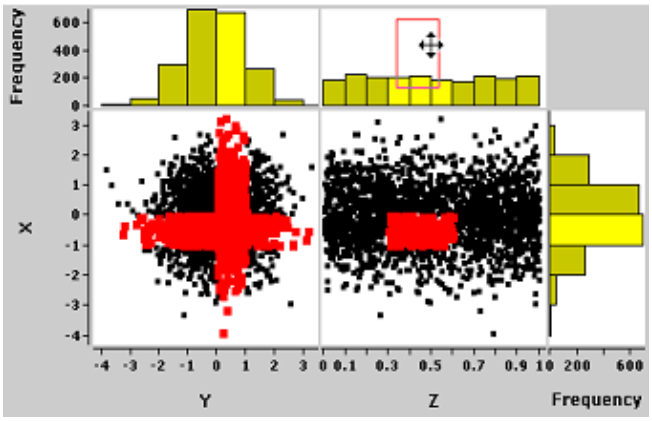

Figura 87. Exemplo da utilização do padrão *Linked Graphs* e *Brushing*.

- Padrão *Brushing* [44]:
  - **Contexto:** manipular um de elementos de dados em um gráfico:
  - **Problema:** como identificar e aplicar diferentes operações em um grupo de elementos usando por exemplo um rato.
  - **Forças:** as operações possíveis de realizar podem ser: apagar, destacar, entre outras. A atualização rápida e imperativa. A seleção é a base para todas estas operações.
  - **Solução:** utilização de um objeto geométrico no gráfico, comumente chamado de pincel ou *brush*. O pincel pode ser arrastado ou rodado através de um dispositivo de entrada como por exemplo o rato. Os elementos que são tocados pelo pincel são candidatos para a seleção e outras operações.
  - **Exemplo:** a Figura 86 também ilustra um exemplo da utilização deste padrão de visualização.

- Padrão *Details Management* [44]:
  - **Contexto:** análise visual exploratória de um conjunto grande e complexo de dados.
  - **Problema:** como descobrir relações e padrões escondidos em grandes conjuntos de dados.
  - **Forças:** geralmente não é viável ou razoável esperar que um gráfico simples possa ser uma demonstração efetiva dos dados quando estes possuem uma grande dimensão e/ou são multidimensionais.
  - **Solução:** organizar os dados em diferentes níveis de detalhe, com base nos dados ou técnicas de processamento visual tais como: extração, exclusão, sumarização, distorção, entre outras. Os detalhes dos diferentes níveis são mostrados quando estes são necessários, quer seja num único gráfico ou em múltiplos gráficos.
  - **Exemplo:** a Figura 86 apresenta um exemplo da utilização deste padrão de visualização.

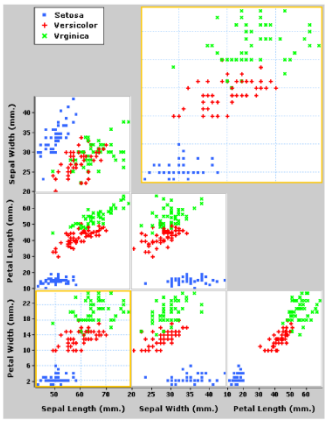

Figura 88. Exemplo do padrão *Details Management*.

- Padrão *Network Flow* [44]:
  - **Contexto:** necessidade de realizar uma tarefa complexa que envolve várias operações.
  - **Problema:** como organizar e monitorizar uma tarefa visual.
  - **Forças:** as operações envolvidas podem incluir diferentes análises, dados e processamento visual.
  - **Solução:** divisão da tarefa em várias sub tarefas (cada uma executa uma funcionalidade diferente) que são modeladas como nós. Os nós são ligados a fim de formar um grafo direcionado. A informação é processada em vários fluxos de nó em nó através das ligações (*links*).
  - **Exemplo:** a Figura 86 exibe um exemplo da utilização deste padrão de visualização.

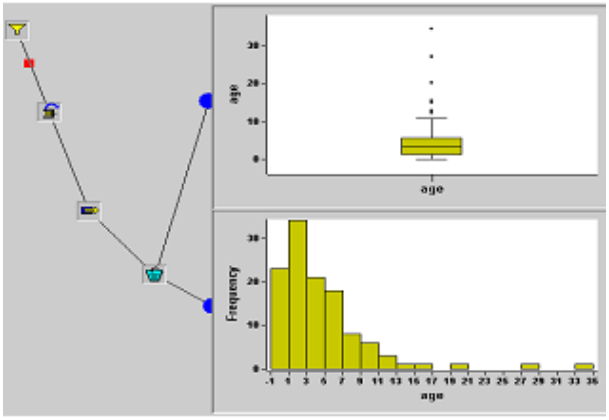

Figura 89. Exemplo do padrão Network Flow.

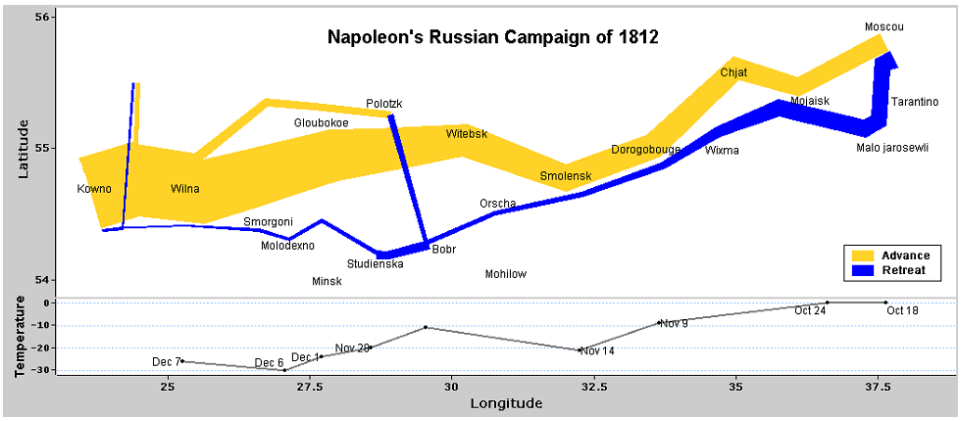

Figura 90. Exemplo do Padrão *Progressive Refinement*.

- Padrão *Progressive Refinement* [44]:
  - **Contexto:** o *display* não é exatamente o que se pretende. Pode ser muito simples, ou existem necessidades adicionais ou diferentes preferências.
  - **Problema:** como utilizar o display de forma a satisfazer as diferentes necessidades.
  - **Forças:** devido a complexidade de desenvolvimento e ao uso de diverso gráficos analíticos, é muito difícil, senão impossível, a criação de uma coleção de gráficos que satisfaz todas as necessidades.
  - **Solução:** iniciar com uma visualização simples que progressivamente seja refinada com os elementos que se pretende.
  - **Exemplo:** na Figura 90 é possível observar um exemplo da utilização deste padrão de visualização. Nesta figura observa-se uma reprodução parcial do famoso gráfico *Minard Chart*.

Anexo III - Glossário do Model Driven Engineering (MDE)

Este anexo pretende fornecer um glossário com os termos mais utilizados no MDE.

3.1 Modelo

O termo "modelo" deriva do latim *modulus*, que significa medida, regra, padrão, exemplo a ser seguido. A definição formal do modelo pode ser: Qualquer pessoa que utiliza um sistema A que não está nem direta nem indiretamente, interagindo com um sistema B, para obter informações sobre o sistema B, está a usar A como um modelo para B. Esta definição é bastante genérica. O conceito modelo pode ser descrito de forma mais precisa, apresentando três critérios para um modelo. Stachowiak [62] descreve que um modelo precisa atender a três critérios essenciais, caso contrário não se trata de um modelo:

- **Mapeamento:** um modelo é baseado em um original. O (sistema) original pode ser algo ainda a ser construído ou pode permanecer completamente imaginário.
- **Redução:** nem todas as propriedades do sujeito são mapeados para o modelo, mas o modelo é de certa forma reduzida. No entanto, um modelo deve refletir, pelo menos, algumas das propriedades do assunto.
- **Pragmática:** um modelo precisa ser útil no lugar de um sujeito com relação a alguma finalidade.

3.2 Transformação de modelos

Transformar um modelo em outro modelo significa que um modelo fonte é transformado em um modelo destino com base em algumas regras de transformação. Diferentes métodos podem ser utilizados para definir as regras de transformação. Em 2007, a OMG deu a conhecer o QVT [94], um modelo de especificação de linguagem de transformação. É possível fazer uma distinção entre as transformações (ou refinamentos) adicionando informações computacionais e as transformações agregando tecnologia (ou plataforma) de informação. O primeiro caso requere de intervenção humana, o último pode ser feito automaticamente, se for alimentada a transformação com a informação da plataforma [95].

3.3 *Query / View / Transformation* (QVT)

A transformação do modelo é uma componente crítica do MDA. Em 2007, a especificação final do *Meta Object Facility (MOF) 2.0 Query / View / Transformation* foi realizada. Esta especificação, mais conhecida como QVT define meta-modelos para definir linguagens de transformação de modelos. As linguagens resultantes podem transformar os modelos de origem em modelos de destino onde os modelos de origem e de destino podem obedecer a um meta-modelo MOF arbitrário. A linguagem de transformação em si também é um modelo e ajusta-se também a um meta-modelo MOF [95].

3.4 *MetaObject Facility* (MOF)

O MetaObject Facility (MOF) é um *standard* do meta-modelo da OMG para a criação dos modelos que são utilizados no MDE. O MOF foi inicialmente utilizado para os modelos UML e posteriormente foi reutilizado pela comunidade de engenharia de *software* como base para a criação de novas linguagens baseadas no UML, estendendo o MOF para as necessidades pessoais dos desenvolvedores. Utilizando a representação do megamodelo, Figura 91, é possível observar as relações entre o MOF e os modelos UML [60].

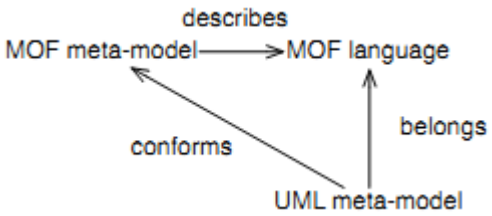

Figura 91. Relações entre o MOF e o UML.

O meta-modelo MOF definido pela OMG descreve a linguagem MOF. O meta-modelo do UML pertence a linguagem MOF e está de acordo com o meta-modelo MOF. Todas as linguagens da família do UML incluem no seu núcleo o MOF, Figura 92 [60].

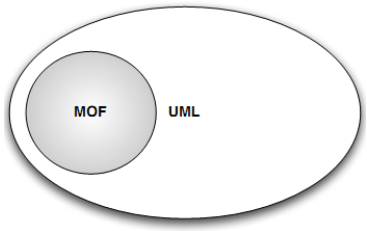

Figura 92. Núcleo do MOF.

3.5 *XML Metadata Interchange* (XMI)

O *XML Metadata Interchange* (XMI) é um padrão definido pela OMG para a troca de informações de meta-dados utilizando o *Extensible Markup Language* (XML). O XMI pode ser utilizado para qualquer meta-dado cujo meta-modelo possa ser expresso utilizando o MOF. O XMI é comumente utilizado como um formato de intercâmbio de modelos UML, embora também seja possível utilizá-lo para a serialização de outras linguagens. Isto permite que seja possível a utilização do XMI em várias ferramentas de modelação [60].

3.6 Geração de código

A geração de código, como o próprio nome indica, consiste na geração do código fonte (em uma linguagem de programação como Java [96] ou C# [9]) a partir de um modelo. Isso pode ser feito de diversas formas, por exemplo, utilizando modelos e regras de reescrita ou pela construção de

um meta-modelo de uma linguagem de programação e definição de transformações formais a partir do meta-modelo do modelo para o meta-modelo da linguagem de programação [95].

3.7 *Domain Specific Languages* (DSL's)

As DSL's são utilizadas para aumentar a flexibilidade, qualidade e entrega dos sistemas de *software*, aproveitando as propriedades específicas do domínio de uma aplicação particular. Como é possível ter abordagens genéricas e especificas para resolver um problema, as DSL's procuram alcançar uma solução que seja ótima para um problema específico [74], [65].

3.8 Domain-Specific Visual Language

É um DSL com uma sintaxe gráfica concreta [95].

3.9 Sintaxe abstrata

A sintaxe abstrata define os conceitos de uma linguagem e suas relações [95].

3.10 Sintaxe concreta

A sintaxe concreta define a aparência física da linguagem. Para as linguagens textuais, isto significa que a sintaxe concreta define como são formadas as frases. Para uma linguagem gráfica, isto significa que a sintaxe concreta define a aparência gráfica dos conceitos da linguagem e como eles podem ser combinados em um modelo [95].

3.11 Semântica

A semântica descreve o significado de uma frase ou modelo especificado em alguma linguagem. No contexto do MDE, isto significa que a semântica de um modelo descrever quais são os efeitos da execução daquele modelo [95].

3.12 Meta-modelo

Um modelo é uma abstração de um sistema ou parte deste. Um meta-modelo é novamente outra abstração, destacando as propriedades do modelo em si. A relação entre um modelo e o seu meta-modelo é a mesma que existe entre um programa e a gramática da linguagem de programação em que este está escrito [60]. Portanto, um meta-modelo é um modelo que descreve uma linguagem de modelação e como tal descreve os aspetos estruturais e semânticos dessa linguagem [11]. Um meta-modelo, também precisa de ser expresso em alguma linguagem. Uma linguagem possível, com esta finalidade é MOF [95].

3.13 *Domain-Specific Model* (DSM)

Os DSM são utilizados para se referirem a um modelo de um domínio específico, ou seja, um modelo especificado usando uma conexão DSL. No entanto, o uso mais comum do DSM é para referir-se a modelação específica de domínio. De acordo com o DSM Fórum o DSM é definido como [95]:

> "O Domain-Specific Model eleva o nível de abstração além da programação, especificando a solução diretamente com os conceitos do domínio. Os produtos finais são gerados a partir dessas especificações de alto nível. Esta automatização é possível porque a linguagem e os geradores precisam atender às exigências de uma única empresa e domínio".

3.14 Multi-modeling

Na literatura, o *Multi-modeling* é definida como o ato de combinar diversos modelos, o que pode ser feito de duas maneiras [95]:

- ***Hierarchical multi-modeling*:** composições de distintos estilos de modelagem hierárquica são combinadas para tirar vantagem das capacidades únicas e expressividade dos estilos de modelagem distintas.
- ***Multi-view modeling*:** modelos distintos e separados do mesmo sistema são construídos para modelar diferentes aspetos do sistema.

Anexo IV - OCL

Este anexo apresenta o OCL dando exemplos e seguindo um pequeno tutorial que foi elaborado no decorrer da dissertação e o qual foi adaptado para ser utilizado com a ferramenta que foi desenvolvida no âmbito da dissertação.

Antes de começar com os exemplos ilustrativos é necessário, em primeiro lugar, conhecer a linguagem a fim de poder perceber o que está a acontecer nos exemplos.

No OCL existem três tipos de restrições [11]:

- **Invariantes:** são restrições que se verificam sempre e cuja palavra reservada é "inv".
- **Pré-condições:** são condições que se verificam a quando da execução de uma dada operação. A palavra reservada é "pre".
- **Pós-condições:** são condições que se verificam após a execução de uma dada operação e cuja palavra reservada é "post".

O OCL é uma linguagem tipificada, possuindo quatro tipos primitivos, nomeadamente:

- **Integer:** número inteiro de qualquer tamanho, por exemplo: 123.
- **Real:** número real de qualquer tamanho, por exemplo: 1,23.
- **String:** conjunto de caracteres, por exemplo, 'UMa'.
- **Boolean:** true ou false.

Para além dos quatro tipos primitivos, o OCL possui um conjunto de operadores, cada conjunto opera sobre um determinado tipo de dados primitivos. A Tabela 6 apresenta os operadores sobre inteiros e reais, na Tabela 7 mostra os operadores sobre *Strings* e na Tabela 8 os operadores sobre booleanos.

Tabela 6. Operadores sobre Inteiros e Reais.

Operador	Notação	Tipo de resultado
Equals	a = b	Boolean
Not equals	a <> b	Boolean
Less	a < b	Boolean
More	a > b	Boolean
Less or equal	a <= b	Boolean
More or equal	a >= b	Boolean
Plus	a + b	Integer /Real
Minus	a - b	Integer /Real
Multiply	a * b	Integer /Real
Divide	a / b	Real
Modulus	a.mod(b)	Integer
Integer division	a.div(b)	Integer
Absolute value	a.abs	Integer/Real
Maximum	a.max(b)	Integer/Real
Minimum	a.min(b)	Integer/Real
Round	a.round	Integer
Floor	a.floor	Integer

Tabela 7. Operações sobre Strings

Operador	Notação	Tipo de resultado
Concatenation	s.concat(String)	String
Size	s.size	Integer
To lower case	s.toLower	String
To upper case	s.toUpper	String
Substring	s.substring(int, int)	String
Equals	s1 = s2	Boolean
Not equals	s1 <> s2	Boolean

Tabela 8. Operações sobre booleanos.

Operador	Notação	Tipo de resultado
Concatenation	a or b	Boolean
Size	a and b	Boolean
To lower case	a xor b	Boolean
To upper case	not a	Boolean
Substring	a = b	Boolean
Equals	a <> b	Boolean
Not equals	a implies b	Boolean
If then else	if a then b1 else b2 endif	Type of b

A linguagem OCL também possui quatro tipos de coleções, nomeadamente:

- **Set:** é uma coleção onde um elemento só pode ocorrer uma única vez e não existe uma ordem específica.
- **OrderSet:** é uma coleção onde um elemento só pode ocorrer uma única e existe uma ordem específica.
- **Bag:** é uma coleção onde um elemento pode ocorrer mais do que uma vez e não existe uma ordem específica.
- **Sequence:** é uma coleção onde um elemento pode ocorrer mais do que uma vez e existe uma ordem especifica.

Na Tabela 9 estão descritos os métodos existentes nas coleções anteriormente descritas.

Tabela 9. Descrição dos métodos existentes nas coleções.

Operação	Descrição
size	Número de elementos na coleção.
count(object)	Número de vezes que o objeto ocorre na coleção.
includes(object)	Retorna True se o objeto for um elemento da coleção.
includesAll(collection)	Retorna True se todos os elementos desta coleção pertencerem à coleção inicial.
isEmpty	Retorna True se a coleção estiver vazia.
notEmpty	Retorna True se a coleção contiver algum elemento.

iterate(expression)	A expressão é avaliada para todos os elementos da coleção.
sum(collection)	Somatório de todos os elementos da coleção.
exists(expression)	Retorna True se a expressão for True para pelo menos um elemento.
forAll(expression)	Retorna True se a expressão for valida para todos os elementos da coleção.
select(expression)	Retorna o subconjunto de elementos que satisfazem a expressão.
reject(expression)	Retorna o subconjunto de elementos que não satisfazem a expressão.
collect(expression)	Forma uma nova coleção com os elementos retornados pela expressão.
one(expression)	Retorna True se um e apenas um elemento da coleção satisfazer a expressão.
sortedBy(expression)	Retorna uma sequência com todos os elementos da coleção na ordem especificada (a expressão tem que possuir o operador <).

Tutorial do OCL

4.1 Requisitos

A fim de poder realizar este tutorial é necessário os seguintes conhecimentos/ferramentas:

- Conhecimentos mínimos em OCL.
- Aplicação "Meta Visualizer".
- O modelo da superestrutura do UML, nível 2 versão 2.1.2

4.2 Projeto

O projeto que irá ser utilizado para exemplificar as consultas em OCL irá ser construído utilizado a aplicação "Meta Visualizer". Nesta aplicação é possível encontrar diversas componentes no painel "Tools". Destas componentes apenas é preciso duas em particular, a componente "XMI" para carregar o modelo pretendido e a componente "OCL" para realizar as consultas em OCL. A fim de ter este projeto é preciso seguir os seguintes passos:

1. Criar um novo projeto. Para tal no menu "File->New Project" irá aparecer uma pequena janela, onde é introduzido o nome do projeto e a directoria onde o projeto irá ser guardado. A título de exemplo, o nome do projeto pode ser "ExemploOCL" e recomenda-se guardar o projeto no ambiente de trabalho.
2. De seguida é necessário verificar se no painel "Tools" encontram-se todas as componentes necessárias. São necessárias as componentes "XMI" e "OCL". Caso estas componentes não se encontram disponíveis é necessário fazer uma procurar até a pasta onde se encontram as componentes do projeto. Depois de verificar que as componentes estão disponíveis, é preciso arrastar estas para a área de desenho do separador

"Visualization Designer". No caso da componente "XMI" é necessário especificar o diretório onde se encontra o modelo sobre o qual se pretende realizar as consultas.

3. Logo a seguir são realizadas as ligações dos respetivos portos das duas componentes como mostra a Figura 93.

Figura 93. Ligação dos portos.

Portanto, com este conjunto de passos já é possível realizar as consultas deste tutorial.

4.3 Consultas em OCL

O OCL possui um conjunto de palavras reservadas que podem ser consultados no documento *OCL 2.0 Quick Reference* no link: http://www.info2.uqam.ca/~chieze_e/INF3140/ocl-ref-short.pdf.

A fim de saber como irão ser realizar as consultas, também é necessário conhecer o meta-modelo do modelo que está-se a utilizar. A componente "XMI" permite consultar o meta-modelo.

As consultas aqui apresentadas irão focar-se nas classes do modelo que estão sendo analisadas já que é interessante saber um conjunto de informações quando é alterado o modelo, como por exemplo: quais são as classes existentes, quantas classes abstratas existem, quais as operações de uma classe, etc.

4.4 Obtenção de informação – Nomes

Nesta secção irão ser realizadas um conjunto consultas com o intuito de obter informação sobre os nomes das classes. Suponha-se então que se pretende saber todos os nomes das classes que existem no meta-modelo. Para tal escreve-se na componente "OCL" a seguinte consulta:

```
Class.allInstances()->collect( c | c.name)
```

Isto retornará uma lista contendo todos os nomes das classes presentes no modelo.

```
Results:
{Comment, DirectedRelationship, LiteralSpecification, LiteralInteger,
LiteralString, LiteralBoolean, LiteralNull, Constraint, ElementImport,
TypedElement, …}
```

Fazendo uma pequena análise, é possível observar que foram utilizadas as palavras reservadas "allInstances" e "collect". Em primeiro lugar a palavra reservada "allInstances" permite aceder a todas a instâncias de um determinado elemento do meta-modelo, neste caso a todas as classes que fazem parte do modelo. Em segundo lugar, a palavra reservada "collect" permite criar uma coleção, neste casso de todos os nomes das classes existentes, devolvendo como resultado 205 classes.

Mas como é que é possível saber que era necessário utilizar a palavra "Class" para aceder as classes, visto que no modelo não existe essa palavra. Ora é necessário consultar o meta-modelo visto que este é quem descreve a linguagem de modelação que se encontra no modelo. Lá consegue-se encontrar cada elemento que faz parte do modelo e como estes estão relacionados entre si. No entanto, se for consultado o meta-modelo, o elemento "Class" não possui nenhuma propriedade "name" que permita obter o nome da classe, mas uma classe é um elemento e portanto possui um "NamedElement".

Portanto, sabendo que tipos de consultas se pretende e tendo o meta-modelo por perto é possível sempre arranjar uma ou mais consultas que permitam obter aquilo que se pretende.

Agora suponha-se que se pretende saber o número de classes que existe no modelo. Para tal escreve-se na consola a seguinte consulta OCL:

```
Class.allInstances()->collect( c | c.name)->size()
```

E obtido como resultado: 205

Como é possível observar trata-se da mesma consulta que foi utilizada para o exemplo anterior com a diferença de que foi acrescentado a operação "size()".

No próximo exemplo pretende-se saber quais são os nomes das superclasses, das classes abstratas, e das classes "normais" que se encontram no modelo. Para tal escreve-se na consola as seguintes consultas OCL:

```
--superClass
Class.allInstances()->collect( c | c.oclAsType(Class).superClass )->flatten()
--abstract
Class.allInstances()->select( c | c.oclAsType(Class).isAbstract = true)->collect( c | c.name)
--class
```

```
Class.allInstances()->select( c | c.oclAsType(Class).isAbstract = false)->collect( c | c.name)
```

Consegue-se observar que surgiram novas palavras nas nossas consultas, como por exemplo a palavra reservada "select" e "flatten()". A primeira permite selecionar um ou mais elementos que satisfazem uma determinada condição e a segunda permite juntar todas as listas que fazem parte do resultado numa única lista. De salientar que foi necessário consultar o meta-modelo a fim de saber como obter as superclasses, as classes abstratas e as classes que não são abstratas.

4.5 Obtenção de informação – atributos

Nos exemplos acima descritos foram consultados os nomes das classes. Nos exemplos seguintes irá ser focado a obtenção de informação relacionada com os atributos.

Suponha-se que se pretende saber todos os atributos de todas as classes. Para tal escreve-se na consola a seguinte consulta OCL:

```
Class.allInstances()->collect(c | c.oclAsType(Class).ownedAttribute)
```

Como se pode constatar, foi utilizada a palavra "ownedAttribute" que faz parte do meta-modelo para obter todos os atributos de todas as classes. De salientar que o resultado retornado por esta consulta é uma coleção de coleções, sendo que se o utilizador pretender que seja uma só coleção, será necessário utilizar a palavra reservada "flatten" no fim da consulta.

Nota: O leitor já poderá estar consciente de que este é um processo que se vai repetindo e que com poucas modificações é possível reutilizar a consulta em outras situações. Também é de salientar, que o leitor mais experiente em OCL pode chegar ao mesmo resultado através da formulação de outra consulta mais ou menos complexa.

Agora suponha-se que se deseja saber os atributos existentes numa determina classe. Para o exemplo irá ser utilizada a classe "`DataType`". Portanto escreve-se na consola a seguinte consulta OCL:

```
Class.allInstances() -> select( c | c.name = 'DataType') -> collect( c | c.oclAsType(Class).ownedAttribute )->flatten()
```

Esta consulta retorna o seguinte resultado:

```
{ownedAttribute, ownedOperation}
```

No próximo exemplo irá ser aumentado o grau de complexidade da consulta. Suponha-se então que pretende-se saber quais são a(s) classe(s) que contêm um determinado atributo. Para o

exemplo irá utilizado como atributo teste o atributo “name”. Portanto escreve-se na consola a seguinte consulta OCL:

```
Class.allInstances()
    ->iterate(
        c:Class;
        cv:Set(Class) = Set{}
        | if (c.oclAsType(Class).ownedAttribute->exists(name
= 'name'))
             then cv->including(c) else cv endif
    )
```

E o resultado deveria ser o seguinte:

```
{Tag, NamedElement}
```

É possível observar que a consulta já é um bocado complexa. Basicamente é necessário percorrer todas as classes a procura das classes que contêm o atributo pretendido. Estas são então adicionadas a um conjunto e é devolvido esse mesmo conjunto. No entanto a consulta não pode ser executada na componente “OCL” devido a problemas relacionados com as condições. Espera-se que muito rapidamente seja resolvido este problema.

4.6 Obtenção de informação – Operações

Nos exemplos acima descritos foram consultados os atributos das classes. Nos exemplos seguintes as consultas irão focar-se na obtenção de informação relacionada com as operações das classes.

Suponha-se que pretende-se saber todas as operações que existem no modelo. Para tal escreve-se na consola a seguinte consulta OCL:

```
Class.allInstances()->collect( c | c.oclAsType(Class).ownedOperation)
```

De salientar que o resultado da consulta anterior devolve uma coleção de coleções, sendo que em alguns casos estas coleções estão vazias o que significa que a classe não possui operações.

Também é possível realizar a consulta da seguinte forma:

```
Operation.allInstances()->collect( o | o.name)
```

Assim, obtém-se todas as operações existentes numa única coleção.

Agora suponha-se que pretende-se saber quais são as operações numa classe, por exemplo a classe “DataType”. Portanto escreve-se na consola a seguinte consulta OCL:

```
Class.allInstances()->select( c | c.name = 'DataType')->collect( c |
c.oclAsType(Class).ownedOperation)->flatten()
```

E o resultado é o seguinte:

```
{inherit}
```

Suponha-se que pretende-se saber quais eram as classes que possuem operações. Para tal escreve-se na consola a seguinte consulta OCL:

```
Operation.allInstances()->collect( o | o.oclAsType(Operation).class)
```

Sabe-se que muitas operações têm parâmetros. Suponha-se então que pretende-se saber quais são os parâmetros de todas as operações. Para tal escreve-se na consola a seguinte consulta OCL:

```
Operation.allInstances()->collect( o |
o.oclAsType(Operation).ownedParameter)
```

Ou também se pode fazer da seguinte forma:

```
Parameter.allInstances()->collect( p | p.name)
```

Como é possível observar, para o mesmo problema, consegue-se ter duas consultas que permitem chegar ao mesmo resultado.

4.7 Conclusão

Portanto como, com a ajuda do OCL é relativamente fácil extrair informação. No entanto, a medida que a complexidade daquilo que pretendesse vai aumentando, também aumenta a complexidade da consulta em OCL, mas nem sempre é necessário que aquilo que se pretende seja complexo para que a consulta seja complexa. Existem muitos casos em que se quer saber algo simples de um modelo e é necessário de um "comboio" de OCL. Num futuro próximo, pretendesse substituir o núcleo do OCL atual pelo núcleo do *Imperative OCL* que irá permitir simplificar estas consultas e resolver problemas que com o atual núcleo de OCL não se conseguem resolver.

Anexo V - Estudo das bibliotecas gráficas

Este anexo apresenta uma análise que foi realizada às bibliotecas gráficas a fim de encontrar aquela que se adequa ao projeto.

5.1 Requisitos das bibliotecas

Basicamente os requisitos que irão ser tidos em contas na análise das bibliotecas são os seguintes:

- Suporte para diferentes tipos de gráficos em 2D e 3D.
- Suporte para interação nos gráficos e a aparência dos mesmos.
- Performance.
- Facilidades de *Binding*.
- Preço e descontos.
- Licenciamento.
- Controlos adicionais incluídos na biblioteca.

5.2 Bibliotecas analisadas

Como em qualquer projeto de *software* existe sempre uma decisão que deve ser tomada: comprar ou produzir os componentes que são precisas para a nossa aplicação. É neste sentido que foi realizada uma análise das soluções que se encontram disponíveis no mercado com base nos requisitos descritos anteriormente. Estas soluções são:

- *Telerik*
- *ComponentArt*
- *Infragistics*
- Visifire

5.2.1 *Telerik*

Esta empresa oferece um conjunto de ferramentas muito completo para a construção de aplicações que utilizam o WPF, através da biblioteca *RadControls*. Os controlos desta biblioteca oferecem uma alta performance e um desenho atraente e altamente personalizável o que permite melhorar a experiencia do utilizador.

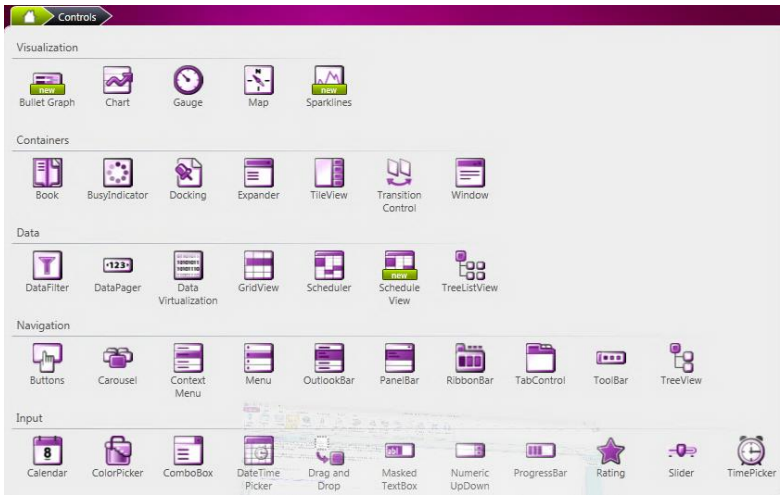

Figura 94. Controlos oferecidos pela Biblioteca.

Em termos de controlos, esta biblioteca oferece um conjunto de controlos muito completo. A Figura 94 apresenta quais são os controlos principais da ferramenta:

5.2.1.1 Primeira Análise

Nesta análise pretende-se verificar quais são as características que o vendedor coloca no sítio da internet e compará-las com os requisitos que estão sendo procurados para a ferramenta.

Em primeiro lugar, a nível de visualização de dados, esta biblioteca possui cinco controlos, nomeadamente: *Bullet Graph, Chart, Gauge, Map e Sparklines*.

Nota: a análise realizada irá só forcar a parte dos gráficos.

O controlo *Chart* é adequado para a visualização de dados através de gráficos de forma a permitir uma análise mais detalhada. O controlo fornece um desenho atraente e suporte completo para o *Expression Blend* o que permite que o desenvolvedor possa melhorar a aparência do controlo. Também o controlo possui diferentes *layouts* pré-definidos que ajuda a fornecer uma aparência consistente na aplicação.

Em termos de características principais deste controlo, este:

- Fornece 24 tipos de gráficos diferentes em 2D, entre os quais: Bar, Linha, Área, Bolha, entre outros.
- Fornece 18 tipos de gráficos diferentes em 3D, entre os quais: Bar, Linha, Espaço, Rosca, entre outros. Para além disso, é possível rodar em 360 graus os gráficos no eixo do X.
- Fornece suporte a *DataBinding*.
- Automatização de *DataBinding* para as coleções de coleções.

- Virtualização de dados e amostra de dados, o que permite gerir milhões de dados, que sejam representados por pontos, em questão de milissegundos. O controlo divide os dados em grupos e cada grupo utiliza uma das muitas funcionalidades de amostragem. O resultado final é um gráfico que resume os dados reais perto o suficiente para uma execução rápida.
- Fornece *Scroll* e *zoom*.
- Existe suporte a *Tooltip*, ou seja, a capacidade de adicionar etiquetas aos gráficos de forma a melhorar a visualização.
- Fornece funcionalidades avançadas para o eixo horizontal de forma a obter um eixo mais inteligente e melhorar a forma como o eixo manipula os dados dos utilizadores. Entre as funcionalidades se podem mencionar as seguintes: definição da distância entre os rótulos, ângulo de rotação dos rótulos e intervalo de valores.
- Agrupamento e agregação
- Compatibilidade de código WPF / Silverlight
- Permite salvar imagens.
- Dá suporte a múltiplos eixos verticais.
- Permite linhas de grelha.
- Permite faixa de Linhas.
- Permite *layout* flexível, permitindo que seja possível possuir mais do que um gráfico ao mesmo tempo.
- Os eixos suportam a funcionalidade de *auto-escala* e *autostep*. A funcionalidade de *autostep* introduz um nível novo e melhorado de redimensionamento do gráfico que permite aos eixos automaticamente reorganizarem os seus valores, a fim de evitar a sobreposição de rótulos.
- Permite valores negativos.
- Permite séries Horizontais.
- Permite exportação para diferente formatos, entre os quais: PNG, BMP, XLSX, SVG.
- Permite a seleção individual de séries, fazendo com que as outras fiquem semitransparentes a fim de ressaltar a série selecionada.
- Suporta dados hierárquicos.

Em termos de licenças e preços, a Figura 95 apresenta aquilo que é oferecido pela empresa.

Figura 95. Características da licença.

De uma forma mais detalhada as licenças:

- Só permitem que um desenvolvedor possa trabalhar com o produto em tempo de design.
- Não têm limitações de tempo de execução e permitir que o desenvolvedor que possua a licença possa utilizar os controlos do *Telerik* para um número ilimitado de aplicativos espalhados por vários servidores (e domínios).
- Os controlos funcionam sem as chaves de licença e podem ser instalados em mais do que uma máquina, sendo que somente o desenvolvedor que possui a licença pode usar o produto em tempo de *design*.
- Todas as licenças do produto são permanentes e livres.

5.2.1.2 Análise mais profunda.

Nesta secção irá ser feita uma análise mais profunda da biblioteca através da implementação do projeto.

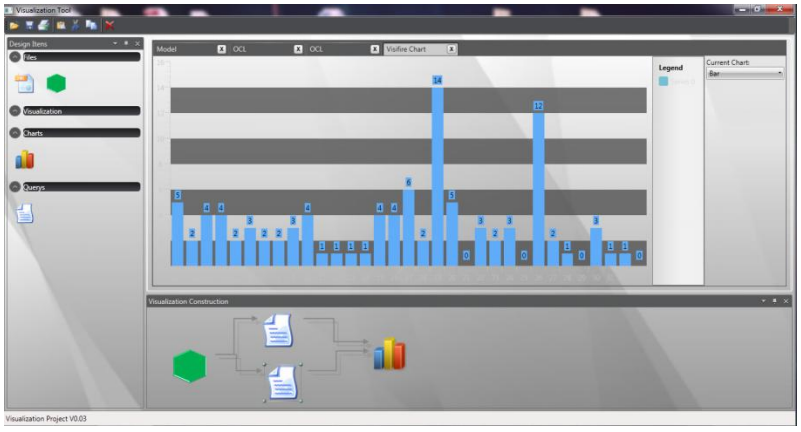

Figura 96. O projeto com a utilização da biblioteca.

A Figura 96 exibe como é que o projeto, na sua versão 0.1, ficou após a utilização da biblioteca.

Em primeiro lugar, a biblioteca oferece um conjunto de controlos adicionais para além dos controlos que permitem a geração dos gráficos. Estes controlos têm várias limitações ao nível de *layout*. Por exemplo o controlo que permite fazer *docking* não é simples e é necessário definir os tamanhos de cada painel. Todavia, se for realizado um ajustamento da janela estes desaparecem ou só mostram metade da informação.

Em segundo lugar, um aspeto positivo encontrado é que é possível definir o mesmo estilo aos diferentes controlos. Por outro lado, é necessário definir o mesmo estilo em cada componente que se pretende.

Em termos de elaboração de gráficos, estes realmente possuem as características que o desenvolvedor forneceu. Todavia existe um senão que é o fator da performance, foi notada uma baixa performance na elaboração do gráfico que se pretendia, isto tendo em conta a performance que tinham-se com a biblioteca inicial de teste (*Visifire*). Para além dos problemas de performance, não é simples a mudança de tipos de gráficos sendo que a única forma de o conseguir é através de código *behind* o que não vai ao encontro da filosofia MVVM.

Em último lugar, o aspeto não é realmente apelador sendo necessário alterar os estilos que são fornecidos pela biblioteca, o que implica mais trabalho.

Tabela 10. Requisitos apurados Telerik.

Requisitos apurados	
Gráficos 2D	✓ (24 tipos)
Gráficos 3D	✓ (18 tipos)
Interação	✓
Performance	Media
Aparência	Media
Binding	✓
Preço	799$
Desconto	10% Para estudantes
Licenciamento	Todas as licenças do produto são permanentes e livres.
Controlos adicionais	✓ (43 controlos adicionais)

A Tabela 10 apresenta os principais requisitos que foram apurados desta biblioteca.

5.2.2 *ComponentArt*

A empresa *CompenentArt* especializa-se na criação de vários componentes para diferente tecnologia. Ela possui uma biblioteca chamada *DataVisualization*. Esta biblioteca oferece cinco tipos de controlos, nomeadamente: *charting*, *gauges*, *maps*, *gridView*, *timeNavigator* e *calcEngine*.

5.2.2.1 Primeira Análise

Esta biblioteca implementa a tecnologia *pixel shading* 3D que em conjunto com o XAML permite ter uma imagem brilhante e com maior qualidade. Esta tecnologia é capaz de processar milhares de elementos por segundo, sendo que o resultado é uma interface atraente com animações suaves o que permite aumentar a produtividade e o prazer na utilização das ferramentas que utilizam esta biblioteca.

A visualização de dados foi projetada para permitir que grandes quantidades de dados possam ser manipuladas através de um WCF *service* altamente eficaz. Além disso todos os controlos são capazes de lidar com dezenas de milhares de dados.

Um acesso comum aos dados e um poderoso mecanismo de cálculo/ processamento de dados estão no núcleo da visualização de dados. Para além disto, a *suite* inclui alguns elementos genéricos para a construção de interface de utilizadores como é o casso do *DashboardLayout*, *DashboardPanel* e *DashboardPopup*. Todos estes elementos são reutilizados por todos os controlos de visualização de dados o que permite uma *framework* comum e consistente em toda a aplicação.

Um dos objetivos da biblioteca é aumentar a produtividade do utilizador final. Isto é efetuado através do fornecimento de animações tais como: *drill-down*, possibilidade de seleção de dados, *popups* com informação relevante, *zoon*, rótulos interativos entre outros.

Para além da interatividade adicionada, é possível utilizar temas que irão ser aplicados a todas as características visuais da interface. Todavia, existe também a possibilidade de utilizar paletes de cores que são partilhadas por toda a *suite*.

Em termos de gráfico a biblioteca permite criar os seguintes: *Bar* & *Column Charts, Pie* & *Donut Charts, Line Charts, Area Charts, Bubble & Marker Charts, Financial & Stock Charts, Funnel Charts, Radar Charts, TreeMaps*.

Existem várias opções que permite que os gráficos sejam apresentados com duas ou três dimensões enriquecendo a visualização de dados.

Em termos de licenças e preços, a Figura 94 apresenta o que é oferecido pela empresa.

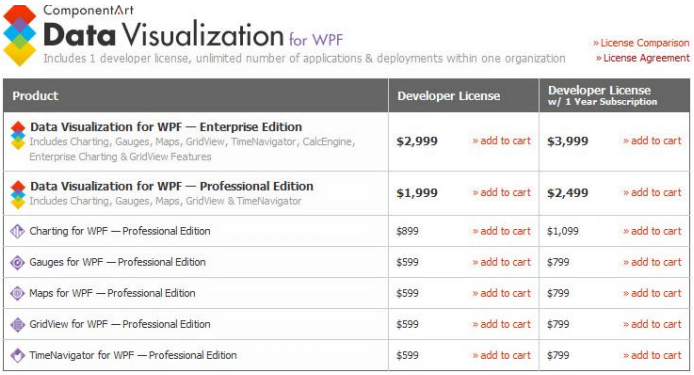

Figura 97. Características da licença.

A licença é válida para um desenvolvedor, sendo que é possível distribuir as aplicações desde que as aplicações sejam criadas pelo mesmo.

5.2.2.2 Análise mais profunda.

Após criar pequenas aplicações percebe-se que o *data binding* é complexo em relação a outras bibliotecas como o *Visifire*. Também consegue-se observar que consoante o desempenho que esta tem o processador existe uma má *renderização*, o que faz com que diminua a aparência da aplicação.

Tabela 11. Requisitos apurados *ComponentArt*.

Requisitos apurados	
Gráficos 2D	✓ (13 tipos)
Gráficos 3D	✓
Interação	✓
Performance	✓
Aparência	Alta
Binding	✓
Preço	1.999$
Desconto	25%
Licenciamento	Todas as licenças do produto são permanentes e livres.
Controlos adicionais	✓

A Tabela 11 apresenta os principais requisitos que foram apurados desta biblioteca.

5.2.3 *Infragistics*

Esta empresa oferece vários controlos para as diferentes tecnologias existentes em C#. No caso do projeto, está-se interessado na biblioteca *NetAdvantage* WPF. Esta biblioteca oferece um conjunto de controlos cuja interface é agradável e estes permitem criar:

- Aplicações empresariais para os ambientes Windows
- Aplicações em XBAP para aplicações baseadas na Web.
- Aplicações que utilizam estilos que se encontram nos *ThemePacks*.

Os controlos que se encontram no NetAdvantage WPF foram projetados a partir do zero de forma a aproveitar o poder que oferece o *Windows Presentation Foundation*.

5.2.3.1 Primeira Análise

Em termos de gráficos a biblioteca permite criar 27 gráficos permitindo a possibilidade de alguns destes possam ser em 3D. Para além disso, os controlos desta biblioteca permitem a interação com o utilizador, *data binding* e *tooltips* o que melhora a experiência com o utilizador.

Em termos de aparência, esta biblioteca apresenta uma boa aparência, não possui problemas de desempenho e possui um *data binding* simples.

Tabela 12. Requisitos apurados *Infragistics.*

Requisitos apurados	
Gráficos 2D	✓ (27 tipos)
Gráficos 3D	✓
Interação	✓
Performance	✓
Aparência	Alta
Binding	✓
Preço	€829.00
Desconto	10%
Licenciamento	Todas as licenças do produto são permanentes e livres.
Controlos adicionais	✓ (10 controlos adicionais)

A Tabela 12 apresenta os principais requisitos que foram apurados desta biblioteca.

5.2.4 *Visifire*

O *Visifire* é um conjunto de controlos multidireccionado que pode ser usado tanto em aplicações WPF e Silverlight. Os *Visifire Silverlight Controls* também podem ser incorporados em qualquer página da Web como uma aplicação autónoma Silverlight. O *Visifire* é independente da tecnologia do lado do servidor. Pode ser usado com: ASP, ASP.Net, SharePoint, PHP, JSP, ColdFusion, Python, Ruby ou HTML.

5.2.4.1 Primeira Análise

Em termos de características é possível citar as seguintes:

- Permite criar gráficos agradáveis em Silverlight e WPF em poucos minutos.
- Uma única API para ambas as tecnologias Silverlight e WPF.
- Os controlos do *Visifire* são *multi-targeting*.
- Pode ser utilizado para aplicações Desktop, Web ou móveis.

- É Compatível com *Microsoft Expression Blend*, permitindo que todos os gráficos sejam editados com auxilio desta ferramenta.
- Permite a geração de gráficos em tempo real e permite que eles sejam atualizados em tempo real, sendo que as propriedades dos gráficos Visifire podem ser atualizados em tempo real usando código .Net ou JavaScript.
- É independente da tecnologia do lado do servidor.
- Em termos de gráficos, esta biblioteca suporta 19 gráficos em 2D e o mesmo número em 3D.

5.2.4.2 Análise mais profunda.

Em primeiro lugar, a biblioteca permite, de forma muito simples, fazer *data binding*, permite ter *tooltips*, possui temas configuráveis e é uma biblioteca bastante completa, permitindo também a interação com o utilizador e *drill-down*.

Em segundo lugar, a sua utilização é simples e rápida, possui bom desempenho e não foram encontrados quase nenhum problemas ou dificuldades em quanto foi utilizada a versão trial da biblioteca.

Por último, existem vários exemplos disponíveis que ajudam a perceber melhor como fazer as coisas e a documentação disponível é bastante boa.

Tabela 13. Requisitos apurados *Visifire*.

Requisitos apurados	
Gráficos 2D	✓ (19 tipos)
Gráficos 3D	✓ (19 tipos)
Interação	✓
Performance	✓
Aparência	Alta
Binding	✓
Preço	$399.00 sem suporte
Desconto	20%
Licenciamento	Todas as licenças do produto são permanentes e livres.
Controlos adicionais	✓ (2 controlos adicionais)

A Tabela 13 apresenta os principais requisitos que foram apurados desta biblioteca.

5.2 Conclusão

Sempre que se faz um produto de *software* é necessário considerar se irá ser compradas/usadas componentes já feitas que permitem poupar trabalho, ou se irá ser desenvolvidas as componentes. No caso de as componentes serem gratuitas, de se ter acesso ao código fonte e serem de boa qualidade, quase sempre não é pensado na possibilidade de fazer as componentes. Mas devido a que encontrar este tipo de componentes é muito raro, por vezes é necessário optar por decidir entre comprar ou produzir as componentes. No caso do C# isto é uma decisão que acontece muitas vezes devido a que a maior partes das componentes não são gratuitas e por isso é necessário analisar várias componentes de forma a encontrar a aquela que tenha a melhor relação preço qualidade.

Neste documento foi analisado várias soluções que se encontram mercado e aquelas que apresentam o melhor aspeto. Aquela que apresenta a melhor relação preço qualidade é a biblioteca *Visifire*. Portanto devido as características desta biblioteca e ao seu preço este é a biblioteca que é a melhor para a ferramenta.