Ana Isabel Silvestre Fernandes

Licenciatura em Engenharia Informática

Estudo comparativo entre Interfaces de Programação de Aplicações de mapas

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientadores : Profª. Dr.ª Maria Armanda S. Rodrigues Grueau,
Profª. Auxiliar, Universidade Nova de Lisboa
Prof. Dr. Miguel Carlos Pacheco Afonso Goulão,
Prof. Auxiliar, Universidade Nova de Lisboa

Júri:

Presidente:	Prof. Doutor Pedro Abílio Duarte de Medeiros
Arguente:	Profª. Doutora Ana Paula Pereira Afonso
Vogal:	Profª. Doutora Maria Armanda Simenta Rodrigues Grueau

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Novembro, 2012

Estudo comparativo entre Interfaces de Programação de Aplicações de mapas

Aos meus pais

Agradecimentos

Este espaço é dedicado àqueles que, direta ou indiretamente, deram a sua contribuição para que esta dissertação fosse realizada. A todos eles deixo aqui o meu agradecimento sincero.

Aos meus orientadores, os Professores Armanda Rodrigues e Miguel Goulão, agradeço a forma como orientaram o meu trabalho. Agradeço o apoio, as sugestões e críticas, o trabalho, o interesse, a disponibilidade e a paciência que demonstraram ao longo da realização deste trabalho. Foi bastante prestigiante, gratificante e enriquecedor trabalhar sobre a vossa orientação. Estou muito grata pelo vosso apreço e por partilharem a vossa experiência e o vosso conhecimento.

A todos os meus colegas e amigos, pela prestimosa colaboração, amizade e espírito de entreajuda durante o meu percurso académico, dos quais, gostaria de destacar os nomes de Sérgio Silva, Pedro Almeida e André Vieira.

Apesar de não estarem ligados diretamente a esta dissertação, gostaria de agradecer aos meus amigos Filipa Oliveira, Mariana Almeida, João Ramos e João Campos, pelo seu apoio e amizade.

Ao Ricardo, que sempre me acompanhou dando-me força, mostrando interesse, e ajudando-me sempre que estava ao seu alcance.

Aos meus pais agradeço todo o apoio e carinho que me deram, e os valores que me transmitiram durante toda a minha educação.

À minha irmã pela verdadeira amizade em todos os momentos e circunstâncias.

Resumo

Este trabalho debruça-se sobre APIs de mapas para utilização em aplicações com características de georreferenciação, e tem como objetivo realizar um estudo comparativo entre APIs desta natureza, por forma a que o utilizador possa tomar uma decisão informada na escolha de uma API de mapas mais adequada ao seu objetivo.

Neste contexto, esta dissertação foca-se no estudo de duas categorias de APIs de Mapas: as de âmbito empresarial e *open source*. Dentro da categoria empresarial, avaliou-se uma API de grande impacto, a Google Maps JavaScript API da Google e uma diretamente ligada à ciência/engenharia da informação geográfica, a ArcGIS API for JavaScript da Esri. A API de *open source* escolhida foi a OpenLayers JavaScript Mapping Library da Open Source software community.

Como base para a realização do estudo, existem três elementos distintos: três protótipos baseados em cada uma das APIs de mapas escolhidas; as próprias APIs; e um conjunto de aplicações suportadas pela Google Maps JavaScript API, submetidas como trabalhos no contexto da disciplina de Tecnologias de Informação Geográfica (TIG), do Mestrado em Engenharia Informática da FCT/UNL. A todos estes elementos aplicaram-se métricas de avaliação de usabilidade, através de um programa desenvolvido para o efeito. Os protótipos desenvolvidos implementam funcionalidades básicas presentes nas aplicações georreferenciadas reais, e o conjunto de aplicações incluem determinados requisitos mínimos, que, neste caso, são semelhantes aos das funcionalidades básicas dos protótipos.

Assim, o estudo desenvolvido permite, com base nas métricas, tirar conclusões sobre os objetos mais utilizados, a dimensão das APIs, a facilidade de compreensão das APIs e o estado de evolução de cada API.

Palavras-chave: APIs de mapas, Usablidade de APIs, Métricas, Reusabilidade, Online Mapping, Web GIS, Google Maps, ArcGIS API for JavaScript, OpenLayers.

Abstract

This work focuses on map APIs, regarding its use in applications with geocoding characteristics, and, as the main objective, it intends to perform a comparative study between APIs of this kind, allowing the user to make an informed decision of the most adequate and correct choice of a map API that greatly fits his purpose.

In this context, this dissertation focuses on the study of two map APIs categories: enterprise environments and open source environments. In the enterprise environment, we studied a very powerful and used worldwide API: the Google Maps JavaScript API, from Google, and also one directly connected to geographic information science/engineering: the ArcGIS API for JavaScript, from Esri. The chosen API, regarding the open source environment, was the OpenLayers JavaScript Mapping Library, from the Open Source software community.

As the main comparison basis to perform the study, we considered three distinct elements: three prototypes based on each of the chosen map APIs; the APIs themselves and a set of applications, supported by the Google Maps Javascript APIs, built for evaluation in the Geographic Information Technology (GIT) course of FCT/UNL's M.Sc. degree in Computer Science. To all these elements we applied some evaluation metrics, considering usability, using a program built exclusively for this purpose. The developed prototypes implement essential functionalities, which serve as a common base in real geo-referenced applications, and the set of applications, developed in the TIG course, include some particular minimum requisites, which, in this case, are similar to the basic properties of the prototypes.

Thus, the developed study allows, based on the proposed metrics, to draw conclusions about the most used objects, the size of APIs, the ease of understanding of APIs and the state of evolution of each API.

Keywords: Mapping APIs, API Usability, Metrics, Reusability, Online Mapping, Web GIS, Google Maps, ArcGIS API for JavaScript, OpenLayers.

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

1.1 Descrição e contexto

O desenvolvimento de software pode ser facilitado com a utilização de Interfaces de Programação de Aplicações (Application Programming Interfaces - APIs), que consistem em especificações destinadas a serem utilizadas como interfaces, por componentes de software, para a comunicação entre eles [DFMS09, dSRC+04]. Uma das razões pela qual se desenvolvem as APIs, é o facto de estas fornecerem a abstração da informação de forma a que seja possível alterar detalhes de implementação sem afetar o código que a reutiliza e, consequentemente, reduzirem a complexidade de desenvolvimento de novo software. Assim, a reutilização de software é o processo de criação de sistemas de software a partir de software existente ou conhecimento adquirido a partir de outro software, em vez da construção de sistemas de software de raiz [Kru92, FK05], e o seu propósito é melhorar a qualidade de software e a produtividade [FK05].

É importante que durante o desenvolvimento das APIs se tenha em conta atributos de qualidade, pois ao serem considerados o resultado obtido será próximo de uma boa API.

A norma ISO/IEC 9126 [iso91] propõe um conjunto de características que permitem distinguir a qualidade de um produto. Estas características compreendem vários atributos de qualidade externa e interna do software, e são distribuídos por seis categorias principais:

- **Funcionalidade** - consiste num conjunto de atributos que incidem sobre a existência de um conjunto de funções e as suas propriedades especificadas, e em que as funções são as que satisfazem necessidades explícitas ou implícitas;
- **Fiabilidade** - consiste num conjunto de atributos que incidem sobre a capacidade

do software manter o seu nível de desempenho sob condições estabelecidas durante um determinado período de tempo;

- **Usabilidade** - consiste num conjunto de atributos que incidem sobre o esforço necessário para a utilização, e na avaliação individual de tal utilização, por um conjunto, explícito ou implícito, de utilizadores;
- **Eficiência** - consiste num conjunto de atributos que incidem sobre a relação entre o nível de desempenho do software e a quantidade de recursos usados, sob condições estabelecidas;
- **Manutenibilidade** - consiste num conjunto de atributos que incidem sobre o esforço necessário para fazer modificações específicas;
- **Portabilidade** - consiste num conjunto de atributos que incidem sobre a capacidade do software ser transferido de um ambiente para outro.

As qualidades externas correspondem às qualidades medidas durante a execução, e compreendem a funcionalidade, a fiabilidade, a usabilidade e a eficiência. As qualidades internas correspondem às qualidades que não podem ser medidas em tempo de execução, e compreendem a facilidade de manutenção e a portabilidade.

Das categorias enumeradas, no contexto desta dissertação, a categoria de interesse é a usabilidade. Na secção seguinte explicamos qual a motivação para explorar esta área.

1.2 Motivação

Uma API é normalmente utilizada por programadores, quer sejam principiantes ou experientes, mas também pode ser utilizada por alguém que não está diretamente ligado ao desenvolvimento de software. No entanto, para qualquer um destes utilizadores é importante que a API realmente melhore a produtividade, e não o contrário. Esta preocupação tem sido a propulsora de vários estudos sobre a qualidade das APIs e uma vez que uma das qualidades básicas ao mais alto nível é a usabilidade [SM07], esta tem sido uma área de estudo para vários investigadores [MRTS98, Cla04, Cla06, Rob09, Hen09, ESM07, SM07, SC07, ZER11, FWZ10, MC11, XAC09].

A usabilidade consiste num conjunto de atributos que incidem sobre o esforço necessário para utilização, e na avaliação individual de tal utilização, por um conjunto, explícito ou implícito, de utilizadores. Tais atributos incluem a inteligibilidade, a apreensibilidade, a operacionalidade e a atratividade [iso91]. A inteligibilidade representa a facilidade com que o utilizador compreende as funcionalidades oferecidas, e a facilidade em avaliar se estas podem ser utilizadas para satisfazer as suas necessidades específicas. A apreensibilidade identifica a facilidade de aprendizagem. A operacionalidade identifica como o produto facilita a sua operação por parte do utilizador, incluindo a forma como tolera erros de operação. Por fim, a atratividade envolve características que possam

atrair um potencial utilizador para a utilização do produto em questão [iso91]. Existem alguns estudos [Rob09, Cla06], que focam a sua investigação em APIs da Microsoft, onde se discutem os problemas que afetam a usabilidade.

Como dissemos anteriormente, uma API é uma especificação destinada a ser utilizada como uma interface, por componentes de software, para a comunicação entre eles. No entanto, uma especificação de API pode assumir várias formas, incluindo um padrão internacional, tal como POSIX ou fornecedor de documentação, tal como o Microsoft Windows API, ou as bibliotecas de uma linguagem de programação, por exemplo, Standard Template Library em C++ ou Java API [1]. Mas existe uma das categorias em crescimento [1], que é a categoria de APIs de mapas. Estas permitem a visualização on-line e manipulação de dados georreferenciados, e a disponibilidade destas APIs é o resultado do acesso global à captura e armazenamento de dados espaciais. As ferramentas disponíveis variam de pacotes comerciais, ligados aos SIG (Sistemas de Informação Geográfica), com forte suporte para operadores de análise espacial, a bibliotecas *open source*, onde a contribuição das comunidades interessadas tem vindo a fazer a diferença.

O desenvolvimento de aplicações que disponibilizam e manipulam informação georreferenciada é uma área emergente. Estas aplicações são bastante comuns em dispositivos móveis, em alguns casos com integração de GPS, e existe também grande crescimento de aplicações georreferenciadas na Internet [2] (chamaremos a estas aplicações Web - SIG). O desenvolvimento destas aplicações é, em muitos casos, suportado pelo uso de APIs de Mapas disponibilizadas pela Google[3], ESRI[4], Microsoft[5], OpenStreetMap(Open Source software community)[6], entre outras, pelo que um programador que pretenda desenvolver aplicações Web - SIG terá várias opções de APIs de mapas para escolher. É devido a esta tarefa de escolha que pretendemos desenvolver o nosso estudo, pois um dos fatores que poderá ajudar a decidir que API de mapas se adapta melhor às necessidades do programador, é a usabilidade destas. Portanto, como não foram encontrados estudos sobre a usabilidade de APIs de mapas, que permitissem assim alguma ponderação por parte de um programador indeciso na sua escolha, o estudo desenvolvido no contexto desta dissertação pretende contribuir para mitigar este problema.

1.3 Objetivo do trabalho

O objetivo deste trabalho é fornecer, ao programador, informação sobre a usabilidade de APIs de mapas, de forma a que este possua mais um fator de ponderação sobre a escolha de APIs deste tipo. A avaliação que se pretende realizar é conduzida do ponto de vista do utilizador da API, que é neste caso o programador. Desta forma, o foco da avaliação

[1] http://www.programmableweb.com/apis/directory/1?apicat=Mapping
[2] http://trends.builtwith.com/mapping/Google-Maps
[3] Google Maps API Family - http://code.google.com/intl/pt-PT/apis/maps/index.html
[4] ArcGIS Web APIs - http://resources.arcgis.com/content/web/web-apis
[5] Bing Maps APIs - http://msdn.microsoft.com/en-us/library/dd877180.aspx
[6] OpenLayers - http://trac.osgeo.org/openlayers/wiki/Documentation#OpenLayersJavaScriptAPIDocumentation

será a usabilidade das APIs de mapas, qualidade que compreende subcategorias, como a inteligibilidade, a apreensibilidade, a operacionalidade e a atratividade [iso91], diretamente relacionadas com o utilizador.

Como referido atrás, as APIs de interesse neste estudo, são as APIs de mapas, utilizadas em aplicações com características de georreferenciação. Neste contexto, analisaram-se as várias APIs disponíveis, focando-se o estudo em duas categorias de APIs de Mapas: as de âmbito empresarial e *open source*. Dentro da categoria empresarial, decidiu-se avaliar uma API de grande impacto, a Google Maps JavaScript API da Google e uma diretamente ligada à ciência/engenharia da informação geográfica, a ArcGIS API for JavaScript da Esri. A API de *open source* escolhida foi a OpenLayers JavaScript Mapping Library da Open Source software community.

Nesta dissertação estudamos as APIs em si, três protótipos e um conjunto de aplicações desenvolvidas com suporte da API da Google. Através do código das APIs é possível avaliar a facilidade de compreensão entre as APIs utilizadas no estudo. E ainda, através do código de várias versões de cada API, será possível realizar um estudo de evolução das APIs. Os protótipos implementam exatamente o mesmo conjunto de funcionalidades, que permitem manipular uma aplicação orientada ao turismo. Desta forma, podemos avaliar a utilização efetiva de cada API na resolução de um mesmo problema concreto. Além da avaliação da utilização efetiva através dos protótipos desenvolvidos, graças à disponibilidade de um conjunto de aplicações suportadas pela Google Maps JavaScript API, e submetidas como trabalhos no contexto da disciplina de Tecnologias de Informação Geográfica (TIG), do Mestrado em Engenharia Informática da FCT/UNL, foi possível desenvolver uma análise da utilização efetiva da mesma API. Os trabalhos desenvolvidos incluem determinados requisitos mínimos, que, neste caso, são semelhantes aos das funcionalidades básicas dos protótipos. Tanto as funcionalidades básicas, como os requisitos mínimos dos trabalhos tomados em consideração, são apresentados no capítulo 4.

Apesar dos estudos de usabilidade de APIs encontrados não incidirem sobre a categoria de APIs de mapas, é importante perceber que técnicas e que questões são abordadas nestes estudos de forma a obter pontos de referência.

A pesquisa sobre o trabalho relacionado, discutido no capítulo 3, permite concluir que a abordagem mais adequada para o estudo a desenvolver, considerando os recursos disponíveis para a realização desta dissertação de Mestrado, é a avaliação quantitativa de artefatos de software, neste caso os protótipos e as APIs, através de ferramentas desenvolvidas especificamente para o efeito, aplicando métricas de avaliação. Uma métrica consiste na medição de uma propriedade de um artefato de software ou das suas especificações. As propriedades avaliadas são requisitos não funcionais como a reusabilidade, a adaptabilidade, a coesão, o acoplamento, etc.

Desta forma, iremos utilizar um conjunto de métricas que combina diferentes propostas de métricas independentes. As métricas selecionadas foram originalmente propostas para a avaliação de APIs, mas não para APIs para mapas. No capítulo 4 apresentamos as

métricas selecionadas, descrevendo o que estas permitem avaliar.

1.4 Principais contribuições

Todos os estudos realizados, ou seja, o estudo de comparação entre as APIs, o estudo da evolução das APIs e o estudo de utilização da API da Google, contribuem para a área de pesquisa relacionada com estudos de APIs de mapas.

Também toda a implementação inerente ao estudo é uma contribuição. Os protótipos são uma contribuição à utilização das APIs, e portanto podem ser utilizados noutros estudos sobre API de mapas. O programa desenvolvido para a recolha de dados é uma contribuição para possíveis análises futuras das versões das APIs que irão sendo lançadas, ou para outros protótipos.

A utilização das métricas envolvidas no estudo, contribui para a validação das métricas aplicadas. Para as métricas ainda não validadas, passa a existir algum suporte de validação das mesmas, para as métricas já antes testadas, pode ser visto como mais um reforço à validação já realizada.

Por fim, os resultados e conclusões são contribuições que poderão ajudar um programador a decidir que API é mais adequada às suas necessidades.

1.5 Estrutura do documento

O documento está organizado da seguinte forma. Após esta introdução, em que os conceitos envolvidos no estudo são abordados de forma mais superficial, o capítulo 2 apresenta os conceitos base envolvidos na temática da dissertação, onde os termos já serão explicados de forma mais detalhada. No capítulo 3, abordamos o estado da arte. No capítulo 4, introduzimos a abordagem seguida para a realização do estudo. No capítulo 5, descreve-se que tecnologias foram utilizadas, e os detalhes mais relevantes da implementação de suporte ao estudo. No capítulo 6, apresentamos os resultados do estudo e discutem-se os mesmos. Por fim, no capítulo 7, descrevem-se as conclusões finais do trabalho realizado nesta dissertação, sendo também efetuadas algumas sugestões para trabalho futuro.

2

Conceitos Base

O objetivo deste capítulo é introduzir o leitor aos conceitos bases desta dissertação.

Desde que o acesso a um computador se tornou mais fácil, o número de programadores aumentou e, consequentemente, também o código desenvolvido aumentou. De início, se um programador queria criar software, tinha de escrever código de raiz. Atualmente, com a quantidade de código disponível, é possível ter acesso a código com muitas das funcionalidades de que necessitamos implementadas, o que facilita o desenvolvimento de aplicações.

A reutilização é, portanto, o processo de criação de sistemas de software a partir de software existente ou de conhecimento adquirido a partir de outro software, em vez da construção de sistemas de software de raiz [Kru92, FK05]. Os ativos reutilizáveis podem ser tanto de software reutilizável como de conhecimento adquirido a partir de outro software.

O propósito da reutilização de software é melhorar a qualidade de software e a produtividade, permitindo que se desenvolvam sistemas maiores e mais complexos, mais confiáveis, menos dispendiosos e que são entregues dentro do tempo estipulado[FK05]. Para que a reutilização de software seja algo realmente útil, é importante que os módulos de código sejam, o mais possível, independentes uns dos outros, sendo esta independência alcançada através do nível de abstração. Esta abstração permite que os programadores sintam a independência entre módulos, e facilita aspetos de colaboração. Este é o princípio subjacente às APIs.

2.1 Interfaces de Programação de Aplicações(Aplication Programming Interfaces -APIs)

Uma API é uma especificação que facilita a troca de mensagens ou dados entre duas ou mais aplicações de software diferentes [DFMS09, dSRC+04].

Uma definição de API mais informal, e normalmente adotada entre engenheiros de software profissionais, é apresentada em [dSRC+04], e abrange qualquer interface bem definida que defina o serviço que um componente, um módulo, ou uma aplicação fornece a outros elementos de software.

A criação e utilização de APIs deve-se a várias razões. Primeiro, o programador economiza tempo ao reutilizar funcionalidades que a API fornece. Segundo, uma API proporciona a ocultação de detalhes de implementação, possibilitando assim a alteração/evolução desses detalhes, sem afetar o código que faz uso desta. Por fim, muitas APIs disponibilizam funcionalidades de difícil implementação [SM07], o que simplifica o desenvolvimento de aplicações que necessitam dessas funcionalidades. Isto tem impacto positivo na qualidade do software desenvolvido, pois precisamente por proporcionarem funcionalidades de difícil implementação e por serem usadas por muitos utilizadores, as *frameworks* que estão por detrás das APIs são tendencialmente mais testadas por toda a sua comunidade de utilizadores, de modo que a remoção de eventuais defeitos que as *frameworks* tenham, vão sendo naturalmente, resolvidos. Assim, ao reutilizarmos uma API, a probabilidade desta conter defeitos será menor do que a de introduzimos inadvertidamente erros no código, ao programar de raiz as mesmas funcionalidades. Finalmente, um erro corrigido na *framework* por detrás da API fica corrigido para todos os utilizadores da API que tenham acesso à versão já corrigida, o que proporciona um valioso efeito de escala.

Uma API pode ser a interface para uma *framework*, biblioteca, *toolkit* ou SDK(software development kit). Apesar de existirem algumas distinções significativas entre estes tipos de APIs, como por exemplo a sua dimensão, a finalidade e utilização das mesmas recai sobre o mesmo conceito, a reutilização de código [SM07, DFMS09].

Tal como já referimos na introdução, uma API é normalmente utilizada por programadores, quer sejam principiantes ou experientes, mas também pode ser utilizada por alguém que não esteja diretamente ligado ao desenvolvimento de software. Para qualquer um destes utilizadores é importante que a API realmente melhore a produtividade, e não o contrário. Esse fator deve ser tido em conta quando se está a desenvolver uma API, e para tal são consideradas várias decisões de desenho que têm impacto na qualidade das APIs resultantes. As qualidades básicas, ao mais alto nível, de uma API são a sua usabilidade e a sua capacidade[SM07].

A usabilidade consiste num conjunto de atributos que incidem sobre o esforço necessário para utilização, e na avaliação individual de tal utilização, por um conjunto, explícito ou implícito, de utilizadores. Tais atributos compreendem a inteligibilidade, a

apreensibilidade, a operacionalidade e a atratividade [iso91]. A inteligibilidade representa a facilidade com que o utilizador compreende as funcionalidades oferecidas, e a facilidade em avaliar se estas podem ser utilizadas para satisfazer as suas necessidades específicas. A apreensibilidade identifica a facilidade de aprendizagem. A operacionalidade identifica como o produto facilita a sua operação por parte do utilizador, incluindo a forma como tolera erros de operação. Por fim, a atratividade envolve características que possam atrair um potencial utilizador para a utilização do produto em questão [iso91]. Por exemplo, Dekel e Herbsleb [DH09] discutem a importância de realçar determinadas informações importantes, por exemplo de um método, na documentação de uma API, por forma a melhorar a performance dos utilizadores da API que terão mais facilidade em aprenderem e a compreenderem a mesma.

Testes de usabilidade realizados como parte do desenvolvimento de APIs, demonstraram que APIs com maior usabilidade melhoram a produtividade dos programadores que as usam [Cla04, Cla06, Rob09].

A capacidade de uma API, refere-se aos limites ao código que pode ser criado, e inclui atributos como a expressividade, a extensibilidade, a facilidade de evolução, a performance e a robustez.

Por fim, como referido anteriormente, as APIs podem variar em dimensão e, quando se trabalha com uma API de grandes dimensões, esta pode tornar-se uma barreira à produtividade dos programadores. Apesar de uma API grande oferecer muitas funcionalidades e esperar-se que assim também se obtenha bom impacto na produtividade, esta pode ser afetada a vários níveis: na curva de aprendizagem da API, na facilidade de memorização das chamadas às funcionalidades da API (com erros ou falhas de interpretação que os programadores fazem ao utilizarem a mesma) e na forma como os programadores compreendem a API [MRTS98].

2.2 Sistemas de Informação Geográfica(SIG)

Um Sistema de Informação Geográfica é um sistema que permite capturar, armazenar, manipular, gerir e apresentar dados georreferenciados [Wor95]. Os SIG podem estar associados a vários tipos de funcionalidades tais como, por exemplo, a combinação de vários recursos numa única visualização, ou mapa, o desenho de itinerários com base na informação de estradas, a integração de dados de diferentes fontes para gerar nova informação e a análise de terreno baseada em dados topográficos.

O campo da ciência de informação geográfica inclui matérias de cartografia, posicionamento, sistemas de informação e computação gráfica. A modelação geográfica é bastante importante pelo facto de a generalidade dos fenómenos ser passível de referenciação geográfica [dM01]. Um dos primeiros casos, reportados, de modelação geográfica e de utilização de métodos cartográficos para análise de um fenómeno geograficamente dependente, remonta a 1854, quando Jonh Snow descreveu um surto de cólera em Londres usando pontos para representar a localização de alguns casos individuais, num mapa. O

estudo da distribuição espacial dos casos de cólera levou à descoberta da localização da origem da doença, uma bomba de água contaminada[1].

Desde o lançamento das primeiras APIs de mapas em 2005 [Cho08], a manipulação de mapas passou de recurso acessório para elemento necessário de qualquer site que contenha conteúdo geográfico. Hoje em dia, cada vez mais pessoas estão interessadas na interação com mapas online e, com o poder que a internet oferece, a informação disponibilizada é cada vez mais distribuída. As APIs de mapas existentes permitem que programadores, mesmo que relativamente inexperientes, consigam construir aplicações contendo informação georreferenciada.

2.3 APIs de Mapas

O desenvolvimento de aplicações que disponibilizam e manipulam informação georreferenciada é uma área emergente. Estas aplicações são bastante comuns em dispositivos móveis, em alguns casos com integração de GPS. Existe também grande crescimento de aplicações georreferenciadas na Internet. O desenvolvimento destas aplicações é, em muitos casos, suportado pelo uso de APIs de Mapas disponibilizadas pela Google[2], ESRI[3], Microsoft[4], OpenStreetMap (*Open Source software community*)[5], entre outras.

As APIs que serão usadas para o estudo são APIs de mapas, que são um tipo de API diferente das APIs analisadas nos artigos de referência [MRTS98, Cla04, Cla06, Rob09, Hen09, ESM07, SM07, SC07, ZER11, FWZ10, MC11, XAC09]. Uma API de mapas é uma interface de código fonte que permite que os programadores tenham acesso a uma biblioteca e solicitem serviços para a geração, acesso, visualização e manipulação de um mapa sobre a Internet. Estas APIs surgiram graças à disponibilização de poderosos servidores com uma extensa cobertura de dados espaciais de todo globo. Os dados espaciais incluem dados vetoriais (por exemplo: rede de estradas, características hidrográficas, fronteiras políticas) e imagens oriundas de deteção remota (tanto aéreas como de satélite). Em geral, as imagens de alta resolução e os mapas de ruas só estão disponíveis em centros metropolitanos. Assim, uma API de mapas permite que o utilizador solicite dados geográficos através do Hypertext Transfer Protocol (HTTP), protocolo de acesso a dados na Internet, e incorpore o mapa resultante como um objeto em qualquer site externo [Cho08].

Este tipo de API permite ainda manipular características do mapa e integrar informação adicional no mesmo. As principais características oferecidas pelas APIs de mapas são: a gestão de marcadores, de *overlays* (sobreposição de dados), de direções de condução, de eventos sobre o mapa, áreas geográficas e georreferenciação.
Estas características compreendem o seguinte:

[1] http://www.ph.ucla.edu/epi/snow/1859map/
[2] Google Maps API Family - http://code.google.com/intl/pt-PT/apis/maps/index.html
[3] ArcGIS Web APIs - http://resources.arcgis.com/content/web/web-apis
[4] Bing Maps APIs - http://msdn.microsoft.com/en-us/library/dd877180.aspx
[5] OpenLayers - http://trac.osgeo.org/openlayers/wiki/Documentation#OpenLayersJavaScriptAPIDocumentation

- Marcadores - Os marcadores identificam localizações no mapa e são desenhados para serem interativos. Recebem eventos 'click', por exemplo, e são frequentemente utilizados com detetores de eventos para abrir janelas de informação.

- *Overlays* (sobreposição de dados): Os *overlays* são objetos no mapa que estão vinculados a coordenadas geográficas (latitude e longitude), que movem-se quando se arrasta ou amplia o mapa. Estes refletem objetos que o programador "adiciona"ao mapa para designar pontos, linha, áreas, ou coleções de objetos.

- Direcções - Cálculo de direções com base no meio de transporte, caso a API ofereça variedade. O meios possivelmente suportados são:

  - Condução: Indica direções de condução, utilizando a rede de estradas.
  - Caminhada: Indica direções de caminhada, utilizando os caminhos pedestres.
  - Ciclismo: Indica direções de ciclismo, utilizando caminhos de ciclismo e ruas preferidas.

- Eventos sobre o mapa - Alguns objetos do mapa são desenhados para responderem a eventos do utilizador, tais como eventos do rato ou do teclado. Normalmente os objetos respondem aos seguintes eventos: 'click', 'dblclick', 'mouseup', 'mousedown', 'mouseover' e 'mouseout'.

- Áreas geográficas - São polígonos, retângulos ou círculos.

- Georreferenciação - É o processo de conversão de endereços (como "FACULDADE DE CIÊNCIAS E TECNOLOGIA DA UNIVERSIDADE NOVA DE LISBOA, R. Quinta Da Torre, 2825 Almada, Portugal") para coordenadas geográficas (como latitude 38.65793 e longitude -9.20942), que podem ser utilizados para posicionamento de marcadores ou do mapa.

Dado o crescimento da utilização destas APIs[6], a existência de estudos comparativos sobre as mesmas permitirá, a quem pretende desenvolver uma aplicação com características de georreferenciação, uma decisão mais informada para a escolha da API mais adequada ao objetivo da aplicação.

[6] http://trends.builtwith.com/mapping/Google-Maps

2.4 Utilização de APIs de Mapas na Internet

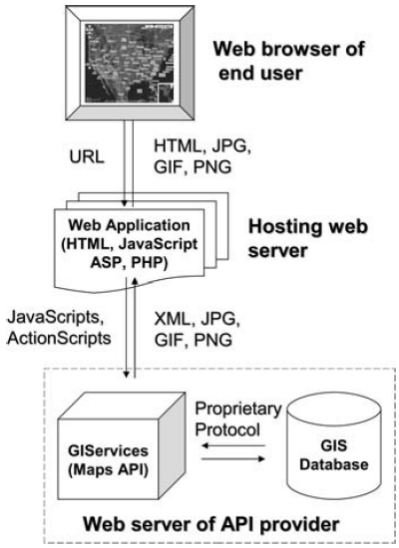

Figura 2.1: Arquitetura típica, de um site que usa funcionalidades e dados fornecidos por uma API de mapas

Na Figura 2.1 [Cho08], é apresentada uma arquitetura típica de uma aplicação web que utilize uma API de mapas. Em geral, a aplicação está hospedada num servidor web que irá devolver ficheiros codificados em Hypertext Markup Language (HTML), principal linguagem de marcação para páginas da internet, e gráficos compatíveis com a web (por exemplo, JPG, GIF, PNG) a pedido de um navegador web (*browser*). Usando JavaScript para fazer a ligação à API de mapas, a aplicação web tem acesso aos servidores do fornecedor da API fazendo pedidos de serviços de informação geográfica (GIServices), tais como *zoom in/out*. O JavaScript é uma linguagem de *script* que é dinâmica, fracamente tipificada, multi-paradigma e que suporta estilos de programação orientado a objetos, imperativos e funcionais [7].

Com base nos parâmetros de entrada e nos valores recolhidos pela interface do mapa da aplicação, o servidor do fornecedor da API irá devolver os dados espaciais (mapa) na forma de gráficos compatíveis com o navegador web (*browser*). Estes dados estão armazenados na base de dados do Sistema de Informação Geográfica (SIG) do fornecedor da API [Cho08].

2.5 Tecnologia relevante

Este trabalho irá debruçar-se sobre APIs de mapas para utilização em aplicações com características de georreferenciação, pelo que selecionámos 3 APIs de mapas, focando

[7] http://en.wikipedia.org/wiki/JavaScript

o estudo em duas categorias de APIs de Mapas: as de âmbito empresarial e as de *open source* . Dentro da categoria empresarial, decidiu-se avaliar uma API de grande impacto, a Google Maps JavaScript API da Google e uma diretamente ligada à ciência/engenharia da informação geográfica, a ArcGIS API for JavaScript da Esri. A API de *open source* escolhida foi a OpenLayers JavaScript Mapping Library da Open Source software community. Segue-se uma breve descrição de cada API.

A Google Maps API é constituída por várias APIs que permitem várias funcionalidades, sendo a mais relevante para este estudo a JavaScript Maps API V3. Este serviço é livre desde que o site no qual está a ser usado seja acessível ao público e que o acesso não seja cobrado. A Google lançou a Google Maps API em Junho de 2005, com o intuito de permitir que os programadores integrassem os mapas da Google nas suas aplicações web[8] [9].

O OpenLayers é uma única API, que permite a exibição de dados geográficos nas aplicações web. A API é disponibilizada em JavaScript. O OpenLayers constitui software livre, desenvolvido para e pela *Open Source Software Community*. A sua utilização facilita a utilização de mapas dinâmicos. O objetivo é separar as ferramentas do mapa dos dados do mapa, para que todas as ferramentas possam operar em todas as fontes de dados. Esta separação quebra a necessidade de ligação ao proprietário que anteriores revoluções SIG tinham ensinado a evitar. O OpenLayers é um projeto da *Open Source Geospatial Foundation* e foi lançado em 2007. Esta API permite a utilização de dados da Google Maps[10] [11] [12] [13].

O ArcGIS é um produto da Esri (Environmental Systems Research Institute). Associado ao pacote SIG desta companhia, existem as APIs ArcGIS Web Mapping que são uma coleção de APIs para a construção e integração de mapas na web. Entre estas APIs está a ArcGIS API for JavaScript, a mais relevante para este estudo. O desenvolvimento com esta API é livre mas a utilização do produto desenvolvido só é livre dependendo das circunstâncias. A API foi lançada em 2008[14] [15].

É importante referir que a historia da Esri está ligada aos SIG desde muito cedo (anos 60). A missão inicial da Esri era organizar e analisar informação geográfica de forma a dar suporte a gestores de ordenamento do território e a responsáveis por planos de ordenamento, para possibilitar a tomada de decisões ambientais bem informadas. Mais tarde, em 1982, a Esri passou de projeto para produto e lançou o primeiro SIG comercial[16].

[8] http://code.google.com/intl/ptPT/apis/maps/index.html
[9] http://en.wikipedia.org/wiki/Google_Maps#History
[10] http://www.osgeo.org/node/462
[11] http://openlayers.org/
[12] http://trac.osgeo.org/openlayers/wiki/History
[13] http://docs.openlayers.org/library/layers.html#google
[14] http://www.esri.com/software/arcgis/web-mapping/index.html
[15] http://blogs.esri.com/Dev/blogs/arcgisserver/archive/2008/07/08/The-ArcGIS- JavaScript-API-is-now-available-to-the-public.aspx
[16] http://www.esri.com/about-esri/about/history.html

3

Estado da arte

3.1 Trabalho Relacionado

Tendo em conta que o estudo que se pretende realizar é centrado na perspectiva do utilizador, a pesquisa de trabalho relacionado gira em torno das questões e técnicas necessárias para se entender quais as dificuldades que os utilizadores sentem ao tentarem utilizar uma API e reutilizar as suas funcionalidades. Identificando quais os problemas apontados, é possível conduzir o estudo de forma a que a existência ou ausência destes problemas forneçam a informação necessária para tomar decisões.

Nos relatórios do encontro do grupo de interesse na usabilidade de APIs [DFMS09, DFSM09] afirma-se que McLellan e os colegas [MRTS98] foram os primeiros a elaborar e a reportar um estudo que realça a importância da análise da usabilidade a nível de APIs. Esse estudo resultou num relatório que indicava recomendações para alterar a documentação, os exemplos de código e a API em estudo, de forma a responder às dificuldades de utilização que os participantes sentiram.

Alguns anos depois, Clarke reporta estudos realizados e adotados pela Microsoft [Cla04, Cla06], onde fazem uso das dimensões cognitivas, explicadas na secção seguinte, para analisar problemas que foram observados durante a avaliação da usabilidade em testes de laboratório. Com as dimensões cognitivas, foi possível perceber qual a raiz dos problemas. Por exemplo, o autor reporta [Cla04] que num estudo observaram os participantes com dificuldades na documentação e que a primeira interpretação era que a documentação tinha de ser alterada. No entanto, com a utilização das dimensões cognitivas para a descrição dos problemas, concluíram que os participantes não conseguiam o que queriam, porque o que procuravam não existia.

Joshua Bloch [Blo06], da Google, também é um dos líderes na divulgação da importância e aplicação da usabilidade das APIs, como forma de melhorar a experiência dos programadores. Algumas das suas ideias são, por exemplo, que uma API deve ser fácil de usar e difícil de utilizar incorretamente; que uma API deve ser auto-documentada; e que o desenho de uma API não deve ser uma atividade solitária.

Recentemente, foram realizados alguns progressos no estudo das decisões de desenho de APIs e na identificação dos desafios de desenho, tais como as decisões arquiteturais e da linguagem de uma API, e a utilização de um padrão de desenho [SM07, ESM07], que confirmam, empiricamente, a hipótese de a usabilidade da API ser um problema significativo para todos os programadores [DFSM09].

3.1.1 Metodologias de avaliação da usabilidade de APIs

Analisando os vários artigos que apresentam estudos sobre a usabilidade das APIs [MRTS98, Cla04, Cla06, Rob09, Hen09, ESM07, SC07, ZER11, FWZ10, MC11, XAC09], é possível observar que métodos são aplicados para avaliar a usabilidade de uma API. O método mais comum envolve testes de usabilidade em laboratório, designados por avaliações empíricas [FWZ10].

As avaliações empíricas, realizadas em laboratório, consistem, normalmente, em tarefas que os utilizadores têm de realizar, utilizando uma determinada API ou potencial API. Durante estes testes são recolhidos dados pelos responsáveis que estão a conduzir o teste ou por pessoas cuja função é apenas recolher e registar os dados que vão sendo observados, enquanto outra pessoa conduz o teste. Quando não é possível a presença do utilizador ou dos utilizadores, estes testes são conduzidos via vídeo conferência. Durante estes testes os utilizadores são encorajados a pensarem em voz alta, para ser possível detetar melhor as dificuldades e melhor direcionar o utilizador nas tarefas a desempenhar no teste[MRTS98, FWZ10, XAC09, SCM06]. Estes testes, realizados em laboratório, têm como desvantagens o consumo de muito tempo e de muitos recursos. Normalmente, apenas uma pequena percentagem da API é avaliada, porque os custos não são comportáveis, num contexto de recursos limitados[FWZ10, SCM06].

Outros métodos que procuram complementar o método empírico, são os métodos de inspeção de usabilidade [FWZ10]. A estrutura destes métodos envolve a existência de avaliadores que, durante a inspeção, avaliam vários artefatos, desde os requisitos ao código, sob a perspectiva da usabilidade.

Alguns estudos além de serem realizados através de um dos métodos mencionados, também utilizam uma framework de dimensões cognitivas que é composta por 12 fatores [GP96]. Esta framework é, originalmente, uma técnica de avaliação para dispositivos de interação ou para notações não interativas. Os 12 fatores ou dimensões permitem capturar os aspectos relevantes de uma estrutura. Esta técnica foi adaptada para ser aplicada na avaliação de APIs e as 12 dimensões variam ligeiramente da versão original [Cla04]. As dimensões, adaptadas para avaliação de APIs, são as seguintes:

- Nível de abstração – Níveis máximo e mínimo de abstração exposta pela API, e níveis máximo e mínimo de utilização por parte do utilizador.
- Estilo de aprendizagem – Requisitos de aprendizagem colocados pela API, e estilos de aprendizagem disponíveis para o utilizador.
- *Working Framework* – Dimensão do conjunto de trabalho do programador, necessário para a eficiência de trabalho.
- *Work-Step unit* – Grau de facilidade em completar uma tarefa num único passo.
- Avaliação progressiva – Até que ponto, código parcialmente completo pode ser executado para obtenção de *feedback* no comportamento do código.
- Compromisso prematuro – Número de decisões que um programador tem de tomar ao escrever código para um dado cenário, e quais as consequências dessas decisões.
- Penetrabilidade – Facilidade de exploração, de análise e de compreensão dos componentes da API, e facilidade com que os utilizadores recolhem o que necessitam.
- Elaboração da API – Até que ponto a API tem de ser adaptada para suportar as necessidades dos programadores.
- Viscosidade da API – Barreiras às alterações inerentes à API, e esforço necessário para executar uma alteração.
- Consistência – Facilidade de aprendizagem do resto de uma API, uma vez apreendida uma parte desta.
- Expressividade de papeis – Quão evidente é a relação entre cada componente exposto pela API e o programa, como um todo.
- Correspondência com o domínio – Clareza do mapeamento entre os componentes da API e o domínio, e truques especiais necessários para que o programador complete determinada funcionalidade.

Na perspectiva da avaliação de usabilidade de uma API, estas dimensões representam o impacto da usabilidade na forma como os programadores trabalham com uma API e na forma como esperam que a API funcione. Estas dimensões podem ser utilizadas para a análise dos dados resultantes da aplicação de qualquer um dos métodos mencionados anteriormente. Assim, os resultados dos estudos são descritos em termos das diferentes dimensões, ou seja, cada questão que surge do estudo é descrita em termos da dimensão apropriada. Em alguns casos essa questão pode ser descrita na perspectiva de mais de uma dimensão, o que encoraja a que as equipas considerem várias soluções para a resolução do problema [Cla06].

No artigo de Umer Farooq et al. [FWZ10], os autores propõem um método de inspeção de usabilidade e realizam um estudo em que aplicam o método proposto e o método de avaliação empírica. No final realizam uma comparação entre os dois métodos. O método proposto é constituído por 4 papeis. O responsável pela *feature*, que é quem, tipicamente, escreve as especificações da *feature*, e o seu papel é conduzir a análise. O responsável pelas *features* no geral, tem conhecimento de como uma *feature* particular a ser analisada, se relaciona com as outras *features*. O seu papel é recolher dados de *feedback* durante a análise. O engenheiro de usabilidade, é o responsável por avaliar a usabilidade da API para a *feature*, e o seu papel é moderar a discussão da análise sobre a perspetiva da usabilidade da API. Por fim, os revisores são parceiros organizacionais do responsável pela *feature*. Estes revisores não estão familiarizados com a *feature* a ser analisada mas têm conhecimento necessário para fornecer *feedback*. A análise é processada em 3 fases. Planeamento, Análise e Deteção de Erros.

No mesmo estudo, em ambos os métodos, os dados são analisados e categorizados através das dimensões cognitivas. Os métodos não avaliaram exatamente as mesmas *features* da API, mas *features* da mesma área. Após a análise e categorização de todos os dados, os autores chegaram à conclusão que os métodos identificam problemas em diferentes dimensões cognitivas. O método de inspeção de usabilidade proposto, permitiu detetar problemas relacionados com as dimensões cognitivas, que revelam explicitamente a expectativa do programador em relação à facilidade de aprendizagem de uma API. O método de avaliação empírica permitiu detetar erros de usabilidade relacionados com a correspondência de domínio, que revelam se o programador tem de conhecer detalhes específicos para completar determinadas funcionalidades. No final, concluíram ainda que o método de inspeção proposto e o método de avaliação empírica podem ser usados em conjunto para a avaliação da usabilidade de APIs em diferentes pontos do ciclo de desenho.

Tendo em conta que esta dissertação tem como objetivo realizar um estudo sobre APIs, além de ser necessário perceber que métodos são normalmente utilizados para realizar estudos sobre APIs, é também importante saber que fatores influenciam a usabilidade destas, para se perceber que propriedades têm de ser comparadas entre as APIs em estudo.

3.1.2 Dificuldades de usabilidade reportadas por vários estudos

Quando se pretende utilizar uma nova API, a tendência dos utilizadores é consultar a documentação, de forma a estudar as funcionalidades e até mesmo as limitações que esta oferece. Vários estudos, discutidos de seguida, referem que a consulta da documentação das APIs é bastante frequente por parte dos utilizadores.

Martin P. Robillard [Rob09], perante o facto de a dificuldade de utilização de APIs poder anular os ganhos de produtividade que estas oferecem, este investigador tenta perceber o que é que torna difícil de aprender uma API, desenvolvendo um estudo onde

a informação recolhida é realizada por meio de intervistas aos programadores, muito experientes ou com alguma experiência, da Microsoft, sobre os obstáculos que enfrentaram quando estavam a aprender a utilizar APIs. Neste estudo de 80 respostas do questionário realizado, 78% dos entrevistados afirma que a consulta da documentação é uma meio aprendizagem, levando, assim, o investigador a apontar que os recursos de aprendizagem, como a documentação, são bastante importantes e merecem tanta atenção, no seu desenvolvimento, como os aspectos estruturais das APIs.

Minhaz F Zibran et al. [ZER11], analisaram 562 relatórios de erros relevantes em questões de usabilidade. Mais de um quarto (27,3%) dos relatórios reflectem problemas sobre a documentação, o que indica que, por um lado, existe bastante consulta de documentação e, por outro, que os problemas com a documentação se refletem, de facto, na usabilidade conseguida.

Num artigo onde se tenta perceber quais os critérios que um programador utiliza para a escolha de uma API [XAC09], o estudo revela quais as atividades necessárias para a escolha de uma API. Os autores categorizam estas atividades por tema: pesquisar, fazer e planear. Os aspectos relacionados com o processo de pesquisa são: informação acessível, a documentação e comunidade ativa. Para o tema relacionado com a tentativa de utilização da API (fazer), os aspectos são: a facilidade de aprendizagem, a configuração de ambiente, exemplos, pequenos projetos e nível de abstração. Por fim, o planeamento compreende os seguintes aspectos: manutenção, longevidade, flexibilidade e interoperabilidade. Os resultados do estudo sugerem que a consulta da documentação é um recurso importante para a aprendizagem de uma API, quando o utilizador está na atividade de pesquisa.

Na Microsoft, num estudo executado para determinar se os utilizadores estão aptos a usar a API .Net [Cla06], observou-se que os participantes tiveram muitas dificuldades com a documentação o que os levou a passar grande parte do tempo a procurar, na mesma documentação, algo que lhes mostrasse como completar uma determinada tarefa. Os participantes demoraram 2 horas a completar a tarefa pretendida. Os autores concluíram que a razão para os participantes pesquisarem na documentação e perderem aí muito tempo, se devia ao facto do nível de abstração da API ser muito baixo. Seis meses depois, realizaram um novo estudo em que a API oferecia um maior nível de abstração, e para a realização da mesma tarefa, os participantes deste novo estudo demoraram agora 20 minutos. Este resultado mostra como o nível de abstração da API está relacionado com a procura da documentação para ultrapassar os obstáculos encontrados. Quando a API tinha um nível baixo de abstração, os participantes tinham acesso a mais detalhes da implementação e procuravam por mais esclarecimentos na documentação. Quando a API passou a ter maior nível de abstração, houve evolução na eficiência porque os participantes já teriam acesso a menos detalhes e também não precisavam de consultar tanto a documentação de forma a se esclarecerem, pois assim geravam-se menos dúvidas.

A consulta à documentação dá-se também na tentativa de procurar exemplos de código, o que constitui outra forma de aprendizagem frequente. Em muitas ocasiões, os

utilizadores consultam a documentação para procurar exemplos para uma primeira experimentação e como chave de aprendizagem [Rob09, MRTS98, Cla04, XAC09]. Esta afirmação é suportada pelas seguintes observações: Segundo o estudo realizado no artigo [Rob09], de 80 respostas do questionário realizado, 55% dizem utilizar exemplos como chave de aprendizagem de APIs. No estudo realizado no artigo [MRTS98], os participantes afirmam que a utilização de exemplos de código facilita a compreensão da utilização geral da API. Voltando às atividades necessárias para a escolha de uma API [XAC09], apresentadas anteriormente, o estudo confirma que os utilizadores, na tentativa de utilização de uma API, procuram exemplos de forma a aprenderem e a familiarizarem-se com esta, e a terem uma melhor ideia do que é possível fazer e como.

Apesar dos exemplos de código serem frequentemente uma mais valia, também existem frustrações, e estas estão normalmente relacionadas com o facto de os utilizadores tentarem usar exemplos com propósitos que vão mais além da interação básica com a API ou de não saberem como algumas partes de código se combinam com outras, guiando-se simplesmente pelos exemplos [Rob09].

Por forma a minimizar os obstáculos de aprendizagem, os principais aspetos que a documentação de uma API tem de incluir sãp: a existência de exemplos que ilustrem como utilizar a API, a existência de exemplos de erros típicos que devem ser evitados e devem estar o mais completas possível [MRTS98].

Estas indicações, de como a documentação deve minimizar os obstáculos, podem ser úteis para projetos ainda em desenvolvimento, mas a realidade é que já existem muitas APIs e respetiva documentação, e mesmo com várias décadas de desenvolvimento destas, continuam a existir problemas nas APIs. Uma das causas apontadas para a existência de problemas na documentação, é o facto de esta ser frequentemente escrita depois da API ter sido implementada e essa tarefa ser executada pelo programador que desenvolveu a API.

Quando o programador que desenvolveu a API é a pessoa que escreve a documentação, há tendência a simplificar, assumindo que algumas coisas são óbvias quando na verdade não o são. Esta tendência é um problema bem conhecido que compreende a dificuldade em transferir conhecimento para outra pessoa, através da escrita ou mesmo verbalizando [San05].

Quanto à melhor altura para o desenvolvimento da documentação, é melhor que esta seja realizada antes da implementação, pois através dos ciclos de desenvolvimento da API, que incluem a participação do utilizador final, são tidas em conta as necessidades que o utilizador tem, e assim a API atende às suas necessidades, impedindo que surjam muitas falhas de desenho [Hen09].

Outro fator que tem impacto na usabilidade de uma API, são os nomes usados para funções, variáveis ou classes. Se não existir uma convenção de nomes, e se esta não for consistentemente usada na API, ou se os nomes não forem claros, torna-se difícil a sua utilização [Blo06, Hen09]. Nomes descritivos, por exemplo em CamelCase, são preferíveis a nomes abreviados (por exemplo, notações Hungarian) [ZER11, Zib08].

Steven Clarke [1] realizou um estudo relativo a outro fator que influencia a usabilidade, os atributos que fazem parte da API. Neste estudo, ele descobriu que os programadores têm dificuldades quando a API requer que se configurem vários atributos, de forma a alcançarem funcionalidades específicas, por exemplo, quando uma alteração do comportamento da aplicação em desenvolvimento implica a alteração de mais do que um atributo de uma vez. As razões para este fator ser considerado um obstáculo, são a baixa visibilidade dos atributos envolvidos e a dificuldade em compreender as interações entre atributos.

A questão dos atributos também é discutida ao nível dos construtores. É do conhecimento geral que os construtores parametrizados são normalmente usados para instanciar objetos e atribuir valores iniciais a atributos ao mesmo tempo. No entanto, um estudo de Jeffrey Stylos e Steven Clarke [SC07] permitiu que estes descobrissem que os participantes, programadores experientes, são mais eficientes e preferem utilizar construtores sem parâmetros e métodos de modificação (*setter methods*) para a instanciação de objetos e definição de atributos. O estudo sugere que construtores parametrizados interferem com estratégias de aprendizagem, causando compromisso prematuro indesejado.

Relacionado ainda com os construtores temos o padrão de desenho fábrica [GHJV95] e o estudo realizado em relação à utilização deste padrão [ESM07]. O estudo, que compara a usabilidade do padrão fábrica e de construtores, revelou que obter a instância de uma classe de uma fábrica, muitas vezes impõe dificuldades aos programadores. Os resultados indicam que os programadores necessitam significativamente de mais tempo para construir um objeto com uma fábrica do que com um construtor.

Por fim, a escolha imprópria de tipos de dados também pode causar trabalho extra aos programadores, pois podem necessitar de fazer *casting* de variáveis ou atributos, o que é inconveniente [ESM07]. Por exemplo, no artigo de Joshua Bloch [Blo06], onde existem várias indicações de como desenhar uma boa API, é sugerido que se evite a utilização de *strings* se existir um tipo melhor.

Em resumo, os métodos de avaliação da usabilidade mais frequentemente encontrados são os métodos empíricos e os de inspeção. Estes podem ser complementados com a utilização da *framework* de dimensões cognitivas para a análise dos dados. Aplicam-se estes métodos de forma a recolher dados da usabilidade das APIs, e envolvem recursos humanos para desempenhar os papéis de mediador ou só de observador, no caso em que só recolhe dados, e participantes que representam a amostra de utilizadores reais.

3.1.3 Métricas de avaliação

Quando não é possível ter acesso aos recursos necessários para realizar a avaliação da usabilidade através de métodos empíricos ou de inspeção, a avaliação pode realizar-se através de métricas, um método que não requer mediadores nem participantes, mas que requer a construção do suporte de ferramentas, para que depois se recolham os dados.

[1] http://blogs.msdn.com/stevencl/archive/2004/05/12/130826.aspx, http://blogs.msdn.com/stevencl/archive/2004/10/08/239833.aspx

Após a recolha dos dados, é ainda necessário analisar os mesmos.

Uma métrica consiste na medição de uma propriedade de um artefato de software ou das suas especificações. As propriedades avaliadas são requisitos não funcionais como a reusabilidade, a adaptabilidade, a coesão, o acoplamento, etc.

A avaliação da reusabilidade permite avaliar a facilidade de utilização pelos programadores, pois quando um programador usa uma API, está a reutilizar as funcionalidades oferecidas pela API. Portanto, é interessante explorar as métricas existentes para avaliação da facilidade de reutilização.

A avaliação da coesão e do acoplamento interno de uma API podem ser avaliados se tivermos acesso ao código [WYF03, RD05]. No entanto, as duas propriedades podem ser avaliadas a um nível de abstração mais alto, considerando apenas o acesso à interface.

Várias métricas foram definidas para componentes de software. Em geral, as métricas para componentes incidem sobre as suas interfaces, ou seja, na prática, sobre as suas APIs (quer a oferecida, a API no sentido mais tradicional, quer na requerida, a API que o componente requer para se poder integrar numa aplicação).

Num estudo sobre várias propostas de métricas sobre componentes de software [GA04], por Goulão e Abreu, foi realizada uma avaliação de maturidade das mesmas. O estudo revela que existia falta de maturidade dessas várias propostas, e que, entre outros motivos, na altura, esta se devia ao facto deste tema de pesquisa ser relativamente recente. No entanto, este estudo já tem oito anos e o benefício da dúvida atribuído à relativa novidade do tema de pesquisa, como razão para a falta de maturidade, já não se aplica, pois no geral não são conhecidos desenvolvimentos relativos às métricas e estas continuam portanto com falta de maturidade. Recentemente (2009), Inoue e os seus colegas [IYY+05, IKMF09], conseguiram patentear uma proposta de métrica, para reusabilidade entre componentes, baseada num ranking produzido pelo sistema que estes desenvolveram, o SPARS-J.

Com a avaliação através de métricas é difícil avaliar a documentação, pois esta é orientada para uma audiência humana. No entanto, uma API pode ser analisada por um computador, permitindo medir propriedades desta. Para entender como as interfaces afetam a reusabilidade, é importante conseguir medir as suas propriedades [BA04].

Para a avaliação em termos de reusabilidade existem várias propostas de conjuntos de métricas. As propostas de métricas que se seguem, propõem avaliações da complexidade da interface da API, ou seja, uma avaliação de características respeitantes à estrutura da API em si (por exemplo, número de atributos e número de métodos).

Rotaru et al. [RD05], propõem métricas de avaliação da reusabilidade de componentes, com base nas propriedades de facilidade de composição e facilidade de adaptação. Estas propriedades podem ser medidas estudando a interface.

Os autores explicam que o grau de facilidade de composição de um componente pode ser qualitativamente definido, estudando os parâmetros e valores de retorno dos métodos da sua interface. Se a interface é constituída por métodos que não têm nem parâmetros, nem valores de retorno, então tem o maior grau de facilidade de composição. Caso os

métodos não tenham parâmetros mas tenham valor de retorno então o grau de facilidade de composição já é mais baixo. O grau mais baixo de todos, acontece quando os métodos têm tanto parâmetros, como valor de retorno. A multiplicidade de um componente é definida pela soma das multiplicidades dos métodos da sua interface. A multiplicidade de um método é definida pela soma da sua multiplicidade de retorno, que toma o valor 0 se não tem valor de retorno ou toma valor 1 se tiver valor de retorno, com a multiplicidade de assinatura. Esta última por sua vez, é definida pela soma da multiplicidade dos argumentos dos métodos, que toma valor 1 se corresponder a um valor ou um valor r>1 se for uma referência. Assim, o grau de facilidade de composição de um componente é inversamente proporcional à sua multiplicidade, ou seja, quando a multiplicidade tende para infinito, a facilidade de composição tende para zero.

Um componente de fácil adaptação deve ser robusto o suficiente de forma a tolerar alterações sem intervenção externa. No entanto, quando a facilidade de adaptação é muito elevada, a complexidade também aumenta. A facilidade de adaptação de um componente de uma *framework* é a soma da facilidade de adaptação do componente com a facilidade de adaptação da *framework*.

Portanto, considerando uma perspectiva qualitativa, a facilidade de adaptação de um componente depende da sua complexidade e os autores afirmam que neste caso um grande nível de reusabilidade também aumenta a adaptabilidade.

Como conclusão, os autores revelam que a facilidade de composição e de adaptação, juntos, permitem medir a reusabilidade no contexto dos componentes.

Estas métricas propostas pelos autores não foram validadas com a avaliação de vários componentes.

Além das propriedades acabadas de referir, e que afetam a reusabilidade no contexto de interação entre componentes, as propriedades que afetam a compreensão são especialmente importantes para a análise de reusabilidade, no contexto da interação do utilizador com a API.

Boxall et al. [BA04], propõem um conjunto de métricas que permitem avaliar propriedades que podem afetar a compreensão, baseando-se na interface dos componentes. Estas métricas permitem medir propriedades como a dimensão média de uma interface, a consistência de nomes e tipos de argumentos, o seguimento de um padrão ao nomear identificadores e o peso médio do comprimento dos nomes dos identificadores. A dimensão de uma interface influência o nível de reusabilidade que esta tem, pois quanto maior é, maior é o esforço para compreendê-la. A consistência entre nomes é um fator que afeta a facilidade de compreensão de uma interface, pois caso esta propriedade exista o utilizador tem maior facilidade em compreender a interface através da transferência de conhecimento, que facilita o reconhecimento dos nomes que estão associados ao mesmo tipo de funcionalidade.

Relacionada com propriedades referentes ao nomes está também a abordagem que é adotada. Quanto mais sistemática, maior facilidade terá o utilizador em compreender a interface. Por exemplo, se encontramos vários métodos denominados set*Nome1*(val),

set*Nome2*(val), set*NomeN*(val), e se entendermos para que serve o primeiro set com que nos deparamos, quando encontrarmos os outros é possível relacionar com o que já aprendemos, e supôr que todos têm o mesmo tipo de funcionalidade.

Por fim, a dimensão que os nomes têm também influência a compreensão, pois nomes de comprimento maior tendem a conter mais informação do que nomes mais pequenos. Assim, interfaces com nomes de maior comprimento terão tendência a serem mais auto-documentadas.

Teoricamente, aplicando estas métricas a várias APIs para as quais exista uma avaliação exterior de reusabilidade (por exemplo, baseada na opinião de peritos) é possível determinar estatisticamente que valores tendem a garantir uma melhor reusabilidade por parte do utilizador. Na prática, este conjunto de métricas não foi avaliado experimentalmente pelos seus proponentes (ou, se tal avaliação ocorreu, não foi reportada) e não são conhecidos estudos posteriores para a sua validação.

As propostas mencionadas anteriormente permitem a avaliação de propriedades das APIs, o que permitirá compará-las e obter alguma informação sobre cada uma delas. Mas também seria interessante poder avaliá-las durante a utilização efetiva, ou seja, ter acesso a software com reutilização de funcionalidades disponibilizadas pelas APIs, e avaliar propriedades dessa reutilização. Esta avaliação normalmente é realizada com base em exemplos retirados de repositórios.

Sarkar et al. [SKR08], propõem uma métrica para calcular o índice de interação de módulo, uma métrica para avaliar a coesão de uma API e métricas baseadas na dimensão. Este estudo é realizado com base em vários módulos de software orientado a objetos, onde cada módulo pode processar múltiplas APIs. A métrica para calcular o índice de interação de um módulo, mede a frequência com que um método é utilizado pelos outros módulos no sistema. A coesão é medida do ponto de vista da utilização das funcionalidades fornecidas pela APIs dos módulos, por outros módulos. Uma API é considerada coesa se ao se reutilizarem as suas funcionalidades, a maioria destas são efectivamente utilizadas. As métricas de dimensão medem um índice de variabilidade do número de classes nos módulos, um índice de variabilidade da dimensão de uma classe, pela contagem do número de métodos, e um índice da variabilidade da dimensão de uma classe pela contagem do número de linhas de código da classe.

Neste artigo, os autores validaram as métricas com oito sistemas de software e revelam que os valores das métricas confirmam os atributos recolhidos a partir de análise manual do software.

3.2 Resumo e análise crítica do estado da arte

Com a informação recolhida sobre o estado da arte, é possível concluir de imediato que ainda existe muito trabalho a desenvolver na investigação da avaliação da usabilidade das APIs. Quanto ao trabalho já desenvolvido, existem bastantes indicações sobre como conduzir e realizar um estudo desta natureza.

A recolha de dados para estes estudos pode ser realizada através de métodos empíricos e métodos de inspeção de usabilidade, em que estes podem ser complementados com a aplicação das dimensões cognitivas.

As técnicas aplicadas para a recolha de informação, durante um estudo que envolve o utilizador final, requerem alguns recursos e bastante tempo de recolha de informação. Tendo em conta que o estudo a ser realizado, é no contexto de uma dissertação com tempo limitado para elaboração e sem uma amostra de utilizadores que possa ser estudada, não será possível aplicar métodos empíricos nem métodos de inspeção de usabilidade que envolvam uma amostra de utilizadores.

Os vários artigos analisados consistiam sobretudo em estudos aplicados a APIs da Microsoft [Cla04, Cla06, Rob09, Hen09, FWZ10], não tendo sido encontrados estudos sobre a API de mapas que a Microsoft disponibiliza (Bing maps). Até à data da entrega deste documento, as pesquisas não revelaram a existência de publicação de algum estudo sobre o tipo das APIs a analisar.

Contudo, apesar de não ser possível aplicar as técnicas que os vários autores aplicaram, e das APIs estudadas por esses autores não serem da categoria das APIs a analisar, é possível perceber que problemas estas técnicas permitem detetar.

Os problemas identificados e com mais informação reportada, centram-se na documentação. Esta é utilizada como passo inicial de aprendizagem, sendo procurada para o estudo da API em si através de análise e leitura, ou pela busca de exemplos de código que ajudam a compreender melhor como se pode utilizá-la. Portanto, é importante que a documentação inclua bons exemplos e informem o utilizador de forma completa, ou seja, também deve ser incluída informação sobre o seu comportamento da API e erros. Num dos artigos é reportado que os problemas a nível da documentação e da dificuldade de aprendizagem através desta, se devem ao facto de, muitas vezes, esta ser escrita por quem desenvolveu a API o que revela uma tendência de simplificação. É ainda defendido que o desenvolvimento da documentação deve ser realizado antes da implementação, porque através dos ciclos que o desenvolvimento impõe as necessidades do utilizador são tidas em conta.

No caso da documentação, por ser orientada para uma audiência humana, não é prático aplicar métricas de avaliação.

Outro problema identificado está relacionado com os identificadores utilizados. No caso de existir uma convenção para a atribuição de identificadores, se esta não for consistentemente usada na API, ou se os identificadores não forem claros, torna-se difícil a utilização da API. Neste caso, já será possível executar uma avaliação baseada em algumas métricas que serão mencionadas mais à frente.

A configuração de vários atributos, de forma a alcançarem funcionalidades específicas também é um dos problemas identificados, pois quando estão envolvidos muitos atributos, estes têm baixa visibilidade e torna-se difícil compreender as interações entre estes. Quando se realizar a recolha de informação das APIs será possível recolher informação sobre a quantidade de atributos e realizar comparações entre as APIs, sobre esta

questão.

Ainda relativamente aos atributos, alguns artigos (e.g [SC07, ESM07]) reportam estudos sobre preferências de programadores, bem como sobre a eficiência de programadores, no que diz respeito à utilização de construtores. Verifica-se que, em geral, os programadores preferem usar construtores sem parâmetros, acompanhados por métodos modificadores (*setter methods*) para a instanciação e configuração de objetos. Considera-se também que os programadores são mais eficientes ao utilizarem construtores para a instanciação de objetos, em vez desta instanciação ser realizada através do padrão de desenho de fábricas [ESM07].

Por fim, também é reportado um problema que reside na escolha imprópria de tipos de dados que pode causar trabalho extra aos programadores, pois obriga ao casting de variáveis ou atributos, o que é inconveniente.

Com a utilização de métricas é possível medir questões do mesmo tipo das medições realizadas com os outros métodos mencionados anteriormente. Por exemplo, é possível medir questões relacionadas com os nomes.

Algumas das métricas pesquisadas, e que serão utilizadas, permitem avaliar propriedades que podem afetar a compreensão, baseando-se na interface dos componentes. Estas permitem medir propriedades como a dimensão média de uma interface, a consistência de nomes e tipos de argumentos, o seguimento de um padrão ao nomear identificadores e o peso médio do comprimento dos nomes dos identificadores. Aplicando estas métricas às APIs em estudo, é possível determinar qual tende a permitir melhor reusabilidade por parte do utilizador.

Um dos estudos apresentados no trabalho relacionado [RD05] referia que a avaliação da reusabilidade de componentes pode ser realizada através de métricas, com base nas propriedades de facilidade de composição e facilidade de adaptação. No entanto, a reusabilidade avaliada através destas métricas, não é a reusabilidade na perspectiva do utilizador da API mas sim a reusabilidade no contexto da interação entre os componentes, logo estas métricas não serão utilizadas. Assim como o estudo acabado de referir, também a métrica patenteada por Inoue e os colegas [IYY+05, IKMF09] não é aplicável pela mesma razão, esta avalia a reusabilidade no contexto de interação entre componentes e não na perspectiva do utilizador da API.

Finalmente, apresentaram-se outras métricas que permitem avaliar a utilização efetiva das APIs através do cálculo do índice de interação de módulo, avaliar a coesão de uma API e métricas de medição do número e da dimensão das classes. De acordo com a estrutura definida para o estudo apenas será possível e interessante utilizar a métrica de índice de interação de módulo, que como avalia numa perspectiva de utilização efetiva entre módulos e não utilização efetiva na perspetiva do utilizador da API, servirá como guia para a definição de uma métricas mais adequada ao estudo.

Portanto, apesar de não ser possível aplicar os métodos empíricos e de inspeção de usabilidade, é possível concluir que, através da utilização de métricas será possível recolher dados relevantes para a compreensão das APIs e efetuar comparações entre as

mesmas.

4

Abordagem seguida para realização do estudo

Neste capítulo apresentamos a abordagem tomada para a condução do estudo de comparação entre APIs de mapas. Na secção 4.1, abordamos os elementos que serão utilizados como base do estudo. Na secção 4.2, apresentamos em as métricas de avaliação para aplicar aos elementos de estudo. Por fim, na secção 4.3, apresentamos a abordagem tomada para a recolha dos dados.

4.1 Elementos base do estudo

4.1.1 Protótipos

Tendo em conta que o objetivo do trabalho é realizar um estudo comparativo entre APIs, da perspetiva do utilizador, desenvolvemos três protótipos em JavaScript, cada um com uma das APIs selecionadas, iguais ao nível das funcionalidades suportadas. Desta forma estes protótipos permitem avaliar as características das APIs em condições semelhantes.

Considerou-se que as funcionalidades base de uma aplicação deste domínio são: zoom, full extent, pan, controladores, *overview map*, entidades georreferenciadas, informação associada a entidades e pesquisa de localizações. De forma a confirmar se as funcionalidades base consideradas estão realmente presentes em aplicações desta natureza, desenvolveu-se uma pesquisa sobre aplicações implementadas com a utilização das APIs em questão e com contexto semelhante ao que se pretende implementar.

As aplicações utilizadas para fazer esta pesquisa foram selecionadas dos seguintes sites: Google Maps: 100+ Best Tools and Mashups[1], OpenLayers gallery[2], ArcGIS Resource Center code gallery[3].

A tabela 4.1 apresenta as aplicações analisadas e respetivas funcionalidades, seguida de uma breve descrição de cada aplicação e de cada funcionalidade. O "X" significa que a aplicação tem a funcionalidade, o "-" significa que não tem.

	Google						ArcGIS		OpenLayers	
	Taxicab zones	Wines and times	HealthMap	Theatre in Chicago	SafeFood Finder	I Need A Bike	Property information	Place matters	PegelOnline	Real Estate
Zoom in/out	X	X	X	X	X	X	X	X	X	X
Pan	X	X	X	X	X	X	X	X	X	X
Overview Map	-	X	-	-	X	-	X	-	X	-
Controladores	X	X	X	X	X	X	X	X	X	X
Pesquisa de localizações	X	-	X	X	X	X	X	X	-	X
Visualização de entidades georreferenciadas	X	X	X	X	X	X	X	X	X	X
Visualização de informação associada às entidades	X	X	X	X	X	X	X	X	X	X
Full extent	-	X	-	-	-	-	-	X	-	-

Tabela 4.1: Aplicações vs. Funcionalidades

Descrição das aplicações:

- Washington, DC Taxicab Zones[4] - Mapa de zonas que cobre cada zona de táxis de Washington.
- Wines and Times[5] - Aplicação que permite planear um viagem de vinhos nos Estados Unidos.
- HealthMap[6] - Aplicação com distribuição de casos de doenças a nível mundial.

[1] http://mashable.com/2009/01/08/google-maps-mashups-tools/
[2] http://trac.osgeo.org/openlayers/wiki/Gallery
[3] http://help.arcgis.com/en/webapi/javascript/arcgis/help/gallery.html
[4] http://maps.google.com/maps/ms?msa=0&msid=117880727426175971427.00043db9054be42f0dc8f&ie=UTF8&z=11&om=0
[5] http://winesandtimes.com/wnt/index.php
[6] http://www.healthmap.org/en/

- Theatre In Chicago [7] - Aplicação que permite saber que peças estão no momento no teatro e quais as que vão estar.
- SafeFoodFinder[8] - Aplicação que apresenta o mapa dos restaurantes de Seattle e as pontuações que lhes foram atribuídas pelo Departamento de Inspeção.
- I Need A Bike[9] - Aplicação que permite encontrar locais para aluguer de bicicletas em torno de Paris, Lyon, e Marseille.
- PegelOnline[10] -Aplicação sobre dados relacionados com lagos na Alemanha.
- Larson Group Real Estate[11] - Aplicação que permite procurar anúncios de imóveis em Brainerd, MN.
- San Francisco Property Information Map[12] - Aplicação que permite visualizar relatórios de propriedades imobiliárias, entre outros dados.
- New York City: Place Matters[13] - Aplicação que permite consultar informação sobre pontos de interesse na cidade de Nova Iorque.

Descrição das funcionalidades típicas que se encontram nas aplicações:

- *Zoom in/out* – Ferramentas que permitem controlar o detalhe da visualização do mapa;
- *Full extent* – Caso particular de *zoom in/out*, que permite a adaptação dinâmica do nível de ampliação do mapa, de forma a incluir todas as localizações contidas no conjunto de dados utilizado no momento;
- *Pan* – Característica que permite que se modifique o foco do mapa, o que pode levar a alterar a área a visualizar para uma zona não visível inicialmente, arrastando-o;
- Controladores – Ferramentas que permitem que se modifique o foco do mapa, o que pode levar a alterar a área a visualizar para uma zona não visível inicialmente, selecionando-as;
- *Overview Map* – Integração de um mapa de contexto na janela de visualização, o que permite identificar a zona visível no mapa principal, dentro de uma área geográfica de maior dimensão;
- Entidades georreferenciadas – entidades pontuais, lineares ou poligonais, vinculadas a coordenadas geográficas (latitude e longitude);

[7] http://www.theatreinchicago.com/maps/mapTheatres.php
[8] http://www.safefoodfinder.com/
[9] http://ineedabike.gmapify.fr/
[10] http://www.pegelonline.wsv.de/gast/karte/standard;jsessionid=BB10C864C291AD0E96C51FC4A4C74B62
[11] http://www.larsongrouprealestate.com/index.php/map-search
[12] http://ec2-50-17-237-182.compute-1.amazonaws.com/PIM/
[13] http://mapstories.esri.com/placematters/

- Informação associada a entidades - Janelas associadas a entidades geográficas do mapa e que fornecem informação adicional relativa a estas entidades (por exemplo: fotos do local ou descrições);
- Pesquisa de localizações – Capacidades de georreferenciação que consistem na conversão de endereços de moradas para coordenadas geográficas (*geocoding*) ou, o contrário, coordenadas geográficas para endereços de morada (*reverse geocoding*).

De forma a incorporar as funcionalidades identificadas, os protótipos implementados representam uma aplicação orientada ao turismo em que se visualizam museus e construções de valor histórico como entidades georreferenciadas representadas através de marcadores ou polígonos. Estas entidades têm informação associada que pode ser visualizada, e o utilizador pode interagir com a aplicação, introduzindo localizações para obter direções, pesquisando sobre a lista de pontos de interesse, ou utilizando as ferramentas que permitem controlar o detalhe da visualização e movimentar-se dentro do mapa. É ainda possível modificar o foco do mapa, adaptar dinamicamente a ampliação ao conjunto de dados utilizado no momento, consultar passeios recomendados que permitem uma caminhada com visita de pontos de interesse (representados sob a forma de linhas) e visualizar polígonos que representam a área de alguns pontos de interesse. Os pontos de interesse utilizados para efeitos de teste, representam pontos de interesse de Portugal, mais precisamente da cidade de Lisboa. No entanto, fazendo alterações aos dados carregados a mesma aplicação poderia funcionar noutra localização desejada.

4.1.2 APIs

Além dos protótipos, as próprias APIs, implementadas em JavaScript, permitem também extrair alguma informação para comparação entre elas. O que permite complementar as observações retiradas a partir dos protótipos. Além da comparação entre as APIs utilizadas para o desenvolvimento dos protótipos, foi ainda possível comparar várias versões de cada API, analisando a evolução entre versões.

4.1.3 Conjunto de aplicações suportadas pela a API da Google

Como referido na introdução, na secção 1.3, graças à disponibilidade de um conjunto de aplicações suportadas pela Google Maps JavaScript API, e submetidas como trabalhos no contexto da disciplina de Tecnologias de Informação Geográfica, do Mestrado em Engenharia Informática da FCT/UNL, foi possível desenvolver uma análise da utilização efetiva da mesma API. Os autores deste conjunto de 10 aplicações, escolheram o tema do trabalho a desenvolver, que inclui os seguintes requisitos mínimos:

- Visualização no mapa de entidades georreferenciadas, disponibilizadas de forma persistente, seja através de sistemas de arquivos existentes (XML ou KML, residentes no servidor), ou através de bases de dados;

- Visualização da informação associada às entidades georreferenciadas no mapa, utilizando, por exemplo, janela de informação incluindo diversos componentes de informação;
- Legenda das categorias mapas visualizadas atualizada, dependendo dos dados carregados e categorias, num dado momento;
- Inserção, atualização e eliminação de entidades espaciais (incluindo as suas localizações espaciais) e informações associadas, de forma persistente (permitindo, na inserção e atualização, a edição das informações a associar a cada local), usando a interface do utilizador;
- Zoom in/Zoom out e pan (alteração do foco/centro da janela do mapa);
- Full extent (para adaptação dinâmica do nível de zoom da janela do mapa, para incluir todos os locais, visto num determinado momento, dependendo das categorias selecionadas);
- Importação automática de novas camadas de dados XML ou KML, incluindo a respetiva legenda, tanto para o sistema de arquivos do servidor ou para a base de dados. O processo de importação deve incluir a atualização da janela do mapa;
- Utilização de serviços da Google, de acordo com as necessidades do tópico da aplicação (por exemplo: georreferenciação).

4.2 Métricas de avaliação

Uma vez que este estudo não envolve testes empíricos em laboratório, com o auxílio de utilizadores reais, nem métodos de inspeção de usabilidade, a avaliação da usabilidade é realizada aplicando métricas que permitam avaliar a mesma, sobre os dados recolhidos das APIs e dos protótipos.

Como se pode observar no capítulo 3, a pesquisa de trabalho relacionado inclui métricas existentes para a medição de propriedades relevantes para a reutilização, pois um programador, ao utilizar uma API, está a reutilizar as funcionalidades oferecidas por esta.

Entre as métricas apresentadas nesse capítulo selecionaram-se algumas métricas que permitem medir propriedades que podem afetar a compreensão e reutilização de interfaces, como a necessidade de configuração de vários atributos, a falta de consistência entre os nomes e tipos dos argumentos, a falta de consistência na nomeação dos métodos e a falta de auto-documentação por parte das interfaces [BA04]. Medindo estas propriedades e analisando estatisticamente os resultados obtidos é possível avaliar se alguma destas APIs é mais fácil de compreender ou utilizar que as outras.

Para a apresentação das métricas utilizadas para a avaliação de usabilidade, utilizamos a abordagem GQM (*Goal-Question-Metric*) que consiste na definição de um conjunto de métricas nas quais as propriedades de um sistema podem ser medidas.

A metodologia GQM permite a monotorização e medição do desempenho de atividades de software, e foi desenvolvida por Basili e pela sua equipa do *NASA Software Engineering Laboratory*[BCR94]. Este paradigma é uma abordagem orientada a objetivos para medições de sistemas de desenvolvimento de software que prevê três níveis de abstração:

- **Nível 1.** Nível Conceptual (Objetivo) - identificar objetivos de medição;
- **Nível 2.** Nível Operacional (Questão) - propôr questões para os objetivos de medição;
- **Nível 3.** Nível Quantitativo (Métrica) - definir métricas que ajudem a responder às perguntas colocadas.

A possibilidade que a abordagem GQM fornece em definir questões abstratas e responder com o auxílio de métricas, a fim de avaliar a qualidade, traz um enorme interesse para o âmbito desta dissertação, que visa, exatamente, avaliar alguns atributos de qualidade. Para além disto, é uma abordagem bem conceituada na comunidade, sendo aconselhada pelo próprio *Software Engineering Standards Committee da IEEE Computer Society* como uma *framework* adequada para estabelecer métricas ao nível organizacional [iee98].

O objetivo é analisar a facilidade de compreensão, a utilização efetiva das APIs em estudo e caraterizar a evolução das APIs. Portanto, seguindo a abordagem GQM, o nível conceptual é composto pelos objetivos de analisar a facilidade de compreensão e a utilização efetiva das APIs modelo.

O processo de construção das métricas de complexidade propostas nesta secção foi inspirado por métricas de complexidade de software propostas noutros contextos [BA04, SKR08].

As Tabelas 4.2, 4.3 e 4.4 sintetizam o resultado da aplicação da abordagem GQM para propor um conjunto de métricas que permitam satisfazer os objetivos de avaliação da facilidade de compreensão, da utilização efetiva das APIs modelo e da caracterização da evolução das APIs.

Na Tabela 4.2 temos o modelo GQM para alcançar a avaliação da facilidade de compreensão das APIs. Em resumo, teve-se em conta os identificadores no geral e os argumentos dos métodos das APIs, pois é com estes elementos que o programador interage diretamente.

Tabela 4.2: GQM para a avaliação da facilidade de compreensão da API

Objetivo	Avaliar a facilidade de compreensão da API.
Pergunta	**Métrica**
P1. Qual a dimensão média dos métodos da API, relacionada com os seus argumentos?	**M1.** Argumentos por procedimento.
P2. Será que existe tendência a seguir um padrão na nomeação dos identificadores?	**M2.** Média de semelhança entre *strings*.
P3. Quão auto-documentada é a API?	**M3.** Comprimento médio de identificadores.

Na Tabela 4.3 temos o modelo GQM para alcançar a avaliação da utilização efetiva das APIs. Para este modelo tem-se em conta os protótipos desenvolvidos precisamente para a avaliação da utilização efetiva das APIs, bem como as respetivas APIs modelo.

Tabela 4.3: GQM para a avaliação da utilização efetiva da API

Objetivo	Avaliar a utilização efetiva da API.
Pergunta	**Métrica**
P4. Quão complexa é a implementação das funcionalidades dos protótipos?	**M4.** Índice de utilização da API.
P5. Qual é a frequência de utilização das funcionalidades da API?	**M5.** Frequência de utilização de funcionalidades.

Na Tabela 4.4 temos o modelo GQM para alcançar a caracterização da evolução das APIs. Para este modelo tem-se em conta várias versões de cada uma das APIs estudadas. Além das várias versões das APIs, aplica-se a métrica de caracterização da dimensão da API a cada uma das APIs estudadas, de forma a ser possível comparar as dimensões das APIs.

Tabela 4.4: GQM para a caracterização da evolução das APIs

Objetivo	Caracterizar a evolução das APIs
Pergunta	**Métrica**
P6. Qual a dimensão da API?	**M6.** Número de objetos, métodos e propriedades.
P7. Quantos objetos foram eliminados?	**M7.** Número de objetos eliminados.
P8. Quantos objetos foram adicionados?	**M8.** Número de objetos adicionados.
P9. Quantos objetos foram mantidos?	**M9.** Número de objetos mantidos.

4.2.1 Métricas para avaliação da facilidade de compreensão das APIs

Sabe-se que a capacidade humana para reter uma quantidade pequena de informação é limitada. Apenas cerca de sete itens podem ser armazenados de cada vez [Mil56]. Assim, sabemos que um número de argumentos superior a sete por método é um fator que poderá afetar a compreensão da API, pois é difícil estar sempre a recordar quais os argumentos necessários, apesar de se poder argumentar que com mais argumentos, aumenta a expressividade. No entanto, isto apenas serve para dar uma ideia do limite aceitável, pois o que interessa medir é se alguma das APIs tem em média menos argumentos por método, o que indicará que é mais fácil de compreender. A métrica APP na Tabela 4.5 permite medir este valor.

Tabela 4.5: Métricas que satisfazem a avaliação da facilidade de compreensão da pergunta P1

P1 - Qual a dimensão média dos métodos da API, relacionada com o seus argumentos?	
Nome	APP - Arguments Per Procedure (Argumentos por procedimento)
Definição informal	Número médio de argumentos por procedimento.
Definição formal	$\frac{n_a}{n_p}$ Onde n_a é a contagem total de argumentos dos métodos e n_p é a contagem total de procedimentos, em que os métodos *object-oriented* são tratados como procedimentos.
Comentários	Um valor baixo de APP indica que a interface será mais fácil de entender.

A abordagem tomada para a nomeação de identificadores também é um importante fator relativamente à facilidade de compreensão. Quanto mais sistemática, maior facilidade terá o utilizador em compreender a interface. A métrica MSC na Tabela 4.6 permite medir este fator.

Tabela 4.6: Métricas que satisfazem a avaliação da facilidade de compreensão da pergunta P2

P2 - Será que existe tendência a seguir um padrão para nomeação de identificadores?	
Nome	MSC - Mean String Commonality (Média de Semelhança entre *Strings*)
Definição informal	Valor médio de semelhança entre *strings*.
Definição formal	$\frac{\Sigma(x,y)\in comb(A)\frac{\lvert lcs(x,y)\rvert}{\max(\lvert x\rvert,\lvert y\rvert)}}{n^{C_2}}$ Onde A é o conjunto de identificadores, n é a contagem total de identificadores, *comb(A)* calcula as combinações do conjunto de identificadores, *lcs(x,y)* calcula a maior sequência comum de x e y e *max(*$\lvert x \rvert, \lvert y \rvert$*)* retorna o tamanho da *string* maior.
Comentários	Se o conjunto de identificadores tiver zero ou um elemento, o MSC é indefinido. Uma interface com maior valor de MSC tem uma abordagem de nomes mais sistemática.

A dimensão dos nomes também influencia a compreensão, pois nomes de comprimento maior tendem a conter mais informação do que nomes mais pequenos. Assim, interfaces com nomes de maior comprimento serão tendencialmente melhor auto-documentadas. A métrica MIL na Tabela 4.7 permite medir este fator.

Tabela 4.7: Métricas que satisfazem a avaliação da facilidade de compreensão da pergunta P3

P3 - Quão auto-documentada é a API?	
Nome	MIL - Mean Identifier Length (Comprimento Médio de Identificadores)
Definição informal	Valor médio do comprimento dos identificadores.
Definição formal	$\frac{\Sigma(x)\in A, length(x)}{n}$ Onde A é o conjunto de identificadores, n é a contagem total de identificadores e *lenght(x)* calcula o tamanho do identificador.
Comentários	Interfaces com maior valor de MIL são mais auto-documentadas.

4.2.2 Métricas para avaliação da facilidade de utilização das APIs

No que diz respeito às métricas para avaliação da utilização efetiva das APIs, temos a métrica APIUI na Tabela 4.8. Tendo em conta que os três protótipos implementam as mesmas funcionalidades, esta métrica permite medir quantas chamadas a construtores, chamadas a funções e chamadas a propriedades da API, cada funcionalidade necessita,

em cada implementação.

Tabela 4.8: Métricas que satisfazem a avaliação da utilização efetiva da pergunta P4

P4 - Quão complexa é a implementação das funcionalidades dos protótipos?	
Nome	APIUI - API Usage Index (Índice de Utilização da API)
Definição informal	Número de chamadas à API.
Definição formal	$C + F + P$ Onde *C* é o total de chamadas a construtores, *F* é o total de chamadas a funções e *P* é o total de chamadas a propriedades.

Para o estudo particular para a API da Google é interessante medir a frequência de utilização de um objeto, no contexto de todas as aplicações realizadas com esta API, pois assim é possível perceber que objetos são imprescindíveis para o desenvolvimento de uma aplicação com os mesmos requisitos destas aplicações. A métrica Frequência de Utilização da API, na Tabela 4.9, permite medir a frequência de utilização de um objeto, nas aplicações.

Tabela 4.9: Métricas que satisfazem a avaliação da facilidade de compreensão da pergunta P5

P5 - Qual a frequência de utilização da API?	
Nome	Frequência de Utilização da API
Definição informal	Número de vezes que um objeto é utilizado.
Definição formal	$count(X_i)$ Onde X_i são os diferentes objetos utilizados em todos os trabalhos, e $count(X_i)$ é o número de vezes que cada objeto se repete.

4.2.3 Métricas para caracterização da evolução das APIs

De acordo com as leis de evolução do software [Leh78, Leh79] à medida que um sistema evolui, a sua complexidade aumenta e o conteúdo funcional de um sistema deve aumentar continuamente. O aumento da complexibilidade e do conteúdo funcional pode ser avaliado medindo o número de objetos, de métodos e de propriedades entre versões. A métrica na tabela 4.10 permite medir a dimensão das APIs.

Tabela 4.10: Métricas que satisfazem a caracterização da evolução da pergunta P6

P6 - Qual a dimensão da API?	
Nome	Número de objetos, métodos e propriedades
Definição	Número total de objetos, métodos e propriedades na API.
Comentários	Variações neste número sugerem variações na complexidade da API.

A instabilidade tem impacto na utilização da API, porque os seus utilizadores têm de se atualizar. Através das métricas nas tabelas 4.11, 4.12 e 4.13, é possível medir quantos objetos são mantidos, removidos e adicionados, entre versões e assim analisar a instabilidade.

Tabela 4.11: Métricas que satisfazem a caracterização da evolução da pergunta P7

P7 - Quantos objetos foram eliminados?	
Nome	Número de objetos eliminados.
Definição	Número total de objetos eliminados.
Comentários	A eliminação de objetos leva a uma quebra na compatibilidade das aplicações que utilizam versões anteriores, contribuindo para o aumento dos custos de evolução dessas aplicações.

Tabela 4.12: Métricas que satisfazem a caracterização da evolução da pergunta P8

P8 - Quantos objetos foram adicionados?	
Nome	Número de objetos adicionados.
Definição	Número total de objetos adicionados.
Comentários	A adição de objetos pode ser associada a novas funcionalidades oferecidas pela API, e pode ser considerado como um indicador de instabilidade.

Tabela 4.13: Métricas que satisfazem a caracterização da evolução da pergunta P9

P9 - Quantos objetos foram mantidos?	
Nome	Número de objetos mantidos.
Definição	Número total de objetos mantidos.
Comentários	Estes objetos representam a parte "estável"da API.

4.3 Recolha de dados

Por fim, depois de desenvolvidos os exemplos de utilização efetiva das APIs e recolhido o código fonte das APIs, o último passo de implementação é o desenvolvimento dos suportes para recolha de dados.

Para a extração de informação, a abordagem passou primeiro pela utilização de uma plataforma denominada node.js e da biblioteca Esprima, que consiste num parser de Javascript, para fazer a análise sintática quer dos protótipos quer do código fonte das APIs. Após este passo, para cada ficheiro JavaScript, obtivemos um ficheiro no formato JSON, com a árvore de sintaxe abstrata (AST).

Num segundo passo da abordagem, foi implementado um programa que converte os ficheiros do formato JSON (JavaScript Object Notation) para o formato XML (Extensible Markup Language), e recolhe os dados a partir deste último formato.

A recolha destes dados foi feita com base nos valores necessários às métricas selecionadas para avaliação da usabilidade das APIs.

Os detalhes da implementação dos protótipos e do programa de recolha de dados são apresentados mais adiante no capítulo 5.

Resumindo, a abordagem é baseada em três elementos focais, os protótipos, desenvolvidos no contexto da dissertação, as APIs em si e as aplicações desenvolvidas, pelos alunos de TIG, com suporte da API do Google. Aos três elementos serão aplicadas métricas de avaliação, para a recolha de dados, de forma a avaliarmos a facilidade de compreensão e a facilidade de utilização das APIs. A recolha de dados, envolve o desenvolvimento de um programa específico para esse objetivo.

5

Implementação

Neste capítulo apresentamos os detalhes mais relevantes de toda a implementação envolvente neste estudo. Primeiro falamos um pouco das tecnologias utilizadas, seguindo-se os detalhes de implementação dos protótipos e por fim os detalhes de implementação do programa de recolha de dados.

5.1 Introdução às tecnologias utilizadas

Antes de prosseguirmos com os detalhes mais relevantes da implementação é importante oferecer um enquadramento sobre as tecnologias utilizadas.

Anteriormente mencionou-se quais as APIs em estudo, mas é importante informar quais as versões que foram utilizadas na implementação dos protótipos. São estas a Google Maps JavaScript API V3 versão 3.8, a ArcGIS API for JavaScript versão 2.7 e a OpenLayers JavaScript Mapping Library versão 2.11.

Como suporte à aplicação web desenvolvida foi utilizado o servidor Apache 2.2.21. Para a recolha de dados foi utilizada a plataforma Node.js [1], que foi necessária para a utilização da biblioteca Esprima.js [2], que por sua vez permite fazer o *parsing* de ficheiros no formato JavaScript. Quanto ao programa da recolha efetiva dos dados, este é implementado na linguagem de programação Java.

Por fim, as versões analisadas para cada API são as seguintes:

- Google Maps JavaScript API V3 - versões 3.7, 3.8 e 3.9.
- ArcGIS API for JavaScript - versões 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 3.0 e 3.1.

[1] http://nodejs.org/
[2] http://esprima.org/

- OpenLayers JavaScript Mapping Library - versões 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11 e 2.12.

5.2 Protótipos

Com a criação dos protótipos é possível avaliar a utilização efetiva das APIs, como já referido anteriormente. Assim, a solução implementada é constituída por três ficheiros JavaScript, GISFunctionalitiesEsri, GISFunctionalitiesGoogle e GIGFuncionalitiesOpenLayers, que implementam cada um as funcionalidades apresentadas na figura 5.1.

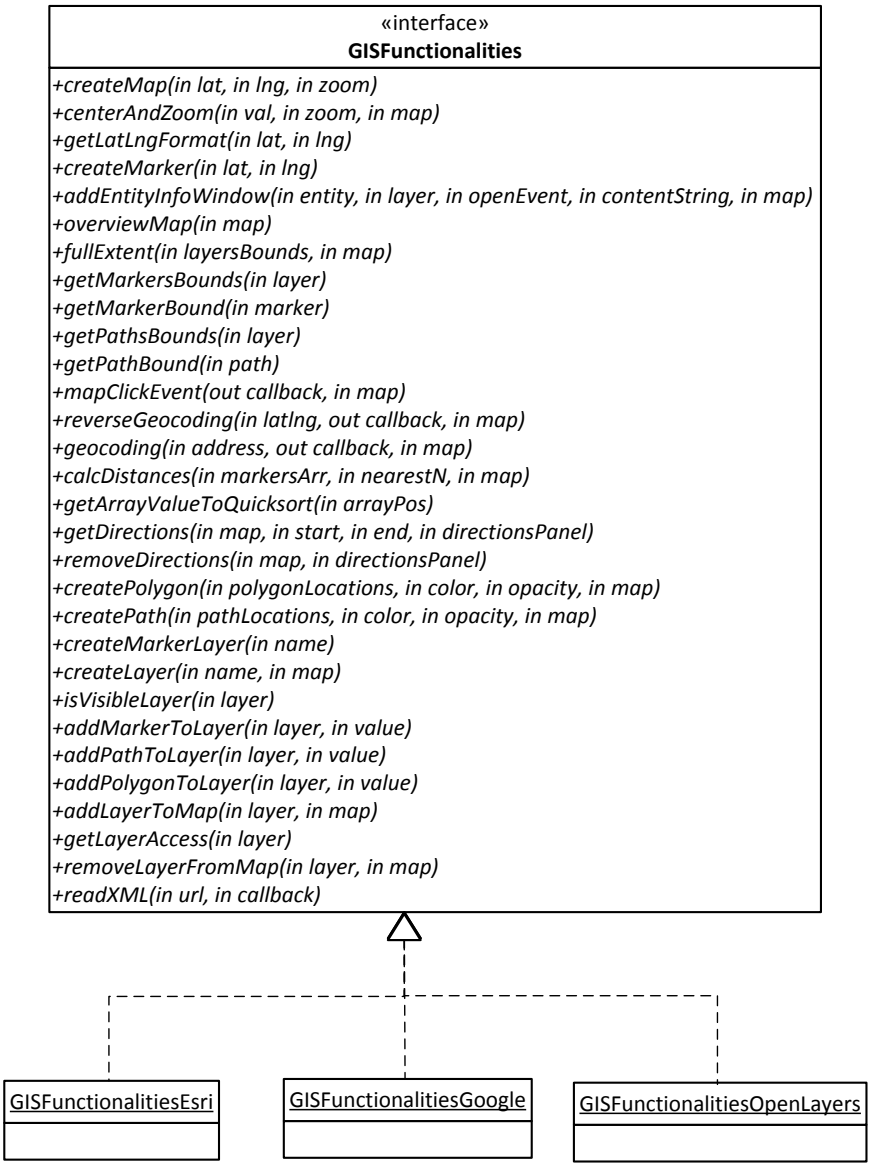

Figura 5.1: Estrutura da aplicação

Descrição das funcionalidades:

- `createMap` - funcionalidade que permite criar um mapa centrado numa determinada localização e com um determinado zoom;
- `centerAndZoom` - funcionalidade que permite centrar o mapa numa determinada localização. Zoom opcional;
- `getLatLngFormat` - funcionalidade que permite obter o tipo latLng da API;
- `createMarker` - funcionalidade que permite criar um marcador ancorado numa determinada localização;
- `addEntityInfoWindow` - funcionalidade que permite realizar a associação de uma *infowindow* e seu conteúdo a uma entidade, no mapa;
- `overviewMap` - funcionalidade que permite criar um *overviewmap*;
- `fullExtent` - funcionalidade que permite a adaptação dinâmica do nível de ampliação do mapa, de forma a incluir todas as localizações contidas no conjunto de dados (entidades) utilizado no momento;
- `getMarkersBounds` - funcionalidade que permite obter os limites de um layer de marcadores;
- `getMarkerBound` - funcionalidade que permite obter os limites de um marcador;
- `getPathsBounds` - funcionalidade que permite obter os limites de um layer de caminhos;
- `getPathBound` - funcionalidade que permite obter os limites de um caminho;
- `mapClickEvent` - funcionalidade que permite criar um evento de click sobre o mapa;
- `reverseGeocoding` - funcionalidade que permite realizar a conversão de coordenadas geográficas para endereços de morada. O argumento `callback` contém o resultado obtido;
- `geocoding` - funcionalidade que permite realizar a conversão de endereços de moradas para coordenadas geográficas. O argumento `callback` contém o resultado obtido;
- `calcDistances` - funcionalidade que permite obter os N elementos mais próximos;
- `getArrayValueToQuicksort` - funcionalidade que permite definir o valor de ordenação;

- `getDirections` - funcionalidade que permite obter as direções de determinada origem a determinado destino;
- `removeDirections` - funcionalidade que permite remover as direções desenhadas no mapa;
- `createPolygon` - funcionalidade que permite criar um polígono em determinadas localizações;
- `createPath` - funcionalidade que permite criar um caminho em determinadas localizações;
- `createMarkerLayer` - funcionalidade que permite criar um layer específico para marcadores;
- `createLayer` - funcionalidade que permite criar um layer;
- `isVisibleLayer` - funcionalidade que permite determinar se um layer está ou não visível;
- `addMarkerToLayer` - funcionalidade que permite adicionar um marcador a um layer;
- `addPathToLayer` - funcionalidade que permite adicionar um caminho a um layer;
- `addPolygonToLayer` - funcionalidade que permite adicionar um polígono a um layer;
- `addLayerToMap` - funcionalidade que permite adicionar um layer ao mapa;
- `getLayerAccess` - funcionalidade que ter acesso ao conteúdo(entidades) de um layer;
- `removeLayerFromMap` - funcionalidade que permite remover um layer do mapa;
- `readXML` - funcionalidade que permite ler dados no formato XML.

Através desta estrutura obtemos a mesma aplicação implementada com três APIs diferentes. As funcionalidades descritas permitem incorporar as principais características identificadas na secção 4.1.1. Estas funcionalidades podem ser experimentadas na aplicação desenvolvida. Com o auxílio das figuras apresentadas de seguida, descrevemos o que o protótipo permite fazer.

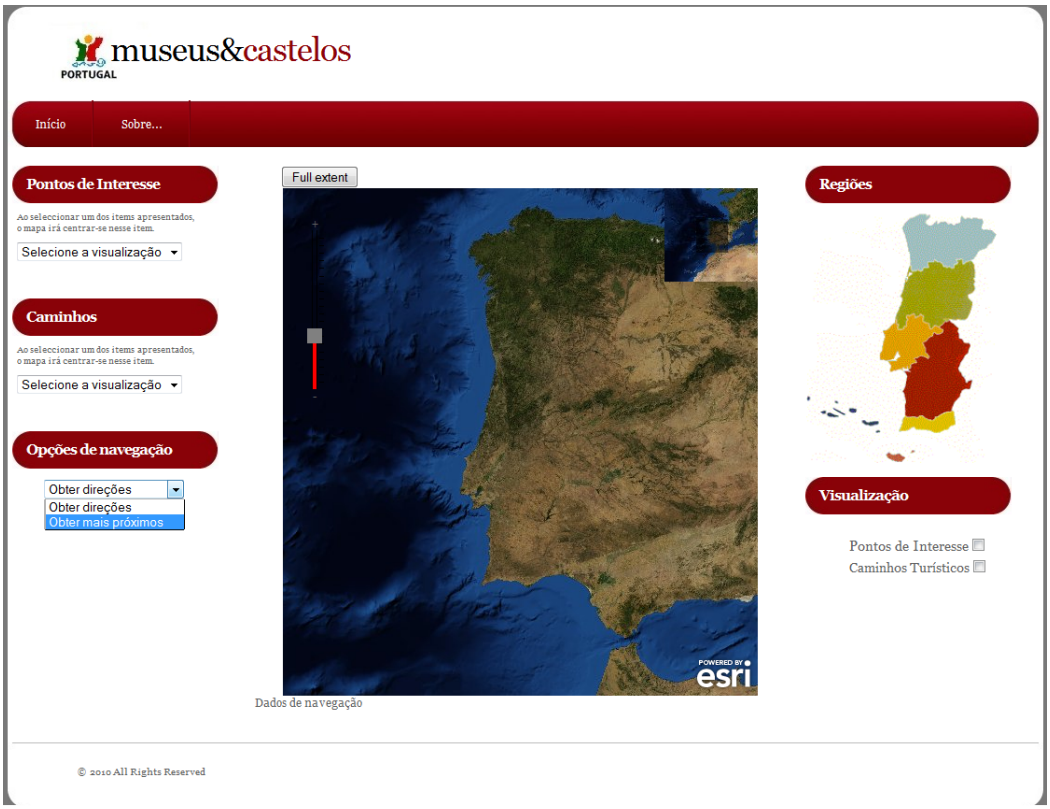

Figura 5.2: Visualização geral da aplicação

Na figura 5.2, apresentamos a visualização geral da aplicação, em que a API em utilização é a ArcGIS API for JavaScript. Na coluna da esquerda, existem vários elementos como uma lista de pontos de interesse, uma lista de caminhos turísticos e uma lista de opções de navegação. À direita, observa-se uma legenda por regiões e um campo de seleção de elementos para visualização no mapa. Finalmente, ao centro é visível o mapa e o botão de *full extent*, funcionalidade que permite realizar uma adaptação dinâmica do nível de ampliação do mapa, por forma a incluir todas as localizações contidas no conjunto de dados utilizado no momento.

Alguns dos elementos pertencentes à coluna da esquerda, dependem de elementos da coluna da direita, como é o caso da lista de pontos de interesse e da lista de caminhos turísticos. Estas listas são preenchidas quando os campos de visualização, à direita, estão selecionados, ou seja, a lista de pontos de interesse é preenchida se o campo de visualização *Pontos de Interesse* estiver selecionado. O mesmo acontece com a lista de caminhos turísticos, que é preenchida quando o campo de visualização *Caminhos Turísticos* está selecionado.

No elemento de opções de navegação, na coluna da esquerda, são permitidas duas opções:

- **Obter Direções** - O utilizador introduz um endereço de origem e um endereço de destino, onde este último tem como alternativa a seleção de um ponto no mapa e o

endereço é preenchido automaticamente.

- **Obter mais próximos** - O utilizador introduz um endereço ou seleciona um ponto no mapa e o endereço é preenchido automaticamente e, seguidamente, escolhe o número de entidades mais próximas que pretende (3 ou 5).

Com estas opções utilizam-se as capacidades de georreferenciação das APIs, que são o processo de atribuição de coordenadas geográficas a uma descrição textual (o processo inverso é denominado Reverse Geocoding).

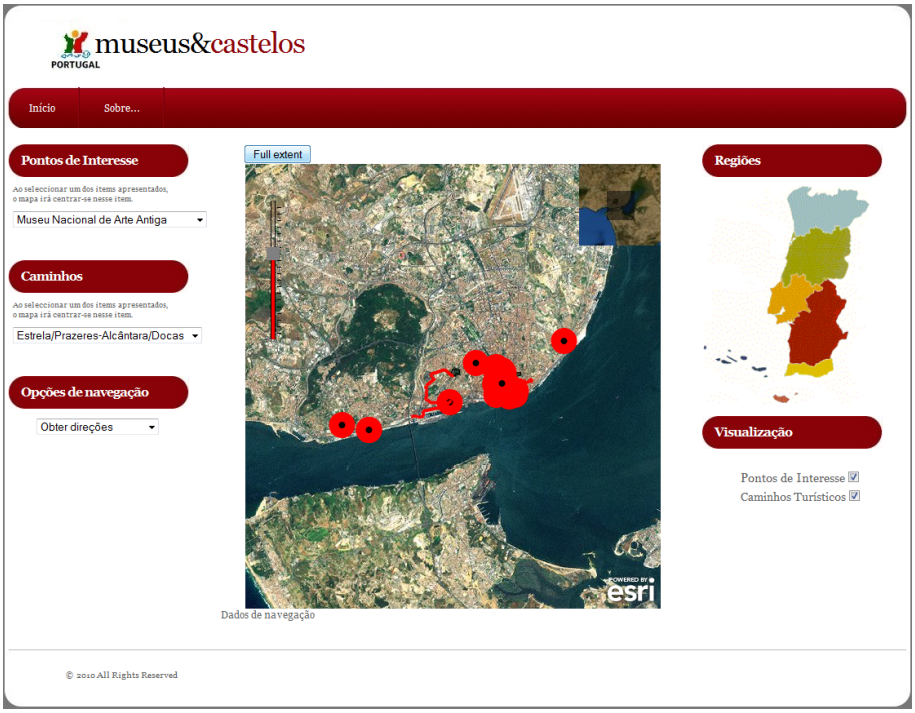

Figura 5.3: Visualização da funcionalidade de *full extent* entre outras

Na figura 5.3 é apresentada a visualização da funcionalidade de *full extent*, que adaptou o nível de ampliação do mapa para incluir todas a entidades visíveis no momento, neste caso os pontos de interesse e os caminhos turísticos. Além desta funcionalidade é possível observar o mapa de contexto (*overview map*), a barra de zoom in/out e os controladores, sobre o mapa.

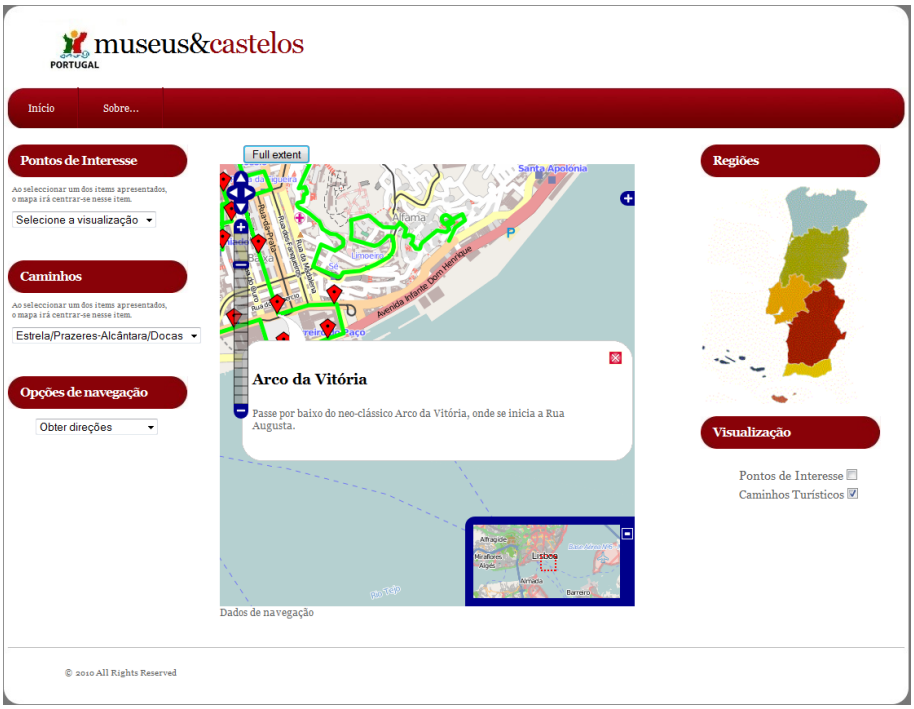

Figura 5.4: Visualização de informação associada a entidades

Na figura 5.4 observamos uma janela de informação, que no contexto em que está a ser visualizado, representa a informação de um ponto turístico de um dos caminhos turísticos.

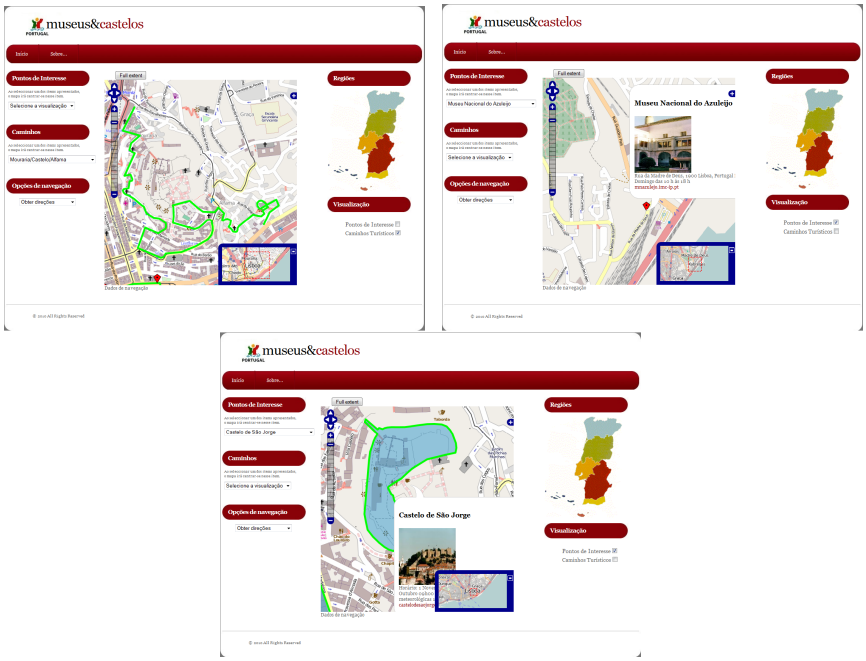

Figura 5.5: Visualização de entidades georreferenciadas

Na figura 5.5 estão representadas os três tipos de entidades que a aplicação contém. Na primeira, observa-se uma linha que representa um caminho turístico. Na segunda, observa-se um marcador que representa um ponto de interesse, neste caso o Museu Nacional do Azulejo. Na terceira e última, observa-se uma área que representa um ponto de interesse, neste caso o Castelo de São Jorge.

Como é possível observar em cada uma das imagens, as entidades visualizadas foram selecionadas na respetiva lista à esquerda e o mapa centrou a sua posição nessa entidade.

Figura 5.6: Visualização da opção de navegação *Obter Direções*

Na figura 5.6 observamos a informação de direções, obtida com cada uma das APIs. Neste caso só é possível implementar esta funcionalidade para a Google Maps Javascript API V3 e a ArcGIS API for JavaScript, pois a OpenLayers JavaScript Mapping Library não suporta a funcionalidade de georreferenciação. É ainda possível observar que para a mesma origem e destino as APIs traçam caminhos diferentes. Este acontecimento deve-se ao facto de os serviços terem dados diferentes.

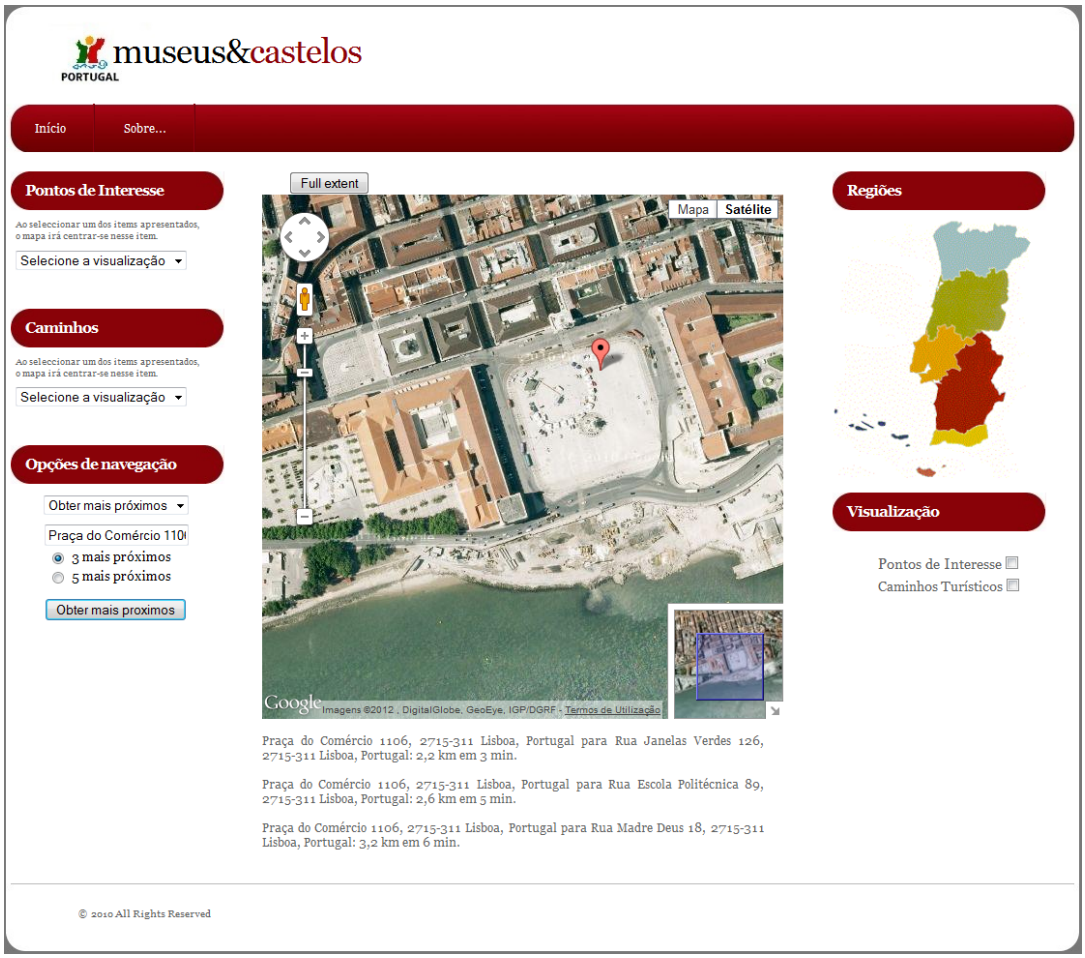

Figura 5.7: Visualização da opção de navegação *Obter mais próximos*

Na figura 5.7 observamos a informação de distâncias obtida. Também esta funcionalidade só foi possível implementar com a Google Maps Javascript API V3 e a ArcGIS API for JavaScript, pois a OpenLayers JavaScript Mapping Library não oferece suporte para o cálculo de distâncias.

Por forma a esquematizar as características de uma aplicação deste domínio, na figura 5.8 apresentamos um diagrama de características (*feature model*) que compõe a aplicação implementada. Uma característica é um aspeto de domínio visível para o utilizador. As características definem os aspetos comuns do domínio, bem como as diferenças entre os sistemas relacionados com o mesmo domínio. As características são também utilizadas para definir o domínio em termos das características obrigatórias, opcionais, ou alternativas destes sistemas relacionados [KCH+90].

Na figura 5.8 o modelo apresentado capta as características básicas do domínio da aplicação desenvolvida. As características presentes em aplicações deste domínio ou semelhante, são a existência de um mapa, a funcionalidade para obter direções, a funcionalidade para obter entidades mais próximas, a leitura de dados e a capacidade de

geo-localização que consiste na conversão de endereços de moradas para coordenadas geográficas (*geocoding*) ou, o contrário, coordenadas geográficas para endereços de morada (*reverse geocoding*). Algumas destas características podem ser definidas por sub-características. O mapa de uma aplicação deste domínio é normalmente constituído por camadas, que por sua vez são constituídas por entidades que poderão ser pontos, linhas ou áreas. Estes três tipos de entidades podem ter como sub-característica uma janela de informação. Um mapa pode ainda re sub-características de visualização, tais como o *zoom*, o *full extent*, o *overviewmap* e a existência de filtros para entidades, que podem ser pontos de interesse ou caminhos. Por fim, a leitura de dados pode também ser definida por sub-características que representam diferentes formatos de dados lidos.

Repare-se que apenas o mapa e as camadas são caraterísticas obrigatórias, o primeiro porque uma aplicação baseada em mapas terá de pelo menos apresentar um mapa, e o segundo porque o próprio mapa apresentado representa uma camada. Todas as outras características são opcionais.

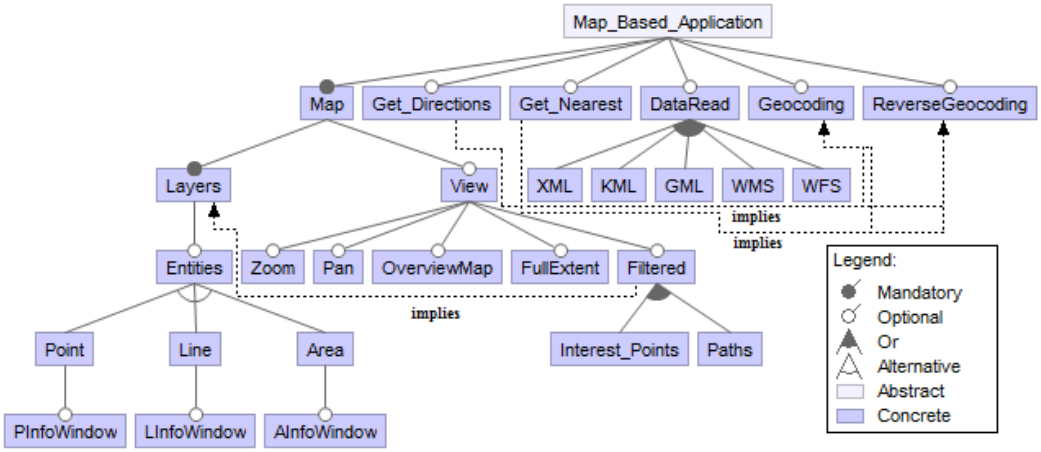

Figura 5.8: Feature Model da aplicação

Durante a implementação dos protótipos foi possível detetar diferenças entre as APIs, no que diz respeito a algumas das *features* presentes na figura 5.8, permitindo assim uma análise qualitativa.

5.2.1 Diferenças entre *features*

5.2.1.1 Exibição do mapa

Um fator muito importante a ter em conta aquando da exibição de um mapa é a sua projeção, pois esta terá influência na forma como as coordenadas são tratadas.

A projeção é o método utilizado para representar a superfície curva da terra sobre uma superfície plana [3].

[3] http://resources.esri.com/help/9.3/arcgisengine/dotnet/89b720a5-7339-44b0-8b58-0f5bf2843393.htm#MapProjections

As APIs utilizadas nos protótipos utilizam a projeção Mercator [4], que trata a Terra como uma esfera, ao invés de uma projeção que trata a Terra como um elipsoide. Isso afeta os cálculos baseados no tratamento do mapa como uma superfície plana. No entanto, embora haja distorção, como os ângulos se mantêm a projeção é boa para mapas interativos onde se passa de escalas pequenas para grandes (zooms).

Em SIG as projeções são normalmente referenciadas pelo seu código "EPSG" gerido pelo European Petroleum Survey Group. O código "EPSG:4326" descreve mapas onde as coordenadas latitude/longitude (baseada no *datum* WGS24) são tratadas como valores x/y. O código "EPSG:900913" descreve mapas onde as coordenadas em metros são tratadas como valores x/y. Um *datum* é a especificação de referência de um sistema de medição, geralmente um sistema de posições coordenadas sobre uma superfície (*datum* horizontal) ou alturas acima ou abaixo de uma superfície (*datum* vertical). Os *datums* baseiam-se em Elipsóides específicos e por vezes têm o mesmo nome que o elipsoide.

O código "EPSG:900913", apesar de não ser um código oficial continua a ser muito utilizado. O código oficialmente registado é o "EPSG:3857", cuja projeção é idêntica à do código "EPSG:900913" [5].

Google

O Google Maps baseia-se numa variante muito próxima da projeção Mercator. Este usa as fórmulas para a projeção Mercator esférica, mas as coordenadas de recurso do Google Maps são as coordenadas GPS baseadas no datum WGS 84, que é um *datum* muito utilizado à escala global. A diferença entre uma esfera e o elipsoide WGS 84 faz com que a projeção resultante não seja precisamente conforme. A discrepância é impercetível à escala global, mas faz com que os mapas das áreas locais se desviem um pouco dos verdadeiros mapas elipsoidais Mercator na mesma escala[6].

Com esta API é possível alterar a projeção.

ArcGIS

Utilizando a API JavaScript da ArcGIS é possível apresentar mapas em qualquer projeção [7]. Isto porque, sempre que se cria uma coordenada é necessário especificar a referência espacial. A referência espacial é definida por uma well-known ID (WKID)[8].

OpenLayers

Utilizando a API JavaScript da OpenLayers é possível apresentar mapas em qualquer projeção, pois na criação do mapa é possível especificar a projeção. No OpenLayers a projeção Mercator utiliza o código "EPSG:900913" [9].

[4] http://docs.openlayers.org/library/spherical_mercator.html
[5] http://ww.epsg-registry.org/export.htm?gml=urn:ogc:def:crs:EPSG::3857
[6] http://en.wikipedia.org/wiki/Google_Maps#Google_Maps_API
[7] http://resources.esri.com/help/9.3/arcgisserver/apis/javascript/arcgis/help/jshelp/overview_api.htm
[8] http://resources.esri.com/help/9.3/arcgisserver/apis/javascript/arcgis/help/jsapi/spatialreference.htm
[9] http://docs.openlayers.org/library/spherical_mercator.html#sphericalmercatorandepsgaliases

5.2.1.2 Layers

Os *layers* são objetos no mapa que consistem num ou mais itens separados, mas que são manipulados como uma única unidade. Geralmente, os *layers* refletem coleções de objetos que se adicionam ao mapa para designar uma associação comum [10]. Não confundir o conceito de *layer* com o conceito de *overlay*, que são objetos no mapa que estão vinculados a uma coordenada latitude/longitude. Os overlays refletem objetos que se adicionam ao mapa, para designar os pontos, linhas, áreas, ou coleções de objetos[11].

Google

A API não suporta a gestão de todos os tipos de *layers*, pelo que a forma de simular *layers* para entidades, por exemplo, é através da utilização de *arrays*. A API apenas suporta a gestão dos seguintes tipos de *layers*: `google.maps.KmlLayer`, `google.maps.BicyclingLayer`, `google.maps.FusionTablesLayer` e `google.maps.TrafficLayer`.

A API do Google Maps suporta os formatos de dados KML e GeoRSS para a exibição de informação geográfica. Estes formatos de dados são exibidos num mapa usando um objeto KmlLayer, cujo construtor recebe o URL de um ficheiro, publicamente acessível, KML ou GeoRSS.

É também possível adicionar informações sobre caminhos de bicicleta para os mapas, usando o objeto `BicyclingLayer`. O objeto `BicyclingLayer` permite disponibilizar um *layer* de percursos de ciclismo e de rotas de ciclismo sugeridas.

A API do Google Maps permite processar os dados contidos nas FusionTables do Google como um *layer* num mapa, usando o objeto `FusionTablesLayer`. Uma FusionTables do Google é uma tabela de base de dados onde cada linha contém dados sobre um determinado recurso.

Por fim, é ainda possível adicionar ao mapa, informação de trânsito em tempo real (quando suportado), usando o objeto `TrafficLayer`.

ArcGIS

A API suporta a gestão de *layers* através da classe `esri.layers.GraphicsLayer`. O objecto `GraphicsLayer` é um *layer* que contém um ou mais objetos `Graphic`, que por usa vez é composto pelo objeto base `Geometry`. Neste contexto uma geometria pode ser um objeto `Point` (uma localização definida por coordenadas x e y), um objeto `Multipoint` (uma coleção ordenada de objetos `Point`), um objeto `Polygon` (um array de anéis, em que cada anel é um array de objetos `Point`) ou um objeto `Polyline` (um array de caminhos, em que cada caminho é um array de objetos `Point`).

Além esta classe, que permite a gestão de *layers* para geometrias, a API também suporta os seguintes tipos de *layers*: `esri.layers.KMLLayer`, `esri.layers.WMSLayer`

[10] https://developers.google.com/maps/documentation/javascript/layers#LayersOverview
[11] https://developers.google.com/maps/documentation/javascript/overlays#OverlaysOverview

e `esri.layers.WMTSLayer`.

O objeto `KMLLayer` permite criar um *layer* baseado num ficheiro KML, o objeto `WMSLayer` permite criar um *layer* para OGC Web Map Services (WMS), que é um protocolo padrão para disponibilizar imagens de mapas georreferenciados na internet [12] e o objeto `WMTSLayer` permite criar um *layer* para OGC Web Map Tile Service (WMTS), que é um protocolo padrão para armazenar e obter dados cartográficos [13].

OpenLayers

A API suporta a gestão de *layers* através dos objetos: `OpenLayers.Layer.Markers` [14] e `OpenLayers.Layer.Vector` [15]. O *layer* `OpenLayers.Layer.Markers` apenas suporta pontos, enquanto que o *layer* `OpenLayers.Layer.Vector` suporta pontos, linhas e polígonos.

Os *layers* baseados no objeto `Vector` são de manutenção fácil, pois este objeto está a ter novo desenvolvimento na OpenLayers. Neste objeto existe mais suporte para opções de estilo, maior configurabilidade sobre o comportamento do layer e sobre as interações com servidores remotos.

O *layer* `Marker` apenas é mantido para compatibilidade com versões anteriores. Em geral, o objeto `Marker` é a forma antiga de lidar com dados geográficos no *browser*. Novos projetos devem, sempre que possível, utilizar *layers* do objeto Vector no lugar de *layers* do objeto `Marker`.

Em relação ao acesso aos *layers* que se acionam para visualização, o último *layer* a ser adicionado ao mapa é aquele ao qual se tem acesso para seleção.

Além dos *layers* referidos, a API também suporta os seguintes *layers* : `OpenLayers.Layer.GeoRSS`, `OpenLayers.Layer.TMS` e `OpenLayers.Layer.WMS`.

O objeto `GeoRSS` , que é um padrão de codificação de localização como parte de um Web feed (Web feeds são usados para descrever *feeds* ("canais") de conteúdo, como notícias, blogs de áudio, vídeo blogs e entradas de blog de texto. Estes *feeds* da Web são processados por programas como agregadores e navegadores web) [16], permite adicionar pontos `GeoRSS` ao mapa. Em `GeoRSS`, o conteúdo de localização consiste em pontos geográficos, linhas e polígonos de interesse. Os `GeoRSS feeds` são projetados para serem consumidos por software geográfico, tais como geradores de mapas.

O objeto `TMS` permite criar *layers* para aceder a tiles de serviços que estejam em conformidade com a especificação Tile Map Service [17].

Por fim, o objeto `WMS` permite criar *layers* para exibição de dados de OGC Web Mapping Services (WMS) [18].

[12] http://www.opengeospatial.org/standards/wms
[13] http://www.opengeospatial.org/standards/wmts
[14] http://docs.openlayers.org/library/layers.html#markers
[15] http://docs.openlayers.org/library/layers.html#vector
[16] http://en.wikipedia.org/wiki/GeoRSS
[17] http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification
[18] http://www.opengeospatial.org/standards/wms

Os *layers* utilizados nos protótipos, para cada uma das APIs, permitem manipular as entidades (pontos, linhas e polígonos). O outros *layers* mencionados, para cada uma das APIs, não foram testados, pois como não existem em todas as APIs não é possível efetuar uma comparação entre elas.

5.2.1.3 Entidades

Neste contexto uma entidade representa um ponto, linha ou polígono.

Google

Pontos: Para a representação de pontos com a API da Google utiliza-se o objeto `google.maps.Marker`. Este objeto disponibiliza um símbolo por defeito.

Linhas: Para a representação de linhas a API disponibiliza o objeto `google.maps.Polyline`. As opções de definição incluem propriedades como `strokeColor`, `strokeOpacity` e `strokeWeight`. Estas propriedades permitem definir a côr, o grau de opacidade e a espessura da linha, respetivamente.

Polígonos: Para a representação de polígonos é disponibilizado o objeto `google.maps.Polygon`. A funcionalidade é idêntica á da linha, embora com a necessidade de definir, além das propriedades `strokeColor`, `strokeOpacity` e `strokeWeight`, as propriedades `fillColor` e `opacity`. Estas propriedades permitem definir a côr de preenchimento e o grau de opacidade do polígono.

ArcGIS

Pontos: Para a representação de pontos é necessária o objeto `esri.geometry.Point`. O objeto que permite criar pontos apenas permite definir as suas coordenadas, e a referência espacial.

Linhas: Para a representação de linhas é necessária o objeto `esri.geometry.Polyline`. O objeto que permite criar linhas apenas permite definir as coordenadas que compõem a linha, e a referência espacial.

Polígonos: Para a representação de polígonos a API disponibiliza o objeto `esri.geometry.Polygon`. O objeto que permite criar polígonos apenas permite definir as coordenadas que compõem a área, e a referência espacial.

Como referido anteriormente na discussão dos *layers*, esta API contém objetos `Graphic`, que por sua vez são compostos por objectos base `Geometry`, os quais podem ser um objeto `Point`, um objeto `Multipoint`, um objeto `Polygon` ou um objeto

`Polyline`. O objecto Graphic permite associar, a estas geometrias, um símbolo e o conteúdo de informação sobre a geometria.

OpenLayers

A API do OpenLayers oferece duas formas para o desenho de entidades no mapa. Uma das formas é o Vector Overlays, que utiliza recursos de desenho de vetor do navegador web (SVG, VML, ou Canvas) para a exibição dos dados. A outra forma é o tipo Marker Overlays, que apresenta objetos de imagem HTML dentro do Document Object Model (DOM) [19]. O Vector Overlays, conseguido através do objeto `OpenLayers.Feature.Vectors`, pode ser um objeto `Poit/MultiPoint`, um objeto `Line/MultiLine` ou um objeto `Polygon/MultiPolygon`. O Marker Overlays, conseguido através do objeto `OpenLayers.Marker`, apenas suporta o objeto `Point`.

5.2.1.4 InfoWindows

Uma janela de informação, como o próprio nome indica, é uma janela sobre o mapa que permite a visualização de conteúdo. Uma janela de informação está associada a um local específico do mapa por meio de coordenadas. Para a visualização de uma janela de informação é necessário clicar sobre uma entidade (mapa, ponto, linha ou polígono) e, caso esta esteja associada a uma janela de informação, surgirá uma janela com o seu conteúdo que lhe foi atribuído.

Google

A adição de uma *infowindow* é realizada através de um evento "click" sobre uma das entidades (mapa, ponto, linha, polígono).

ArcGIS

Como referido anteriormente, o objeto Graphic é composto pelo objeto Geometry (que pode ser um ponto, linha ou polígono) e é também composto pelo objeto InfoTemplate, que permite definir o conteúdo que será visível na *infowindow*. Ao contrário do que acontece com a API do Google, com esta API não existe a necessidade de associar as entidades as eventos, uma vez criado o objeto InfoTemplate o evento é associado automaticamente.

OpenLayers

Os dois tipos de dados existentes (Vector Overlays e Marker Overlays) para desenho de entidades, originam duas formas diferentes de implementação da *infowindow*. A definição da *infowindow* em si, é realizada da mesma forma para os dois tipos (através do objeto `OpenLayers.Popup.FramedCloud`) , e o registo de um evento ocorre também nos dois tipos. No entanto, a forma como se disponibiliza esse evento é diferente. Para

[19] http://docs.openlayers.org/library/overlays.html

os marcadores, o objeto `Marker` disponibilizado pela API permite registar um evento para cada uma das entidades `Marker`. Para os polígonos, a interação é realizada através do controlo `SelectFeatureControl`. Este controlo permite a seleção de *features* utilizando eventos DOM para a captura da *feature* que foi selecionada (através do evento click). Além de se associar o controlo ao *layer* que contém as *features*, também é necessário registar o evento de `featureselected` a esse mesmo *layer*.

5.2.1.5 Geocoding/ReverseGeocoding

O processo de georreferenciação consiste na conversão de endereços de moradas para coordenadas geográficas (*geocoding*) ou, o contrário, coordenadas geográficas para endereços de morada (*reverse geocoding*).

Google

O processo de georreferenciação é conseguido através do serviço `google.maps.Geocoder`. É através do método geocode, passando um objeto `GeocoderRequest`, que contém as propriedades opcionais de endereço de morada, área de delimitação de pesquisa, localização e região, bem como uma função de *callback*, e que permite o acesso a um *array* de objetos `GeocoderResult`, que a API do Google oferece a funcionalidade de georreferenciação. No caso do*geocoding*, ou seja, a conversão de endereço de morada em coordenadas, define-se a propriedade *address* do objeto `GeocoderRequest`. No caso do *reverse geocoding*, ou seja, a conversão de coordenadas num endereço de morada, define-se a propriedade location do objeto `GeocoderRequest`. Através do objeto `GeocoderResult` acede-se ao resultado, ou resultados no caso de existir mais do que um objeto no array, do pedido.

ArcGIS

O processo de georreferenciação é conseguido através do serviço `esri.tasks.Locator`, que é um serviço de *geocoding* disponibilizado pela API REST do ArcGIS Server. Este serviço é usado para gerar candidatos a um endereço, ou para encontrar um endereço para uma localização, e necessita da definição da variável `esri.config.defaults.io.proxyUrl`, que é uma página de proxy que lida com a comunicação com os serviços ArcGIS Server [20].

Para obter um endereço de morada, a partir de coordenadas, utiliza-se o método `addressToLocations`, o qual recebe como parâmetro o endereço de morada, enquanto que para obter coordenadas a partir de um endereço de morada utiliza-se o método `locationToAddress`, o qual recebe uma coordenada como parâmetro. O(s) resultado(s) obtêm-se lidando com o evento `onAddressToLocationsComplete` ou `onLocationToAddressComplete`, respetivamente, e definindo o comportamento da aplicação, por meio de uma função, quando estes eventos são alcançados.

[20] http://help.arcgis.com/en/webapi/javascript/arcgis/help/jshelp/ags_proxy.htm

OpenLayers

A API do OpenLayers não suporta o processo de georreferenciação.

5.2.1.6 Obter Direções

Google

O processo para obter direções é conseguido através do serviço `google.maps.DirectionsService`. O objeto `DirectionsService` permite calcular direções, pois este comunica com o Google Maps API Directions Service que recebe o pedido de direção e retorna os resultados computados [21]. O pedido é realizado através do método `route`, que é composto pelo objeto `DirectionsRequest` e por uma função de `callback`, composta pelo objeto `DirectionsResult`. O objeto `DirectionsRequest` permite definir uma propriedade, denominada `travelMode`, que especifica o tipo de rota a pedir ao servidor. Os modos de viagem válidos são: a conduzir, a pé ou de bicicleta. O objeto `DirectionsResult` contém as direções resposta, obtidas a partir do servidor de direções.

ArcGIS

O processo para obter direções é conseguido através do serviço `esri.tasks.RouteTask`. O objeto `RouteTask` permite encontrar rotas entre dois ou mais locais e, opcionalmente, obter direções de condução. Para tal, é utilizado o serviço de análise de rede ArcGIS Server para o calculo das rotas [22].

Por forma a obter os resultado desejados a partir do objeto `RouteTask` é necessário especificar os detalhes do problema, tais como locais de paragem e locais de barreira. Esta definição é realizada a través do objeto `RouteParameters`. Uma vez criada a `RouteTask` e definidos os parâmetros do objeto `RouteParameters`, utiliza-se o método `RouteTask.solve()` para a resolução do percurso. É ainda necessário lidar com o evento `onSolveComplete` e definir o que a aplicação fará com os resultados, por meio de uma função associada ao evento. Essa função permitirá aceder aos resultados obtidos.

Tal como no serviço `esri.tasks.Locator`, também é necessário que a variável `esri.config.defaults.io.proxyUrl` esteja declarada.

OpenLayers

A API do OpenLayers não suporta o processo para obter direções.

5.2.1.7 Obter distâncias

Google

Para obter distâncias utiliza-se o serviço `google.maps.DistanceMatrixService`.

[21] https://developers.google.com/maps/documentation/javascript/directions#Directions
[22] http://help.arcgis.com/en/webapi/javascript/arcgis/help/jshelp_start.htm#jshelp/intro_route_routetask.htm

O serviço `DistanceMatrix` calcula distâncias e duração da viagem entre múltiplas origens e destinos, de acordo com um dado modo de viagem. Este serviço não retorna as informações de rota detalhada. As informações de rota, como referido anteriormente, são obtidas através do serviço de direções.

É através do método `getDistanceMatrix` que se inicia o pedido ao serviço, passando um objeto `DistanceMatrixRequest`, que contém as origens, destinos e modo de viagem, bem como uma função de `callback` para executar após a receção da resposta. O resultado da resposta é um objeto `DistanceMatrixResponse`, que consiste na morada de origens e destinos formatada, e numa sequência de objetos `DistanceMatrixResponseRows`, um para cada endereço de origem correspondente. Um objeto `DistanceMatrixResponseRows` consiste numa sequencia de objetos `DistanceMatrixResponseElements`, um para cada endereço de destino correspondente. O objeto `DistanceMatrixResponseElements`, por sua vez, contém a duração e a distância a partir de uma origem a um destino.

ArcGIS

Para obter distâncias utiliza-se também o serviço `esri.tasks.RouteTask`, pois nos resultados obtidos pelo serviço, esta contida a informação sobre a distância da origem ao destino. Além deste serviço, o serviço `esri.tasks.GeometryService` também permite calcular a distância entre uma origem e destino, mas esta distância obtida corresponde à distância direta.

OpenLayers

A API do OpenLayers não suporta o processo para obter distâncias.

5.2.1.8 Leitura de dados

Google

A API da google não suporta a leitura de dados XML ao contrário da versão anterior (Google Maps JavaScript API v2), que continha o método `GDownloadUrl`. De forma a adicionar esta necessidade, existe uma extensão, por software, a esta funcionalidade disponibilizada em JavaScript: http://code.google.com/p/gmaps-api-issues/issues/detail?id=1364. Desta forma é possível obter o objeto `XMLDocument`

ArcGIS

Para a leitura de dados XML é necessário converter o URL (Uniform Resource Locator) da localização do ficheiro a ser lido com o método `esri.urlToObject`, que realiza precisamente essa a conversão do URL para uma representação de um objeto JSON. O formato do objeto é: path: <String>, query:key:<Object>.

Por fim, utiliza-se o método `esri.request`, que recebe como parâmetro o URL convertido, o conteúdo associado ao URL e o tipo de ficheiro a ler. Desta forma é possível

obter o objeto `XMLDocument`

OpenLayers

No caso desta API, existem vários formatos para leitura/escrita de dados. Os formatos são os seguinte: `OpenLayers.Format.GML`, `OpenLayers.Format.WKT`, `OpenLayers.Format.XML`, `OpenLayers.Format.KML`, `OpenLayers.Format.XLS`. Nos protótipos realizados apenas se pretende ler dados XML. No entanto, como nenhuma das outras APIs disponibiliza classes semelhantes não se utilizou o objeto `OpenLayers.Format.XML`, pois não seria possível efetuar comparações. Assim, utilizou-se o método `OpenLayers.Request.GET` porque, tal como nas outras APIs, permite obter o objeto `XMLDocument`.

5.3 Programa de recolha de dados

Após a implementação dos protótipos, passámos à recolha de dados com base nas métricas selecionadas.

No capítulo 4, sobre a abordagem tomada para o estudo, foi referido qual o processo seguido no tratamento dos ficheiros e recolha de dados. Relembremos esse processo observando a figura 5.9.

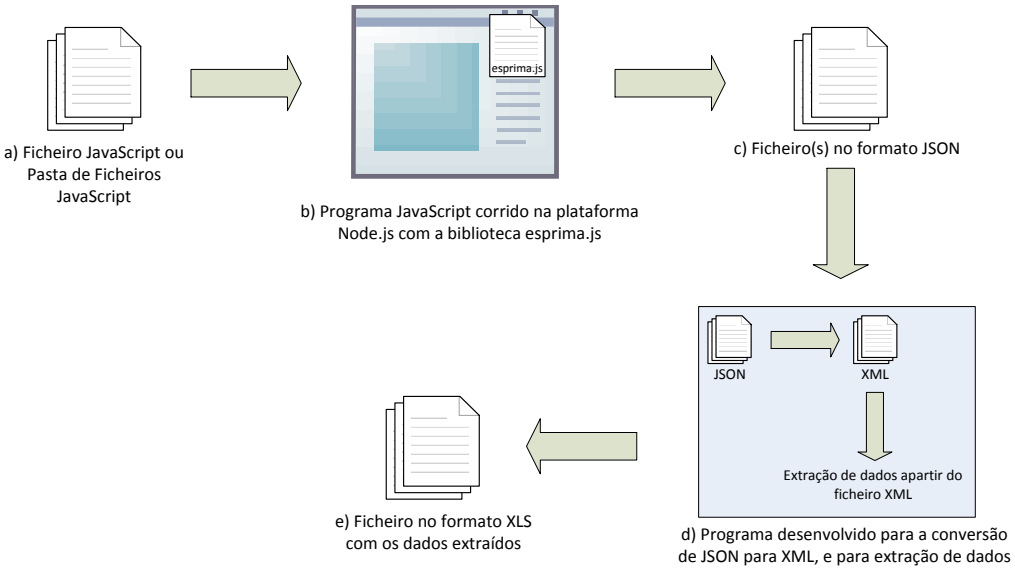

Figura 5.9: Esquematização da recolha de dados

Para a recolha de dados, além dos protótipos, também é necessário o código fonte das APIs, pois algumas das métricas são direcionadas para as interfaces e, como referido

anteriormente, tanto os protótipos como as APIs estão implementados em JavaScript. Assim, para permitir a realização da análise quantitativa com base em métricas de software, começamos por recolher a informação para o cálculo dessas métricas, com uma análise sintática ao código em JavaScript, conseguida através do analisador sintático Esprima [23].

Na figura 5.9, o objeto a) representa o ficheiro ou ficheiros JavaScript que serão analisados, neste caso os ficheiros serão os protótipos e os ficheiros JavaScript que constituem cada uma das APIs. Com o auxílio da plataforma Node.js [24], que é uma plataforma que permite facilmente e rapidamente construir aplicações de rede escaláveis, desenvolveu-se um pequeno programa, representado como o objeto b), que lê os ficheiros JavaScript, faz a sua análise sintática, e tem como resultado um ficheiro no formato JSON com a árvore de sintaxe abstrata (AST), aqui representado como o objeto c). O analisador sintático Esprima, sobe a forma de ficheiro JavaScript, é incorporado como uma biblioteca na plataforma Node.js.

Após a análise sintática, é possível passar à recolha de dados. Para a recolha de dados desenvolveu-se uma programa, objeto d), que lê o(s) ficheiro(s) JSON, converte-o(s) em ficheiro(s) XML e extrai a informação necessária. Toda a extração de dados é armazenada em ficheiros XLS, objeto e). Enquanto que os protótipos seguem todos a mesma estrutura de implementação, e portanto existe uma única classe que trata da recolha de dados, já o código fonte das APIs apresentam estruturas diferentes, tanto dos protótipos como entre elas, pelo que, para cada API, existe uma classe que trata da recolha de dados. Tanto a conversão como a recolha de dados foram implementadas com a linguagem Java.

Finalizado este processo, os dados são analisados estatisticamente através de gráficos gerados a partir do programa IBM SPSS Statistics.

Antes de passarmos ao capítulo 6, de resultados e discussão, apresentamos duas secções dedicadas a alguns detalhes importantes referentes a recolha de dados sobre as APIs e protótipos, respetivamente.

5.3.1 APIs

A recolha de dados das APIs recaiu sobre a informação a que o utilizador tem acesso, tais como os construtores de objetos, e funções e propriedades de objetos, pois é na utilização dos objetos que existe a interação do utilizador com a API. Os detalhes de implementação do código fonte das APIs são irrelevantes para o utilizador, que observa as APIs como caixas negras, para efeitos do estudo de usabilidade da API.

Relativamente às APIs e acesso ao seu código fonte, para a API da Google não é possível aceder a este recurso, e portanto, para a recolha os dados desejados, optámos por utilizar a respetiva página de referência da API. Assim, a recolha de dados realizou-se sobre um ficheiro HTML. Para as restantes APIs, da OpenLayers e da Esri, é possível descarregar um ficheiro no formato ZIP com toda a informação do código fonte.

Relativamente à recolha de dados sobre as API, para as métricas MSC e MIL, que

[23] http://esprima.org/
[24] http://nodejs.org/

recaem sobre os identificadores, consideraram-se os identificadores das funcionalidades oferecidas pelos objetos das interfaces.

Um dos fatores importantes para a compreensão da API está relacionado com os argumentos das funções, pois a consistência de nomes na declaração de argumentos contribui para uma melhor utilização da interface. No entanto, constatámos que nem todas as APIs permitem conhecer o nome dos argumentos, como é o caso da API da Esri, pelo que não foram recolhidos dados sobre estes identificadores, pois não seria possível realizar uma comparação entre as três.

5.3.2 Utilização das APIs

Para recolha de dados da utilização efetiva das APIs, recordamos que as métricas selecionadas permitem medir o índice de utilização da API (APIUI) e a frequência de utilização da API, esta última aplica ao conjunto de aplicações desenvolvidas com o suporte da API do Google. Para o índice de utilização da API considerámos que a utilização é uma chamada a um construtor, uma chamada a uma função ou uma chamada a uma propriedade.

Para a análise da frequência de utilização da API, considerámos apenas a chamada a construtores dos objetos, pois uma vez que o JavaScript é uma linguagem de tipagem fraca e dinâmica, nem sempre é possível saber em tempo de compilação, a que objeto pertence uma propriedade ou função. Por exemplo, no caso de existirem funções com o mesmo nome mas de objetos diferentes, e a variável que está a fazer o acesso a essa função é, por exemplo, um argumento, então não é possível saber a que objeto pertence essa função e consequentemente não é possível medir a frequência de utilização dessa função.

6

Resultados e Discussão

Neste capítulo apresentamos, na secção 6.1, para cada questão proposta para os objetivos de avaliação da usabilidade das APIs em estudo, os dados obtidos através da aplicação das métricas e outros dados relativos às APIs. De seguida, na secção 6.2, discutem-se os dados obtidos. Por fim, as duas últimas secções são dedicadas às ameaças à validade e às limitações.

6.1 Resultados obtidos

6.1.1 Evolução das APIs

As questões P6, P7, P8 e P9 permitem medir e analisar a evolução das APIs. No gráfico da figura 6.1, apresentamos os dados obtidos relativamente ao número de objetos, do número de métodos e de propriedades, que cada API utilizada no estudo contém. Nos gráficos das figuras 6.2, 6.3, 6.4, 6.5, 6.6 e 6.7 apresentamos a evolução do número de objetos, do número de métodos e de propriedades, entre versões das APIs. Estes dados são apresentados em gráficos de colunas.

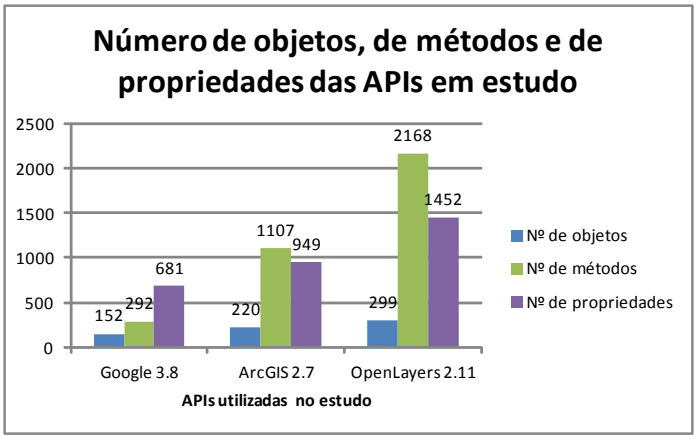

Figura 6.1: Número de objetos, de métodos e de propriedades de cada API utilizada no estudo

No gráfico da figura 6.1, observa-se para cada API, o número total de objetos, o número total de métodos e o número total de propriedades, que a constituem. Um objeto pode ser constituído apenas por propriedades, apenas por métodos, ou por ambos. Comparando com a API da OpenLayers, observamos que a API da Google é constituída por cerca de metade do número de objetos. Por seu lado, a API do ArcGIS apresenta um número de objetos intermédio entre as restantes. Se observarmos o valor dos métodos existentes por API, a diferença ainda é mais acentuada. A API da Google tem cerca de 13% do número de métodos quando comparada com a API da OpenLayers, e cerca de 26% do número de métodos quando comparada com a API da Esri. Quanto ao número de propriedades, a API da Google tem cerca de 47% do número de propriedades quanto comparada com a API da OpenLayers, e cerca de 72% do número de propriedades quando compara com a API da Esri.

Tendo em conta estas diferenças é possível constatar que a API da Google é muito mais pequena, em termos de objetos, métodos e propriedades, do que a API do ArcGIS que, por sua vez, é também consideravelmente mais pequena que a API do OpenLayers.

Além de uma visão estática das três APIs, é importante conhecer a sua evolução, que tem impacto na estabilidade da API ao longo do tempo. Uma forma de o fazer é estudar a evolução dos elementos das APIs que foram mantidos, criados, ou removidos de uma versão para a seguinte.

Nos gráficos das figuras 6.2, 6.4 e 6.6, cada um relativo a uma API, é possível observar três valores diferentes para cada coluna. A cor azul representa o número de objetos mantidos de versão para versão, a cor verde representa o número de objetos acrescentados na respetiva versão e a cor vermelha representa o número de objetos eliminados de versão para versão.

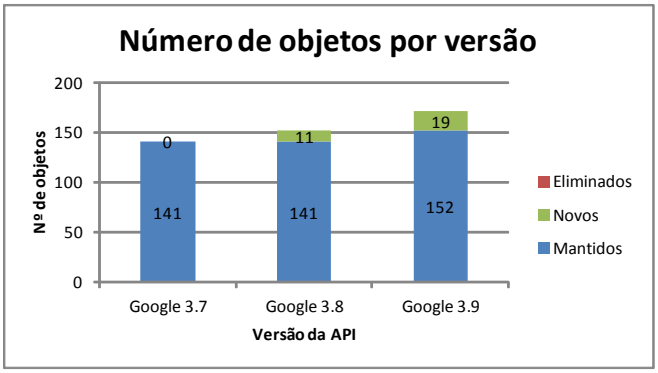

Figura 6.2: Número de objetos ao longo das versões - Google

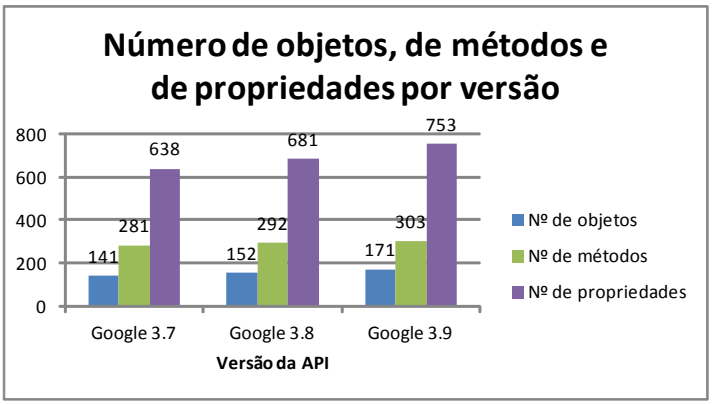

Figura 6.3: Número de objetos, de métodos e de propriedades ao longo das versões - Google

No gráfico da figura 6.2, relativo à API da Google, observa-se a existência de um aumento de 30 objetos, desde a primeira versão da API do Google maps considerada neste estudo (3.7) até à última (3.9), sem a existência de objetos eliminados entre versões, o que é importante para facilitar a retro-compatibilidade, ou pelo menos uma fácil migração de uma versão da API para a versão seguinte. No gráfico da figura 6.3 observa-se que entre a primeira e a última versão houve um acréscimo de 22 métodos e um acréscimo de 115 propriedades. Portanto, entre a primeira e a última versão, a API, revista no âmbito desta dissertação, aumentou cerca de 21% no número de objetos, cerca de 8% no número de métodos e cerca de 18% no número de propriedades.

A versão 3 da API da Google foi lançada em 2009. Neste estudo consideramos apenas a evolução a partir da versão 3.7, dado apenas termos conseguido o acesso a esta versão e as que se seguiram, mas não a versões anteriores.

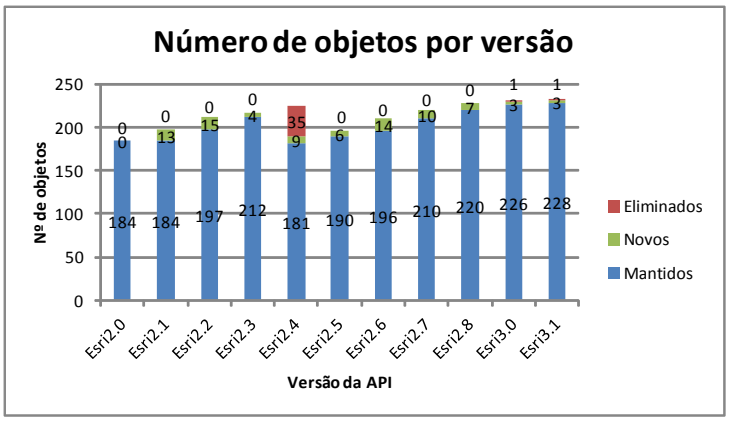

Figura 6.4: Número de objetos ao longo das versões - ArcGIS

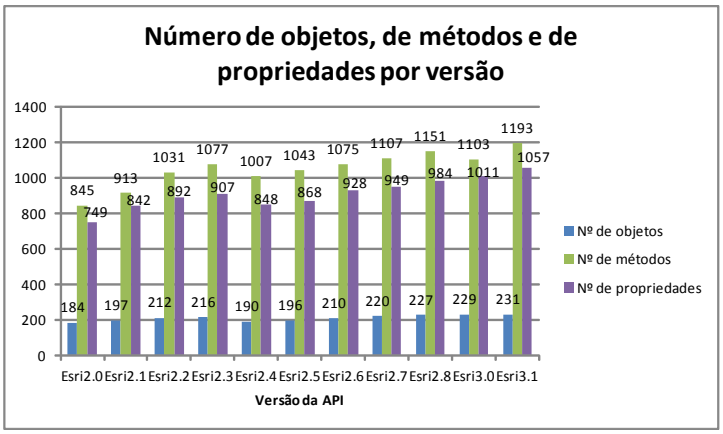

Figura 6.5: Número de objetos, de métodos e de propriedades ao longo das versões - ArcGIS

No gráfico da figura 6.4, relativo à API da Esri, observa-se um aumento de 47 objetos desde a primeira versão até à última. À exceção de uma quebra na versão 2.4, e da eliminação de apenas 1 objeto nas versões 3.0 e 3.1, nas restantes versões não foram eliminados objetos. A partir da versão 2.4 é possível observar, no gráfico da figura 6.5, que o número de objetos e o número de propriedades aumenta gradualmente até a última versão, mas o mesmo não acontece com o número de métodos, que na versão 3.0 diminui ligeiramente. Estas eliminações implicam uma quebra de compatibilidade, para as versões seguintes, de aplicações de uma versão anterior que usassem esses objetos.

Entre a primeira e a última versão, a API cresceu cerca de 26% no número de objetos e cerca de 41% no número de métodos e de propriedades.

Por fim, é importante referir que, no caso desta API, sabe-se que o lançamento da versão 2 teve lugar no ano de 2010.

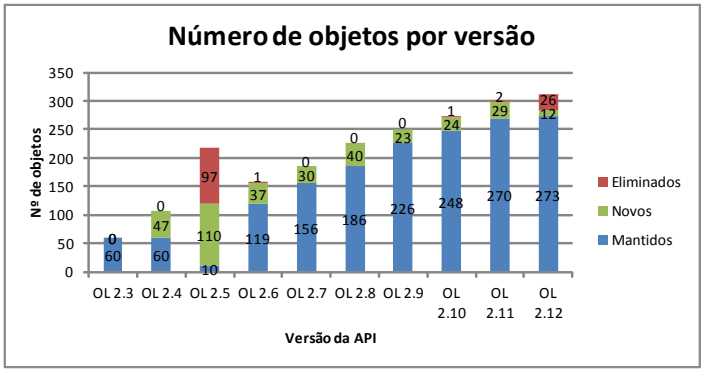

Figura 6.6: Número de objetos ao longo das versões - OpenLayers

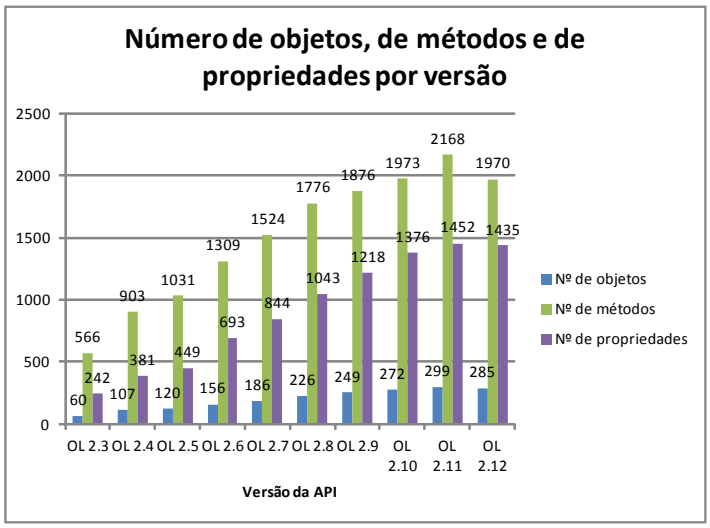

Figura 6.7: Número de objetos, de métodos e de propriedades ao longo das versões - OpenLayers

No gráfico da figura 6.6, relativo à API da OpenLayers, observa-se um aumento de 225 objetos, desde a primeira versão até à última. Em comparação com os aumentos observados nas outras APIs, nesta API existiu um maior aumento no número de objetos. É também possível observar que na versão 2.5 houve uma maior mudança, tendo sido eliminados 97 objetos, acrescentados 110 e mantidos apenas 10. Através da figura 6.7, observamos que a diferença de métodos da primeira versão para a última, é de 1414. Entre a primeira e a última versão a API cresceu cerca de 248% no número de objetos, cerca de 493% no número de métodos e cerca de 375% no número de propriedades. Em comparação com as outras duas APIs, esta API é claramente menos estável. Observamos isto não apenas pela profusão de novos elementos, mas também porque em várias das versões desaparecem alguns elementos. Portanto, também neste caso, estas eliminações

implicam uma quebra de compatibilidade de aplicações de uma versão anterior para as versões seguintes, no caso de uso dos objetos removidos.

No caso desta API, sabe-se que o lançamento da primeira versão da versão 2 teve lugar no ano de 2006.

6.1.2 Usabilidade das APIs

Passemos agora aos resultados obtidos para cada questão proposta para os objetivos de avaliação da usabilidade das APIs em estudo. Em cada questão é apresentado um gráfico de colunas com os dados das diferentes APIs utilizadas no desenvolvimento dos protótipos, seguido de um gráfico de colunas para várias versões de cada uma das APIs.

P1 - APP. A primeira questão permite determinar o número médio de argumentos por procedimento. Relembramos que um valor APP baixo indica que a API será mais fácil de entender.

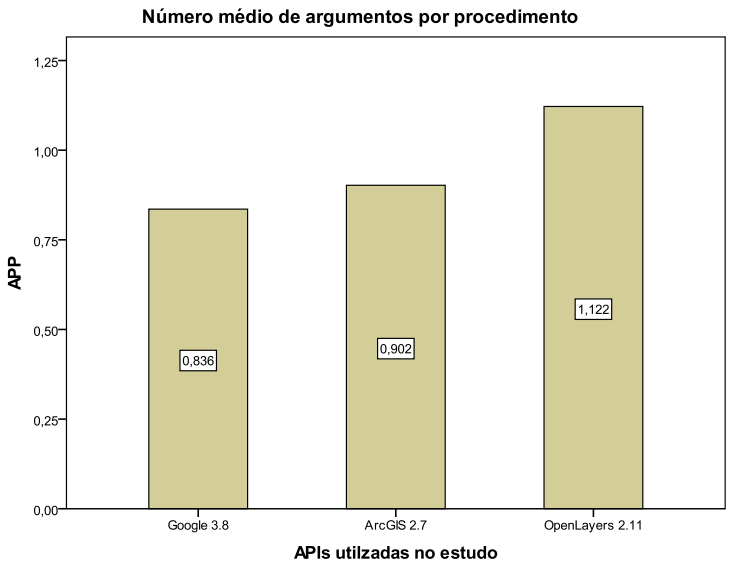

Figura 6.8: Número médio de argumentos por procedimento - APIs utilizadas para o estudo

No gráfico da figura 6.8 podemos observar que, das APIs utilizadas para o estudo, a API do Google é a que apresenta o valor de APP mais baixo e a API da OpenLayers o valor mais elevado. Portanto, a API da Google é a API mais fácil de entender.

Para avaliar se este resultado é significante, ou seja, se é improvável que tenha ocorrido por acaso, realizámos testes estatísticos [Con99] sobre estes valores de APP.

Primeiro realizámos testes de normalidade para determinar que testes de significância são mais adequados. Como as amostras são todas superiores a 50 utiliza-se o teste Kolmogorov-Smirnov para teste de normalidade.

Tabela 6.1: Teste de normalidade para o número de argumentos por método de cada API

	Kolmogorov-Smirnov		
	Estatística	gl	Sig.
Google 3.8	0,263	292	0,000
ArcGIS 2.7	0,265	1107	0,000
OpenLayers 2.11	0,267	2168	0,000

A tabela 6.1 de estatísticas apresenta o valor de estatística, os graus de liberdade (gl) e nível de significância (Sig.). A hipótese nula para o teste de normalidade afirma que a distribuição real da variável é igual à distribuição esperada, isto é, a variável é distribuída normalmente. O valor (Sig.) é que permite determinar se rejeitamos a hipótese nula.

- > 0,05 - Não significante
- 0,01 a 0,05 - Significante
- 0,001 a 0,01 - Muito significante
- ≤ 0.001 - Extremamente significante

Na tabela 6.1, observamos, nos resultados do teste Kolmogorov-Smirnov, que a probabilidade associada ao teste de normalidade (Sig.) é inferior ao nível de significância (0,001), pelo que rejeitamos a hipótese nula e concluímos que, em todas as APIs, o número de argumentos não é distribuído normalmente.

Os histogramas nas figuras 6.9, 6.10 e 6.11, apresentados de seguida, permitem confirmar que o número de argumentos, por API, não é distribuído normalmente, pois estes apresentam um valor de assimetria de distribuição (*Skewness*) diferente de 0 e um valor de achatamento da curva (*Kurtosis*) fora do intervalo -1 a 1.

Assim, o teste de significância é realizado com o teste não paramétrico Kruskal-Wallis.

Tabela 6.2: Teste de significância Kruskal-Wallis para o número de argumentos por método, de cada API

	Nºde argumentos
Chi-Quadrado	109,768
gl	2
Asymp. Sig.	0,000

O teste não paramétrico Kruskal-Wallis permite testar se as amostras são provenientes da mesma distribuição. Este teste é utilizado para a comparação de mais de duas amostras, independentes, ou não relacionadas. A hipótese nula assume que as populações

de onde as amostras originam têm a mesma média. Quando o teste de Kruskal-Wallis conduz a resultados significativos, então pelo menos uma das amostras é diferente das outras amostras.

A tabela 6.2 de estatísticas apresenta o valor do chi-quadrado (Kruskal-Wallis H), os graus de liberdade (gl) e nível de significância (Asymp. Sig.). Este último valor (Asymp. Sig.) é que permite determinar se rejeitamos a hipótese nula.

Assim, no nível de significância 0,000, existe evidência suficiente para concluir que existe uma diferença entre o número médio de argumentos por método de cada uma das APIs.

Antes de passarmos às figuras onde se apresentam os resultados da métrica APP aplicada às várias versões de cada API, é interessante apresentar os histogramas referentes à frequência do número de argumentos, relativos a cada uma das APIs utilizadas no estudo. Para cada histograma, é apresentada uma legenda que contém a média do número de argumentos, o valor do dado central (mediana), o valor estatístico que ocorre com maior frequência (moda), o valor de dispersão em relação à média (desvio padrão), o valor mínimo observado, o valor máximo observado, o valor da assimetria da distribuição (*Skewness*), o valor de achatamento da curva (*Kurtosis*) e o número de amostras (N).

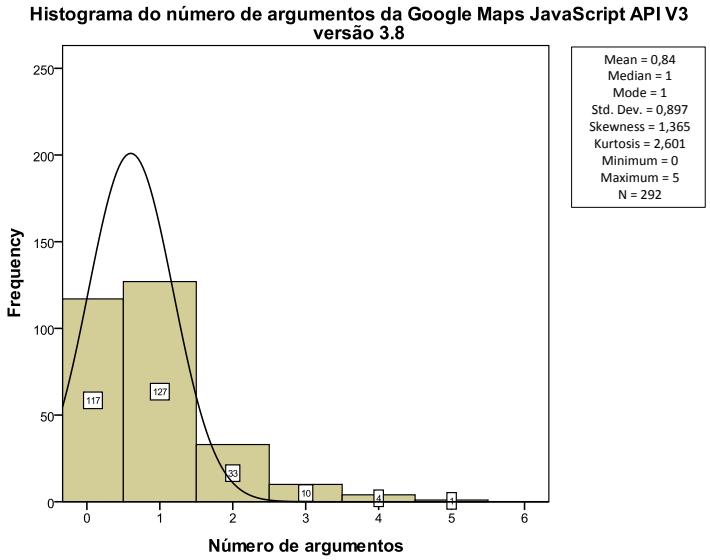

Figura 6.9: Histograma do número de argumentos da Google Maps JavaScript API V3 versão 3.8

No histograma apresentado na figura 6.9, referente à API da Google utilizada no estudo, observa-se que, de entre 292 métodos que ao todo constituem a API (incluindo os construtores), cada método tem entre 0 a 5 parâmetros. A predominância é a existência de métodos com 1 argumento.

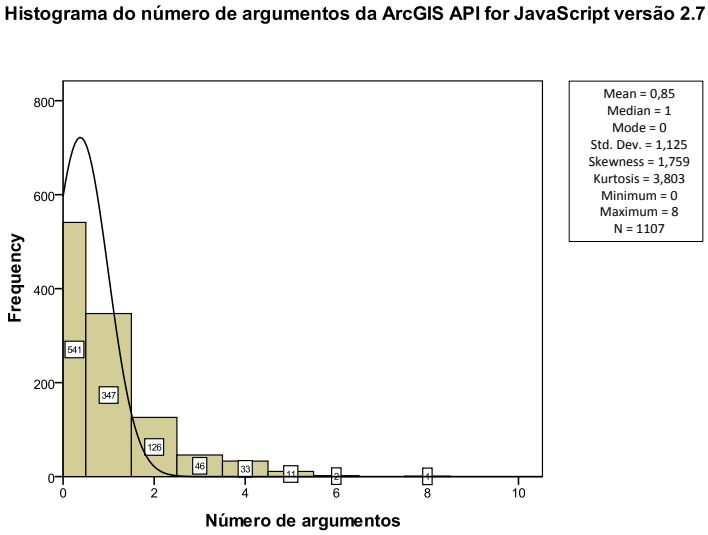

Figura 6.10: Histograma do número de argumentos da ArcGIS API for JavaScript versão 2.7

No histograma apresentado na figura 6.10, referente à API da Esri utilizada no estudo, observa-se que, de entre 1107 métodos que ao todo constituem a API (incluindo os construtores), cada método tem entre 0 a 8 parâmetros, não existindo contudo métodos com 7 argumentos. A predominância é a existência de métodos com 0 argumentos.

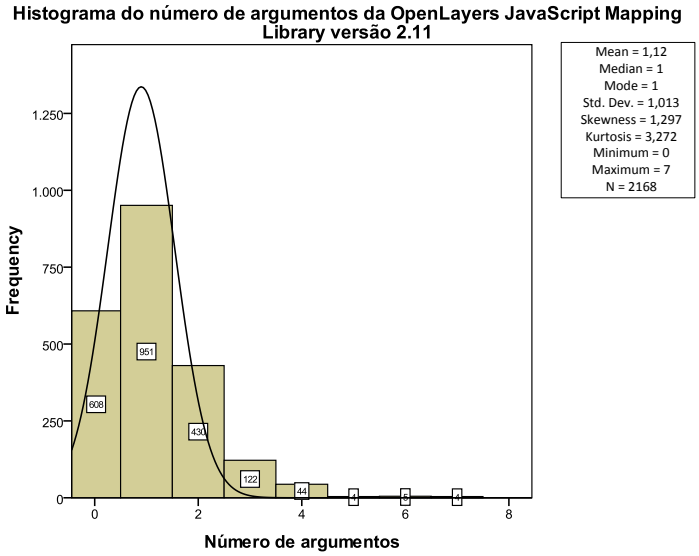

Figura 6.11: Histograma do número de argumentos da OpenLayers JavaScript Mapping Library versão 2.11

No histograma apresentado na figura 6.11, referente à API da OpenLayers utilizada no estudo, observa-se que, de entre 2168 métodos que ao todo constituem a API (incluindo os construtores), cada método tem entre 0 a 7 parâmetros. A predominância é a existência de métodos com 1 argumento.

De seguida apresentam-se os gráficos dos resultados da métrica APP aplicada às várias versões de cada API.

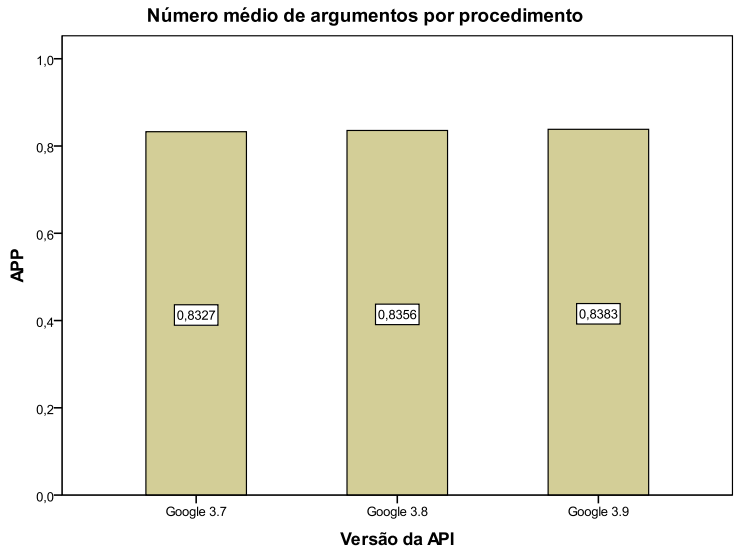

Figura 6.12: Número médio de argumentos por procedimento - Várias versões da API da Google

No gráfico da figura 6.12, podemos observar que a variação no número médio de argumentos entre as versões da API da Google é muito ligeira. Este resultado deve-se ao facto de o próprio conjunto de métodos se manter estável. Mantendo-se a maioria dos métodos, a introdução de um número reduzido de novos métodos provoca, naturalmente, um efeito reduzido no valor da média do número de argumentos em cada método.

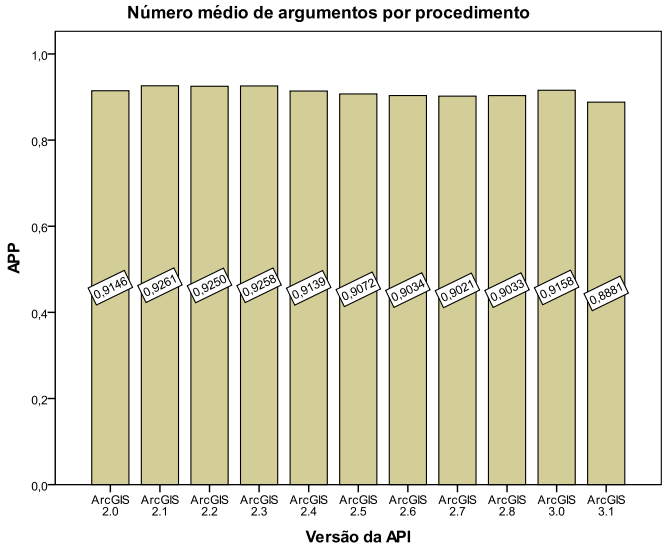

Figura 6.13: Número médio de argumentos por procedimento - Várias versões da API da Esri

No gráfico da figura 6.13, podemos observar que o número médio de argumentos varia pouco entre as versões da API da Esri. Mais uma vez, este resultado deve-se ao facto de o próprio conjunto de métodos que se manter estável. Como são essencialmente os mesmos métodos, o número médio de argumentos mantém-se estável.

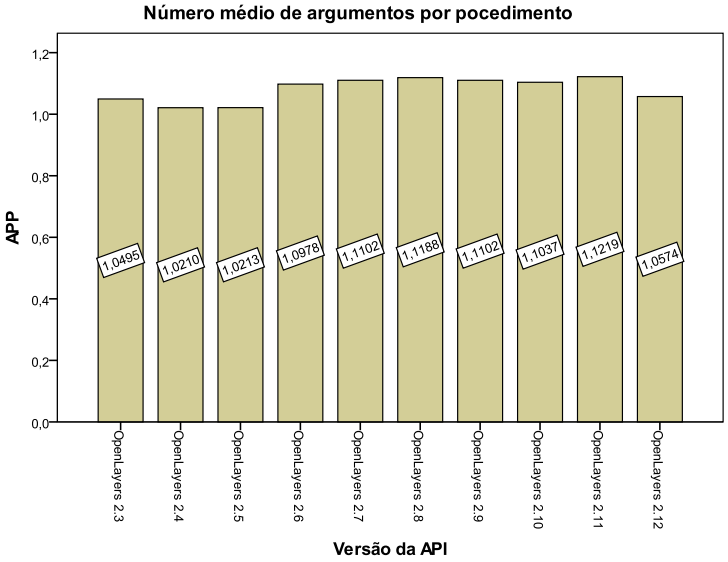

Figura 6.14: Número médio de argumentos por procedimento - Várias versões da API da OpenLayers

No gráfico da figura 6.14, podemos observar que o número médio de argumentos varia pouco entre as versões da API da OpenLayers. No entanto, em comparação com as variações das outras duas APIs, esta é menos estável, em resultado de uma maior variação do conjunto de métodos disponibilizados por esta API.

P2 - MSC. Esta questão permite determinar o valor médio de semelhança entre *strings*. Uma interface com maior valor de MSC segue, normalmente, uma convenção na atribuição de identificadores mais sistemática. Esta sistematização contribui para uma maior consistência nos identificadores, facilitando assim a compreensão da API.

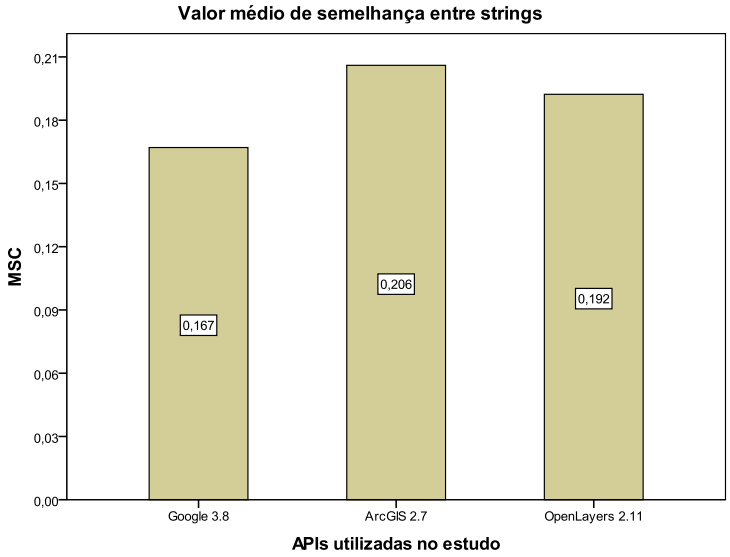

Figura 6.15: Valor médio de semelhança entre *Strings* - APIs utilizadas para o estudo

No gráfico da figura 6.15, podemos observar que das APIs utilizadas para o estudo, a API da Google é a que apresenta o valor de MSC mais baixo e a API da Esri o valor mais elevado. Podemos então concluir que, segundo a métrica utilizada, a API da Esri é a que tem uma abordagem de definição de identificadores mais sistemática. Para determinarmos a significância destes valores, procedemos da mesma forma que na primeira métrica, determinamos primeiro a normalidade da amostra para verificar que teste de significância é mais adequado.

Tabela 6.3: Teste de normalidade para o valor de semelhança entre *strings*

Estatística	
N	37767368
Média	0.20312
Mediana	0.20000
Moda	0.200
Desvio Padrão	0.106335
Skewness	1.237
Kurtosis	6.426
Mínimo	0.000
Máximo	1.000

Através dos resultados da tabela de estatística 6.3, observamos que o valor de semelhança entre *strings* não é distribuído normalmente, pois estes apresentam um valor de assimetria de distribuição (*Skewness*) diferente de 0 e um valor de achatamento da curva (*Kurtosis*) fora do intervalo -1 a 1.

Assim, o teste de significância é realizado com o teste não paramétrico Kruskal-Wallis.

Tabela 6.4: Teste de significância Kruskal-Wallis para o valor de semelhança entre *strings*, de cada API

	Semelhança entre *strings*
Chi-Quadrado	370,800
gl	2
Asymp. Sig.	0,000

A tabela 6.4 de estatísticas apresenta o valor do chi-quadrado (Kruskal-Wallis H), os graus de liberdade (gl) e nível de significância (Asymp. Sig.). Ao nível de significância 0,000, existe evidência suficiente para concluir que, entre as APIs, existe uma diferença significativa no valor de semelhança entre *strings*.

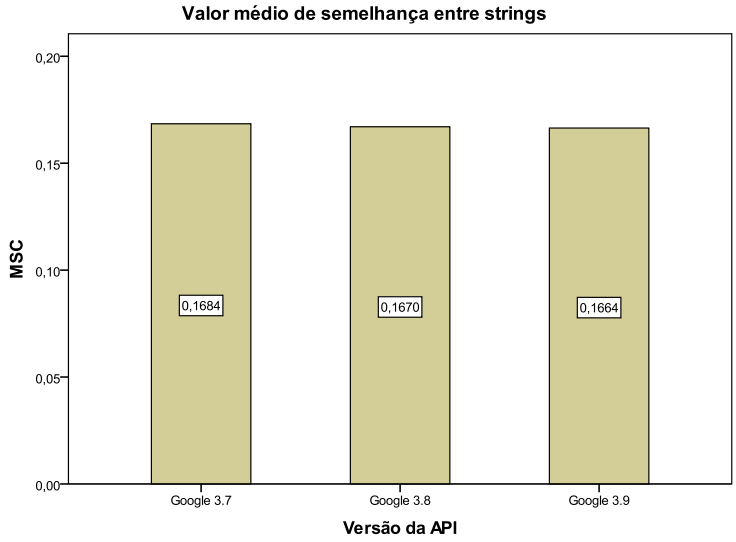

Figura 6.16: Valor médio de semelhança entre *Strings* - Várias versões da API da Google

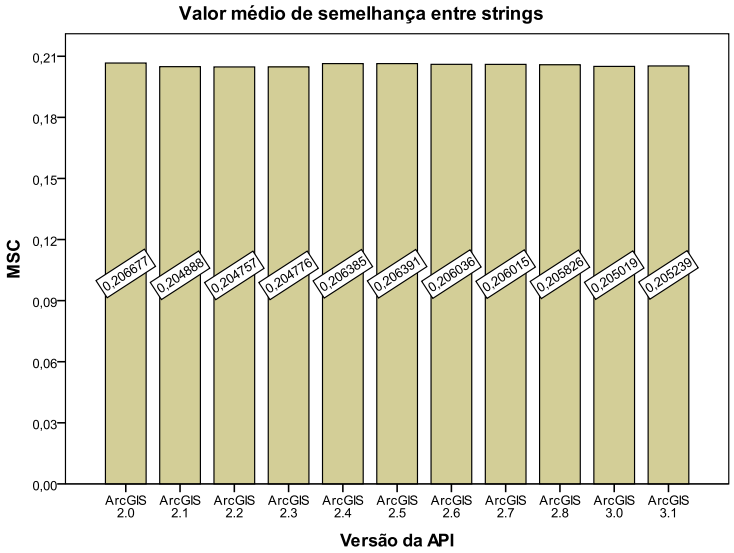

Figura 6.17: Valor médio de semelhança entre *Strings* - Várias versões da API da Esri

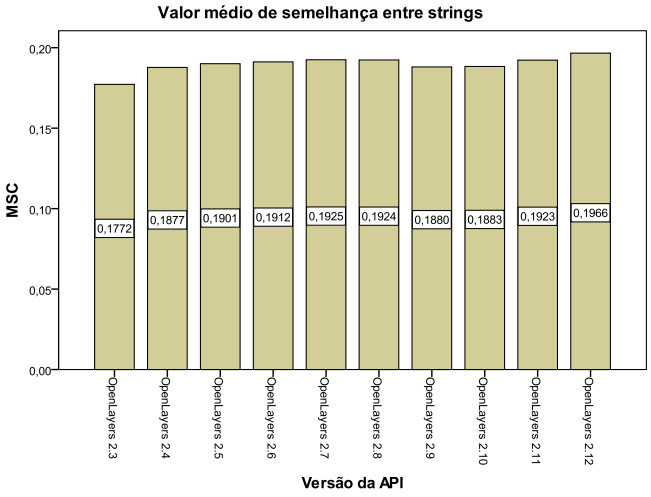

Figura 6.18: Valor médio de semelhança entre *Strings* - Várias versões da API da OpenLayers

Nos gráficos das figuras 6.16, 6.17 e 6.18, podemos observar que entre as várias versões de cada API, o valor médio de semelhança entre *strings* varia pouco. Este resultado deve-se ao facto de o próprio número de métodos, das respetivas APIs se manter estável. No entanto, tal como acontece com o número médio de argumentos por procedimento, a API da OpenLayers é menos estável, em comparação com as outras duas APIs, em resultado de uma maior variação no conjunto de métodos.

P3 - MIL. Esta métrica permite determinar o valor médio do comprimento dos identificadores. As interfaces com maior valor de MIL são mais auto-documentadas.

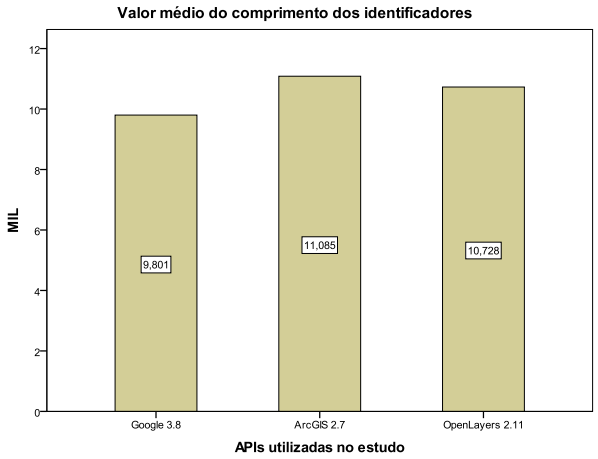

Figura 6.19: Valor médio do comprimento dos identificadores - APIs utilizadas para o estudo

A dimensão dos nomes atribuídos pode indicar o nível de auto-documentação da API pois, teoricamente, nomes maiores contêm mais informação e uma API com identificadores maiores tende a ser mais auto-documentada. Os resultados obtidos para a métrica que permite pedir esta propriedade, presentes na figura 6.19, mostram que a API mais auto-documentada é a API da Esri e que a API menos auto-documentada é a API da Google. Para determinarmos a significância destes valores, procedemos da mesma forma que nas métricas anteriores, determinamos primeiro a normalidade da amostra para verificar que teste de significância é mais adequado. Assim como nas métricas anteriores, as amostras são todas superiores a 50 pelo que, também aqui, se utiliza o teste Kolmogorov-Smirnov para teste de normalidade.

Tabela 6.5: Teste de normalidade para o comprimento dos identificadores de cada API

	Kolmogorov-Smirnov		
	Estatística	gl	Sig.
Google 3.8	0,114	973	0,000
ArcGIS 2.7	0,124	7835	0,000
OpenLayers 2.11	0,097	3635	0,000

Na tabela 6.5, observamos, nos resultados do teste Kolmogorov-Smirnov, que a probabilidade associada ao teste de normalidade (Sig.) é inferior ao nível de significância (0,001), pelo que rejeitamos a hipótese nula de igualdade entre médias e concluímos que, em todas as APIs, o comprimento dos identificadores não é distribuído normalmente. Assim, realizamos o teste de significância com o mesmo teste não paramétrico utilizado anteriormente, Kruskal-Wallis:

Tabela 6.6: Teste de significância Kruskal-Wallis para o comprimento dos identificadores de cada API

	Comprimento dos identificadores
Chi-Quadrado	42,604
gl	2
Asymp. Sig.	0,000

A tabela 6.6 de estatísticas apresenta o valor do chi-quadrado (Kruskal-Wallis H), os graus de liberdade (gl) e nível de significância (Asymp. Sig.). Mais uma vez, ao nível de significância 0,000, existe evidência suficiente para concluir que, entre as APIS, existe uma diferença significativa no comprimento dos identificadores.

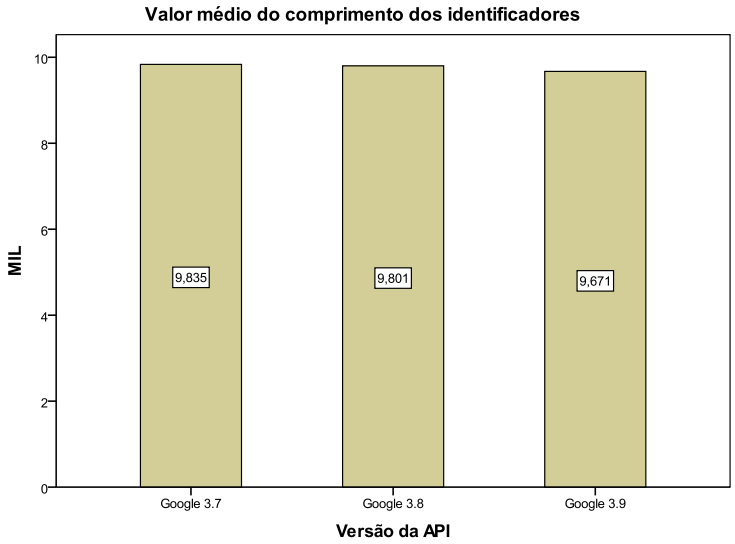

Figura 6.20: Valor médio do comprimento dos identificadores - Várias versões da API da Google

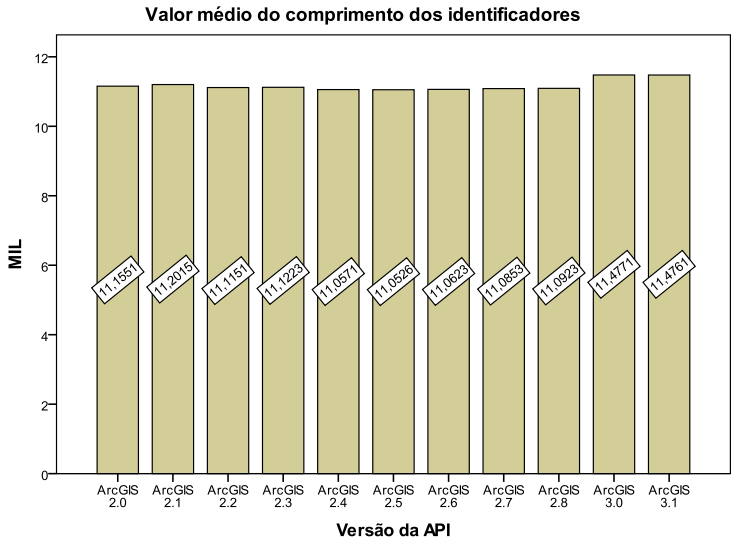

Figura 6.21: Valor médio do comprimento dos identificadores - Várias versões da API da Esri

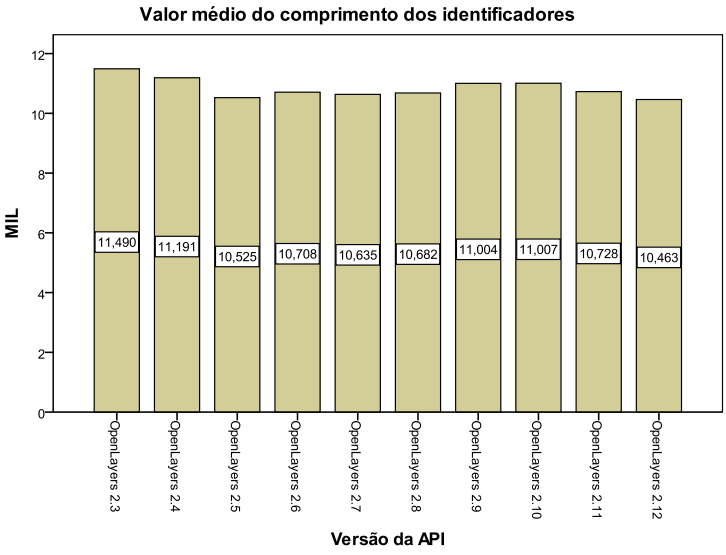

Figura 6.22: Valor médio do comprimento dos identificadores - Várias versões da API da OpenLayers

Nos gráficos das figuras 6.20, 6.21 e 6.22, podemos observar que, entre as várias versões de cada API, o valor médio do comprimento dos identificadores varia pouco. Para este resultado contribui o facto de o próprio conjunto de métodos, por API, se manter estável ao longo do tempo. No entanto, tal como acontece com o número médio de argumentos por procedimento e com o valor médio de semelhança entre strings, a API da OpenLayers é menos estável, em comparação com as outras duas APIs, em resultado de uma maior variação no conjunto de métodos.

6.1.3 Avaliação da utilização efetiva baseada nos protótipos desenvolvidos

Até este ponto apresentámos os resultados obtidos sobre a avaliação direta das APIs. Prossigamos para os resultados obtidos relativos a avaliação da utilização das APIs.
P4- APIUI. Permite determinar o número de chamadas à API. Desta forma, pretendemos analisar se a implementação de determinadas funcionalidades do protótipo, para uma determinada API, é mais complexa que com outra API. Para a apresentação dos dados obtidos utilizamos o gráfico de boxplot e um gráfico de barras.

O gráfico *boxplot* (também conhecido como *box and whisker plot*) resume as seguintes medidas estatísticas: mediana; quantis superior e inferior; valores mínimos e máximos. Assim, os gráficos *boxplot* são compostos por:

- **Caixa ou *box*:** Contém os valores entre o percentil 25 e o percentil 75 da amostra e é constituída por: primeiro quartil (Q1) – correspondente ao limite inferior da caixa;

segundo quartil (Mediana) – correspondente ao centro da amostra (delimitada pelos bigodes); terceiro quartil (Q3) – correspondente ao limite superior da caixa. Se a linha mediana dentro da caixa não é equidistante dos extremos, diz-se então que os dados são assimétricos;

- **Bigodes ou *whiskers*:** Representam os valores mínimo e máximo , a menos que valores *outliers* estejam presentes, nesse caso o gráfico estende-se ao máximo de 1.5 vezes da distância interquartil. Numa amostra normal, cerca de 95% dos dados estão entre os bigodes;

- **Valores atípicos ou *outlier*:** correspondem aos valores da amostra que se encontram desfasados dos restantes valores, e podem ser de dois tipos: atípicos – valores que se encontram à distância de uma caixa e meia do quartil inferior ou do quartil superior (representados por círculos); extremos – valores que se encontram à distância de três caixas do quartil inferior ou do quartil superior (representados por asteriscos).

Passemos então aos gráficos obtidos.

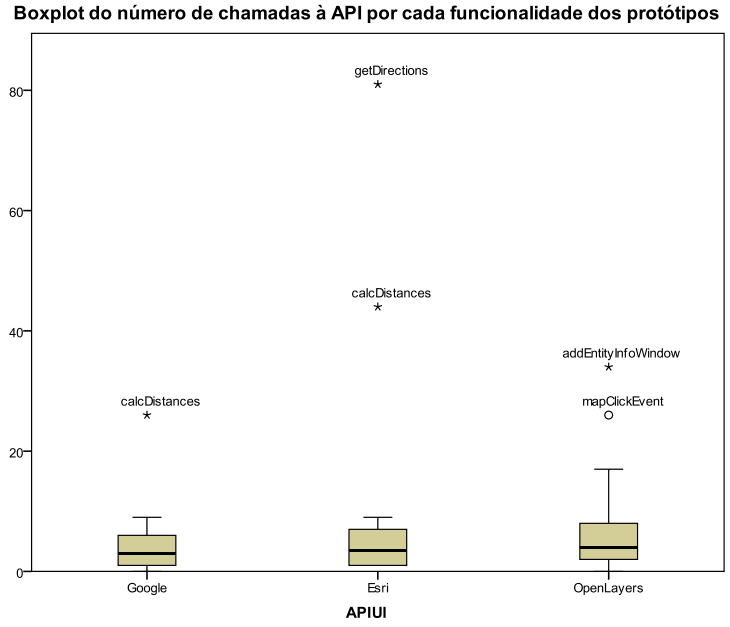

Figura 6.23: Número de chamadas à API

No gráfico da figura 6.23, que representa um gráfico *boxplot* do número de chamadas à API por cada método que compõe o protótipo, podemos observar pela mediana (3 para a API da Google, 3.5 para a API da Esri e 4 para a API da OpenLayers) que os métodos do protótipo desenvolvido com a API da OpenLayers necessitam, em média, de mais chamadas à API.

Quanto a valores atípicos e extremos, observa-se que no protótipo desenvolvido com a API da Google e no protótipo desenvolvido com a API da Esri existe um método em comum, `calcDistances`, com valor extremo. No protótipo desenvolvido com a API da Esri observa-se ainda outro extremo, no método `calcDirections`.

Por fim, no protótipo desenvolvido com a API da OpenLayers observa-se um valor atípico no método `mapClickEvent`, e um valor extremo no método `addEntityInfoWindow`.

Estes valores atípicos e extremos, indicam que as implementações destes métodos foram as que envolveram mais chamadas às APIs, nos respetivos protótipos.

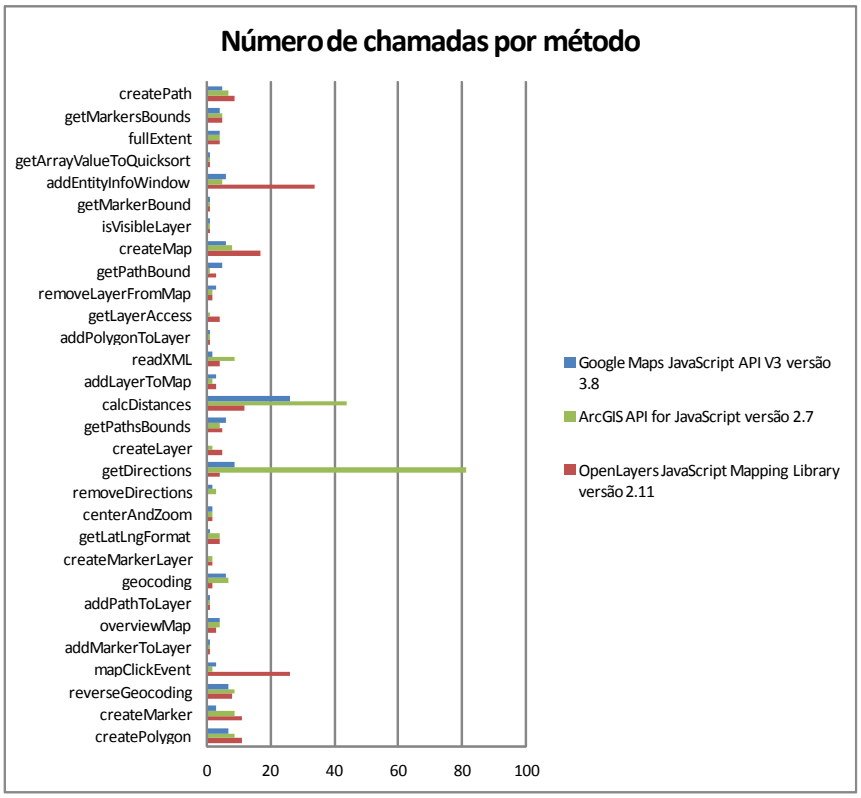

Figura 6.24: Número de chamadas à API

No gráfico da figura 6.24, apresentamos, num gráfico de barras, o número de chamadas necessárias em cada API, por funcionalidade do protótipo. Aqui podemos constatar os valores atípicos extremos visualizados no gráfico de *boxplot*. Os métodos que se destacavam por um número maior de chamadas no gráfico de *boxplot*, são os mesmo que se visualizam com maior número de chamadas no gráfico de barras. No geral, o protótipo desenvolvido com a API da Google, em comparação com as outras, necessita de menos chamadas à mesma. Por contraste, podemos observar que, no gráfico da figura 6.24, a barra que corresponde às chamadas realizadas com a API da OpenLayers, ultrapassa mais vezes as outras barras, reforçando o que observámos primeiro no gráfico de *boxplot*

na figura 6.23, ou seja, que a API da OpenLayers, em média, necessita de mais chamadas por funcionalidade. No entanto, tal como verificámos para a outras métricas, também aqui é importante verificar se a diferença entre o número de chamadas por funcionalidade do protótipo, para cada API, é significante.

Ao contrário das outras métricas analisadas, neste caso todas as amostras são inferiores a 50, o que leva à utilização do teste de Shapiro-Wilk.

Tabela 6.7: Teste de normalidade para o número de chamadas dos identificadores por funcionalidade do protótipo para cada API

	Shapiro-Wilk		
	Estatística	gl	Sig.
Google 3.8	0,642	30	0,000
ArcGIS 2.7	0,424	30	0,000
OpenLayers 2.11	0,694	30	0,000

Da mesma forma, o teste Shapiro-Wilk assume que a distribuição real da variável é igual à distribuição esperada, isto é, a variável é distribuída normalmente, como sendo a hipótese nula para o teste de normalidade.

Na tabela 6.7, observamos, nos resultados do teste Shapiro-Wilk, que a probabilidade associada ao teste de normalidade (Sig.) é inferior ao nível de significância (0,001), pelo que rejeitamos a hipótese nula e concluímos que, em todas das APIs, o número de chamadas não é distribuído normalmente. Assim, mais uma vez, o teste de significância é realizado com o teste não paramétrico, Kruskal-Wallis.

Tabela 6.8: Teste de significância Kruskal-Wallis para o número de chamadas dos identificadores por funcionalidade do protótipo para cada API

	Nºde chamadas
Chi-Quadrado	1,292
gl	2
Asymp. Sig.	0,524

A tabela 6.8 de estatísticas apresenta o valor do chi-quadrado (Kruskal-Wallis H), os graus de liberdade (gl) e nível de significância (Asymp. Sig.). Neste caso o nível de significância é 0,524, e assim não podemos afirmar com segurança que as diferenças no número de chamadas dos identificadores por funcionalidade do protótipo para cada API, sejam significativas.

6.1.4 Avaliação da utilização efetiva baseada nas aplicações suportadas pela API da Google

Além dos protótipos desenvolvidos, e que permitiram a análise, realizada até este ponto, da utilização efetiva das APIs em estudo, também é possível realizar uma análise da utilização efetiva particular para a API da Google. Esta análise é possível devido ao acesso a aplicações WEB-SIG desenvolvidas por outros programadores que compreendem requisitos semelhantes aos dos protótipos. Assim, no gráfico da figura 6.25, apresentamos os dados relativos à utilização particular da API da Google, resultante de 10 aplicações desenvolvidas por programadores diferentes.

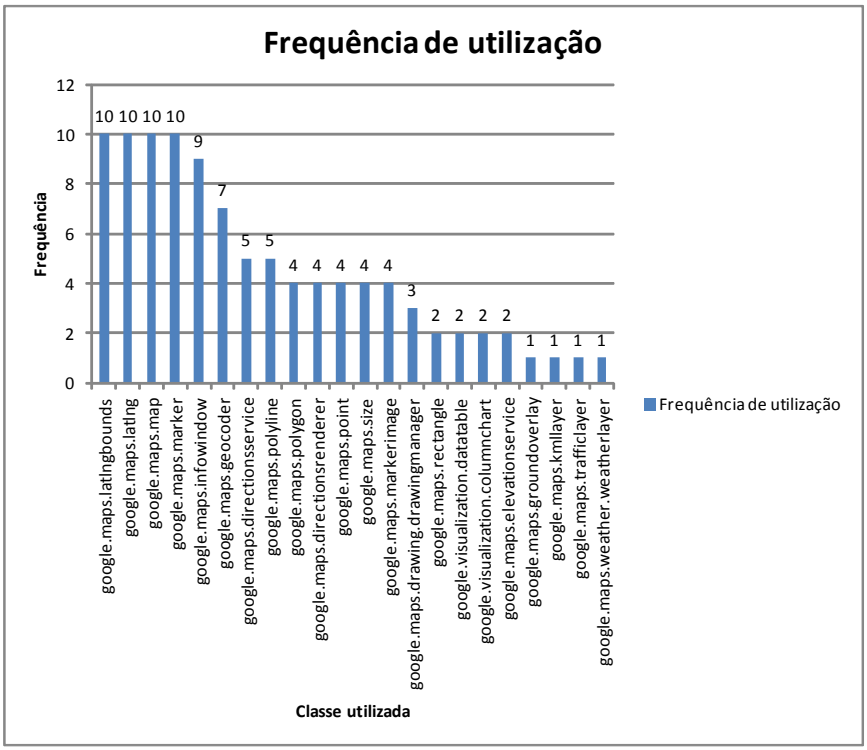

Figura 6.25: Número de chamadas à API

O gráfico da figura 6.25 mostra que, do conjunto diferente de objetos utilizados em todas as aplicações, no mínimo, cada um dos objetos foi utilizado pelo menos num dos trabalhos. No máximo, cada objeto pode ter sido incluído em todas as aplicações, e portanto o valor da sua frequência é 10. Existem 4 objetos da API que foram utilizados em todas as aplicações e 4 que foram utilizados apenas numa aplicação. Este resultado indica que, para o desenvolvimento de aplicações baseadas nos mesmos requisitos, os objetos utilizados em todos os trabalhos são os mais importantes. Desta forma, um principiante em aplicações Web-SIG deverá começar por aprender a utilizar estes objetos.

6.2 Discussão

Na secção anterior foram apresentados os dados obtidos tanto para as três APIs efetivamente utilizadas no estudo desta dissertação, como para várias versões de cada uma destas APIs. No entanto, o principal objetivo é comparar a três APIs utilizadas, pelo que primeiro iremos discutir os dados apresentados anteriormente relativos a comparações entre as três APIs. Depois de discutidos estes dados, discutimos os dados apresentados relativos às várias versões de cada API.

6.2.1 Discussão relativa à comparação entre as APIs efetivamente utilizadas

Através da figura 6.1, observamos que existem grandes diferenças entre as dimensões das APIs. A API da Google tem cerca de metade do número de objetos, cerca de 13% do número de métodos e cerca de 47% do número de propriedades, quando comparada com a API da OpenLayers. Quando comparada com a API da Esri a diferença já só é mais notada em relação ao número de métodos. A API da Google tem cerca de 26% do número de métodos da API da Esri, enquanto que em relação ao número de objetos e ao número de propriedades as diferenças rondam os 70%.

Apesar da diferença de objetos entre a API da Google e a API da Openlayers ser grande, não é nos objetos que os valores são mais acentuados. Isto porque, como podemos observar na diferença para a API da Esri, apesar de a diferença entre o número de objetos não ser grande, a diferença entre o número de métodos já o é, ou seja, o número de objetos até podia ser igual, mas o que interessa é a densidade do número de métodos e de propriedades, que pode ser bastante grande dentro dos objetos. Portanto, em termos de dimensões, a API da OpenLayers é a maior, seguida da API da ESRI, sendo a API da Google é a que tem menores dimensões.

Em relação às dimensões, sabemos que ter mais opções aumenta o grau de liberdade, e consequentemente a escolha. Ao termos mais objetos, temos mais complexidade mas também temos mais flexibilidade. Uma complexidade maior leva a uma curva de aprendizagem da API maior, e ao termos mais flexibilidade, a escolha é dificultada, e portanto pode ser mais difícil utilizar a API. A prazo, esta flexibilidade extra pode ser uma mais valia, sobretudo para programadores experientes.

As métricas utilizadas para os resultados obtidos, permitem, como já referido anteriormente, avaliar as APIs na sua facilidade de compreensão e facilidade de utilização.

Comecemos pelas métricas que avaliam a facilidade de compreensão.

Com a métrica APP avaliamos o número médio de argumentos por método, que, como referido no capítulo 4, está relacionado com a capacidade humana de retenção de informação. O limite são cerca de sete itens, mas mesmo que nenhuma das APIs tenha métodos com mais de sete itens, quantos menos argumentos existirem mais fácil será reter essa informação.

No gráfico da figura 6.8, observamos que existem diferenças nas médias de argumentos por método de cada API, e verificámos que essa diferença é significativa. No entanto, apesar da diferença ser significativa, na prática, a diferença dos valores obtidos é de magnitude reduzida, pois, entre as APIs, não há muita diferença no número de argumentos que é mais frequente. Na API da Google e na OpenLayers é mais frequente existirem métodos só com um argumento, enquanto que na API da Esri é mais frequente existirem métodos com zero argumentos, pelo que o efeito na facilidade de compreensão, a existir, é, deste ponto de vista, marginal, pois são valores muito próximos.

Com a métrica MSC avaliamos o efeito da sistematização na abordagem de nomes. Existindo uma abordagem mais sistemática, significa que existe uma convenção de nomes adotada, o que facilita a compreensão. No gráfico da figura 6.15, observamos que a magnitude das diferenças obtidas é reduzida, pelo que, também neste caso, o efeito na facilidade de compreensão, a existir, não é relevante, pois os valores são muito próximos.

Por último, nas métricas de avaliação da facilidade de compreensão, temos a métrica MIL para avaliar o efeito do comprimento médio dos identificadores. Existindo um maior comprimento médio dos identificadores, a API tem tendência a ser mais autodocumentada. No entanto, tal como nas outras métricas, apesar de existir uma diferença significativa, na prática, a diferença dos valores obtidos é de magnitude reduzida, pelo que, mais uma vez, o efeito na facilidade de compreensão, a existir, é, deste ponto de vista, marginal, pois são valores muito próximos.

Portanto, no que diz respeito à avaliação da facilidade de compreensão, o efeito dos critérios avaliados é pouco relevante. Não é possível distinguir das três APIs qual a mais fácil de compreender.

Analisados os resultados referentes à facilidade de compreensão, ainda temos os resultados relativos à facilidade de utilização. Aqui existem duas avaliações distintas. Temos a avaliação aos resultados obtidos para os protótipos desenvolvidos no contexto da dissertação, e temos os resultados obtidos para as aplicações, que cumprem determinados requisitos mínimos, desenvolvidas, por programadores diferentes, com suporte da API da Google.

Na avaliação da utilização, baseada nos protótipos, observamos que, no gráfico na figura 6.23, é possível observar, pela mediana, que a média do número de chamadas necessárias para realizar as funcionalidades dos protótipos, é ligeiramente maior na API da OpenLayers. No entanto, não é possível afirmar que a diferença é significativa. Ao observarmos o gráfico da figura 6.24, constatamos que a barra de cor vermelha ultrapassa mais vezes as outras barras, indicando o que observámos primeiro no gráfico de *boxplot* 6.23, ou seja, que a API da OpenLayers necessita, em média, de mais chamadas por funcionalidade.

Quanto aos valores extremos e atípicos, estes confirmam a complexidade de implementação sentida na implementação dos protótipos. De todas as funcionalidades implementadas, as que tipicamente requerem maior complexidade são as funcionalidades referentes às distâncias e direções (`calcDistances` e `calcDirections`, respetivamente).

Como se observa no gráfico de barras 6.24, o número de chamadas é realmente mais elevado para a API da Google e da Esri, enquanto que na API da OpenLayers o número de chamadas nessas funções (`calcDistances` e `calcDirections`) é baixa, porque a API não suporta funcionalidades de *geocoding*, e portanto para esta API estas funcionalidades foram implementadas com as funcionalidades da API da Google. O número de chamadas observadas para estas funções no protótipo da API da OpenLayers, são as chamadas necessárias para algumas conversões para coordenadas que o mapa da OpenLayers reconhece. No entanto, para a API da OpenLayers observa-se que, nas funcionalidades `mapClickEvent` e `addEntityInfoWindow`, consideradas funcionalidades de fácil implementação, existe um número de chamadas mais elevado relativamente às outras APIs.

Observando apenas o gráfico da figura 6.24, é possível constatar que a API da Google, relativamente às outras duas, na generalidade das funcionalidades do protótipo, exige menos chamadas à API. Portanto, podemos afirmar que, para esta aplicação em particular, a API do Google é a que responde melhor as necessidades do programador.

Analisados e comparados todos os resultados obtidos através das métricas, é possível concluir que, através das métricas para avaliação da facilidade de compreensão, não conseguimos distinguir qual a API mais fácil de compreender. Contudo, a partir da comparação entre as dimensões das APIs, e dos resultados obtidos com as métricas para avaliação da facilidade de utilização das APIs, é possível concluir que a questão da relação entre a dimensão, o aumento da complexidade e o aumento da flexibilidade, que têm como consequência o aumento da dificuldade de compreensão e da dificuldade de utilização, respetivamente, é confirmada pelos resultados obtidos na utilização das APIs. A API da OpenLayers, por ser de maior dimensão, obriga a um maior esforço no processo de escolha dos objetos adequados para cada tarefa. A maior diversidade de opções resulta, também, numa maior dificuldade na escolha dos objetos mais adequados, particularmente nos casos em que existem objetos com funcionalidades bastante aproximadas. Pelo contrário, numa interface mais reduzida, como a da Google, os objetos disponibilizados oferecem, frequentemente, um nível de abstração ligeiramente superior, o que facilita a aprendizagem e utilização da API. Um sintoma dessa maior facilidade de utilização resultante de um mais adequado nível de abstração é o menor número de chamadas de que nos tivemos de socorrer, ao implementar o protótipo com o suporte da API da Google, por comparação com as que tivemos de fazer no protótipo construído com o suporte API da OpenLayers.

Quanto à avaliação da utilização, baseada nas aplicações, é possível observar no gráfico da figura 6.25, que existem 4 objetos utilizados em todos os trabalhos, o que sugere que estes objetos, `google.maps.latlng`, `google.maps.latlngbounds`, `google.maps.map` e `google.maps.marker`, são mais importantes para aplicações baseadas nos mesmos requisitos mínimos. Quanto aos objetos que só foram utilizados numa aplicação, podemos especular que são específicas ao tópico do trabalho.

Tendo em conta, que um dos desafios na adoção de uma API é saber por onde começar a sua aprendizagem, estes dados permitem identificar quais os objetos de utilização

mais frequente, numa aplicação baseada nos mesmos requisitos, que deveriam constituir uma prioridade na aprendizagem da API, por novos programadores. Depois de aprendidos os objetos mais prioritários, a ordem de frequência com que os restantes objetos são utilizados, sugere a ordem em que estes devem ser aprendidos.

6.2.2 Discussão relativa às várias versões de cada API

Relativamente aos resultados apresentados para as versões de cada uma das APIs, pretendemos com esses valores avaliar a evolução das APIs.

Entre 1974 e 1996 Lehman e Belady [Leh78, Leh79] propuseram um número de leis aplicadas relativamente à evolução de software. Com o objetivo de definir uma teoria unificada para a evolução de software, os dois investigadores examinaram o crescimento e evolução de vários sistemas de grande porte. O estudo resultou na formulação de oito leis que permitem avaliar a evolução de software. No entanto, destas oito leis apenas três são adequadas de acordo com os dados estudados e disponíveis:

- **Mudança contínua (1)** - Os sistemas devem ser continuamente adaptados, ou tornam-se progressivamente menos satisfatórios.
- **Aumento da complexidade(2)** — À medida em que um sistema evolui, a sua complexidade aumenta, a não ser que seja desenvolvido trabalho para a manter ou para reduzir.
- **Crescimento contínuo(6)** - O conteúdo funcional dos sistemas deve ser continuamente aumentado para manter a satisfação do utilizador.

Observando as figuras 6.2, 6.4 e 6.6, podemos verificar a primeira lei de evolução na API da Esri e na API da OpenLayers, pois as mudanças observadas na eliminação e adição de objetos sugerem a adaptação das APIs ao longo do tempo, onde alguns dos objetos eliminados podem corresponder a objetos que se tornaram obsoletos, devido, por exemplo, à adaptação para novos formatos mais utilizados. Quanto à segunda e à sexta lei, podemos verificar a evolução em todas as APIs, pois entre as primeiras e últimas versões observa-se claramente o aumento da dimensão das APIs, assumindo que uma API se torna mais complexa ao adquirir mais objetos, métodos e propriedades, e portanto aumenta, e que ao aumentar tanto o número de objetos, como o número de métodos e propriedades, também o seu conteúdo funcional aumenta. Portanto, através destas leis verifica-se que as APIs estão a evoluir.

Relativamente à facilidade de compreensão entre as várias versões de cada API, observando as figuras 6.12, 6.13, 6.14, 6.16, 6.17, 6.18, 6.20, 6.21 e 6.22, é possível constatar que se mantêm estáveis no número médio de argumentos, no valor médio de semelhança entre *strings* e no comprimento médio do identificadores, de versão para versão.

Assim, com a verificação através destas leis confirmamos que as APIs estão a evoluir e que a facilidade de compreensão entre versões é estável. No entanto, quanto a questões de retro-compatibilidade, na API da Google, por não se verificar remoção de objetos entre

as versões analisadas, é fácil a migração de uma versão da API para a versão seguinte. Nas APIs da OpenLayers e da Esri, por se verificarem remoções de objetos entre versões, há uma quebra de compatibilidade de aplicações de uma versão anterior que usassem esses objetos para as versões seguintes. Este fator pode portanto afetar a usabilidade destas duas APIs.

6.3 Ameaças à validade

É importante perceber as possíveis formas de ameaças à validade do estudo desenvolvido. Estas ameaças são diferentes para as diferentes formas de validade relacionadas com experiências. As categorias de validade são: a validade de construção, a validade interna e a validade externa [LB03, SCC02].

A validade de construção questiona se a abordagem seguida realmente permite alcançar o objetivo apresentado. No caso deste estudo, a pesquisa sobre o trabalho relacionado e a informação que daí foi recolhida, aliada à abordagem seguida permite ir ao encontro das necessidades do programador no que respeita à escolha entre APIs de mapas, pois através dos resultados obtidos com a aplicação das métricas às APIs e aos protótipos, é possível realizar comparações, e assim informar o utilizador.

A validade interna questiona o desenho da experiência em si. Nesta categoria o que pode ameaçar a validade interna é o facto de só ser utilizada uma métrica para avaliar cada uma das propriedades abordadas(ameaça conhecida como *mono-operation bias*).

Por fim, a validade externa, questiona se a abordagem permite ou não generalizar. Esta categoria é uma ameaça presente neste estudo, pois as aplicações criadas não permitem uma generalização a todas as APIs de mapas, e existe a limitação da experiência em programar com as APIs escolhidas.

6.4 Limitações

As maiores limitações presentes nesta dissertação, estão relacionadas com a avaliação da utilização efetiva das APIs, pois não tendo sido encontrados repositórios de aplicações, que abordassem o mesmo contexto, para cada uma das APIs, decidiu-se implementar três protótipos com as mesmas funcionalidades. É neste ponto que encontramos as limitações. Caso existissem recursos disponíveis para vários programadores diferentes implementarem os protótipos, a quantidade de dados obtidos seria maior, conferindo desse modo, maior segurança às conclusões a retirar da avaliação realizada.

A outra limitação é o facto de, por se considerar um determinado contexto de aplicação, as conclusões retiradas não consideram todas as funcionalidades oferecidas pelas APIs.

Por fim, a experiência prévia na utilização das APIs em questão, pode ter influenciado a escolha e as opções tomadas durante a implementação dos protótipos.

7

Conclusão

Neste capítulo apresentamos de forma sucinta as contribuições desta dissertação e discutimos possíveis trabalhos futuros que deverão contribuir para um avanço no estado da arte da avaliação de usabilidade de APIs de mapas.

7.1 Resumo

O objetivo desta dissertação é comparar a usabilidade de diferentes APIs de mapas. Para explorar esta área, é necessário avaliar fatores relacionados com a facilidade de compreensão, e facilidade de utilização. Foram escolhidas as APIs Google Maps JavaScript API, ArcGIS API for JavaScript e OpenLayers JavaScript Mapping Library, por se considerarem exemplos relevantes de APIs para mapas para as comunidades empresarial e *open source*, e com o suporte destas, desenvolvemos três protótipos.

Através dos protótipos foi possível realizar uma análise qualitativa e uma análise quantitativa, entre as três APIs.

Da análise qualitativa, as observações mais relevantes vão para o facto de, ao contrário da outras, a API da Google não permitir a manipulação de *layers* de entidades. Permite a manipulação dos *layers*, `google.maps.KmlLayer`, para o suporte dos formatos de dados KML e GeoRSS para a exibição de informação geográfica, `google.maps.BicyclingLayer`, para apoiar a construção de funcionalidades dedicadas a ciclistas, `google.maps.FusionTablesLayer`, para o processamento dos dados contidos nas FusionTables do Google como um *layer* num mapa , e `google.maps.TrafficLayer`, para apoiar a construção de funcionalidades baseadas em informação de trânsito em tempo real (quando suportado), mas não permite manipular, por exemplo, *layers* de marcadores. Para tal, é necessário recorrer a vetores de forma

a simular *layers*.

Outra observação bastante relevante, é o facto de a API da OpenLayers não suportar o processo de georreferenciação, que por sinal é bastante importante em aplicações desta natureza. Portanto, caso um programador prefira por alguma razão utilizar a API da OpenLayers, poderá implementar esta funcionalidade com o auxilio, por exemplo, da utilização da API do Google.

Para a análise quantitativa, selecionámos métricas adequadas à comparação da usabilidade das diferentes APIs, que depois foram recolhidas nos três protótipos, permitindo assim, fazermos uma avaliação comparativa da utilização das APIs. Da aplicação das métricas obtivemos as seguintes conclusões apresentadas nos seguintes parágrafos.

Constatámos que a API da Google é consideravelmente mais pequena que as outras duas APIs, sendo a API da OpenLayers a maior. É expectável que uma maior dimensão da API implique maior dificuldade na sua aprendizagem, compreensão e utilização, uma vez que a complexidade de uma API é frequentemente associada à sua dimensão.

Das métricas específicas para a avaliação da facilidade de compreensão, como o número médio de argumentos por método, o valor médio de semelhança entre *strings* e o comprimento médio dos identificadores, não foi possível concluir de forma segura qual das APIs é mais fácil de compreender, apesar de termos encontrado diferenças estatisticamente significativas entre as várias APIs, para várias destas métricas. Isto deveu-se ao facto de apesar de estatisticamente significativas, essas diferenças serem bastante reduzidas. Deste modo, o impacto real dessas diferenças na facilidade de utilização é provavelmente negligenciável. Estas diferenças reduzidas que resultam de aspetos de estilo nas definições das APIs traduzem certamente uma maturidade das comunidades que as desenvolvem e não será completamente surpreendente, dado termos escolhido 3 das APIs mais relevantes no panorama da programação de aplicações de mapas. Contudo, dos dados resultantes da aplicação da métrica de avaliação da utilização, que consiste na contagem do número de chamadas à API necessárias, constatámos que o protótipo desenvolvido com a API de maior dimensão, a da OpenLayers, envolveu mais chamadas à mesma, e que, o protótipo desenvolvido com a API de menor dimensão, a da Google, envolveu menos chamadas à mesma. A API da OpenLayers, por ser de maior dimensão, obriga a um maior esforço no processo de escolha dos objetos adequados para cada tarefa. A maior diversidade de opções resulta, também, numa maior dificuldade na escolha dos objetos mais adequados, particularmente nos casos em que existem objetos com funcionalidades bastante aproximadas. Pelo contrário, numa interface mais reduzida, como a da Google, os objetos disponibilizados oferecem, frequentemente, um nível de abstração ligeiramente superior, o que facilita a aprendizagem e utilização da API. Um sintoma dessa maior facilidade de utilização, resultante de um mais adequado nível de abstração, é o menor número de chamadas de que nos tivemos de socorrer, ao implementar o protótipo com suportado pela API da Google, por comparação com as que tivemos de fazer no protótipo construído com o suporte da API da OpenLayers.

Um aspecto importante na adoção de uma API é o respetivo processo de aprendizagem. Neste caso, com os recursos disponíveis na realização desta dissertação, apenas foi possível realizar um estudo sobre a aprendizagem de uma das APIs. Por conveniência do programa da unidade curricular de Tecnologias de Informação Geográfica, do Mestrado em Engenharia Informática da FCT/UNL, a API selecionada para este estudo foi a do Google Maps. Em particular, estávamos interessados em identificar quais os objetos de utilização mais frequente, numa aplicação baseada nos mesmos requisitos, que deveriam constituir uma prioridade na aprendizagem da API, por novos programadores. Os objetos mais importantes, para o tipo de funcionalidades pretendidas neste estudo, são o `google.maps.latlng`, o `google.maps.latlngbounds`, o `google.maps.map` e o `google.maps.marker`. Depois de aprendidos os objetos mais prioritários, a ordem de frequência com que os restantes objetos são utilizados, sugere a ordem em que estes devem ser aprendidos.

Por fim, além da comparação qualitativa das APIs, baseada no processo de desenvolvimento dos protótipos, da comparação quantitativa, com base nos resultados obtidos através dos protótipos e na aplicação de métricas de avaliação de usabilidade, e da comparação de várias aplicações desenvolvidos com a API da Google, realizámos também uma comparação de evolução das APIs, com base em várias versões de cada um delas. Dessa comparação e aplicando as leis de evolução de software, é possível confirmar que as APIs estão a evoluir e que a facilidade de compreensão entre versões é estável. No entanto, quanto a questões de retro-compatibilidade, na API da Google, por não se verificar remoção de objetos entre as versões analisadas, é fácil a migração de uma versão da API para a versão seguinte. Nas APIs da OpenLayers e da Esri, por se verificarem remoções de objetos entre versões, há uma quebra de compatibilidade de aplicações de uma versão anterior que usassem esses objetos para as versões seguintes. Este fator pode portanto afetar a usabilidade destas duas APIs.

7.2 Trabalho Futuro

Após a avaliação de usabilidade baseada em dados recolhidos através de um programa desenvolvido para o efeito, o passo seguinte seria replicar o estudo feito com o conjunto de projetos quer com a API da Google, para verificar se os resultados em diferentes edições da cadeira seriam consistentes, quer nas restantes APIs, para aprendermos algo sobre quais os objetos prioritários para o tipo de projetos que se desenvolvem no contexto de uma cadeira de introdução ao desenvolvimento deste tipo de aplicações.

Outro estudo interessante seria, através da mineração de repositórios de aplicações com mapas expandir o trabalho realizado com os projetos de alunos para projetos reais, fazendo assim um retrato mais fidedigno da utilização efetiva, num contexto profissional, destas APIs.

Outros estudos interessantes de realizar combinariam dados de processo, tais como o esforço (medido como o tempo que demora a realizar uma tarefa, por pessoa) e a taxa de

defeitos introduzidos, ao utilizar as diferentes APIs.

Finalmente, para trabalho futuro fica também a possibilidade de realizar um estudo de avaliação de usabilidade entre APIs de mapas para dispositivos móveis. Estas APIs de mapas direcionadas a dispositivos móveis são ainda mais recentes que as APIs de mapas já existentes, direcionadas para o *browser*. Uma vez, que a tecnologia relativa a dispositivos móveis tem vindo a desenvolver-se rapidamente e o acesso a estes dispositivos tem vindo a generalizar-se mais, é importante desenvolver estudos de usabilidade para esta subcategoria de APIs de mapas.

Bibliografia

[BA04] Marcus A. S. Boxall e Saeed Araban. Interface metrics for reusability analysis of components. In *Proceedings of the 2004 Australian Software Engineering Conference*, ASWEC '04, pág. 40–51, Washington, DC, USA, 2004. IEEE Computer Society.

[BCR94] V. Basili, G. Caldiera, e R. Rombach. Goal question metric paradigm. *Encyclopedia of Software Engineering*, (pp. 469-476), 1994.

[Blo06] Joshua Bloch. How to design a good API and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pág. 506–507, New York, NY, USA, 2006. ACM.

[Cho08] T Edwin Chow. The potential of maps APIs for internet GIS applications. *Transactions in GIS*, 12(2):179–191, Abril 2008.

[Cla04] Steven Clarke. Measuring API usability. *Dr. Dobb's Journal*, 29:S6–S9, 2004.

[Cla06] Steven Clarke. Describing and measuring API usability with the Cognitive Dimensions. In *Cognitive Dimensions of Notations 10th Anniversary Workshop*, 2006.

[Con99] W. J. Conover. *Practical nonparametric statistics*. Wiley, 1999.

[DFMS09] John M. Daughtry, Umer Farooq, Brad A. Myers, e Jeffrey Stylos. API usability: report on special interest group at CHI. *SIGSOFT Softw. Eng. Notes*, 34(4):27–29, Julho 2009.

[DFSM09] John M. Daughtry, Umer Farooq, Jeffrey Stylos, e Brad A. Myers. API usability: CHI'2009 special interest group meeting. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, CHI EA '09, pág. 2771–2774, New York, NY, USA, 2009. ACM.

[DH09] Uri Dekel e James D. Herbsleb. Improving API documentation usability with knowledge pushing. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pág. 320–330, Washington, DC, USA, 2009. IEEE Computer Society.

[dM01] João Luís de Matos. *Fundamentos de informação geográfica*. Lidel, 4 edition, 2001.

[dSRC+04] Cleidson R. B. de Souza, David Redmiles, Li-Te Cheng, David Millen, e John Patterson. Sometimes you need to see through walls: a field study of application programming interfaces. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, pág. 63–71, New York, NY, USA, 2004. ACM.

[ESM07] Brian Ellis, Jeffrey Stylos, e Brad Myers. The factory pattern in API design: A usability evaluation. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pág. 302–312, Washington, DC, USA, 2007. IEEE Computer Society.

[FK05] W. B Frakes e Kyo Kang. Software reuse research: status and future. *IEEE Transactions on Software Engineering*, 31(7):529– 536, Julho 2005.

[FWZ10] Umer Farooq, Leon Welicki, e Dieter Zirkler. API usability peer reviews: a method for evaluating the usability of application programming interfaces. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pág. 2327–2336, New York, NY, USA, 2010. ACM.

[GA04] Miguel Goulão e O Brito E Abreu. Software components evaluation: an overview. *IN PROCEEDINGS OF THE 5ª CONFERÊNCIA DA APSI*, 2004.

[GHJV95] E. Gamma, R. Helm, R. Johnson, e J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1995.

[GP96] T. R. G Green e M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *JOURNAL OF VISUAL LANGUAGES AND COMPUTING*, 7:131–174, 1996.

[Hen09] Michi Henning. API design matters. *Commun. ACM*, 52(5):46–56, Maio 2009.

[iee98] IEEE standard for a software quality metrics methodology. available: http://ieeexplore.ieee.org /xpls/abs_all.jsp?arnumber=749159, 1998.

[IKMF09] Katsuro Inoue, Shinji Kusumoto, Makoto Matsushita, e Hikaru Fujiwara. *Patent US7627855 - Software component importance evaluation system*. 2009.

[iso91] ISO9126, 1991.

[IYY+05] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, e Shinji Kusumoto. Ranking significance of software components based on use relations. *IEEE Trans. Softw. Eng.*, 31(3):213–225, Março 2005.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, e A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Relatório técnico, Carnegie-Mellon University Software Engineering Institute, November 1990.

[Kru92] Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, Junho 1992.

[LB03] Allen Lee e Richard Baskerville. Generalizing generalizability in information systems research. *Information Systems Research*, 14(3):221–243, Setembro 2003.

[Leh78] Meir M. Lehman. Programs, cities, students, limits to growth? *Programming Methodology*, pág. 42–62, 1978. Inaugural Lecture.

[Leh79] M.M. Lehman. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1(0):213–221, 1979.

[MC11] Cleidson R.B. de Souza Marcelo Cataldo. The impact of API complexity on failures: An empirical analysis of proprietary and open source software systems. Technical Report 106, Instituti for Software Research, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Junho 2011.

[Mil56] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, March 1956.

[MRTS98] S. G McLellan, A. W Roesler, J. T Tempest, e C. I Spinuzzi. Building more usable APIs. *IEEE Software*, 15(3):78–86, Junho 1998.

[RD05] O. P. Rotaru e M. Dobre. Reusability metrics for software components. In *Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, AICCSA '05, pág. 85–92, Washington, DC, USA, 2005. IEEE Computer Society.

[Rob09] Martin P. Robillard. What makes APIs hard to learn? answers from developers. *IEEE Softw.*, 26(6):27–34, Novembro 2009.

[San05] R. Sanchez. 'Tacit knowledge'versus 'explicit knowledge'approaches to knowledge management practice. *TEAM LinG*, pág. 191, 2005.

[SC07] Jeffrey Stylos e Steven Clarke. Usability implications of requiring parameters in objects' constructors. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pág. 529–539, Washington, DC, USA, 2007. IEEE Computer Society.

[SCC02] William R Shadish, Thomas D Cook, e Donald T Campbell. *Experimental and quasi-experimental designs for generalized causal inference*, volume 100. Houghton Mifflin, 2002.

[SCM06] Jeffrey Stylos, Steven Clarke, e Brad Myers. Comparing API design choices with usability studies: A case study and future directions", PPIG. 2006.

[SKR08] Santonu Sarkar, Avinash C. Kak, e Girish Maskeri Rama. Metrics for measuring the quality of modularization of Large-Scale Object-Oriented software. *IEEE Trans. Softw. Eng.*, 34(5):700–720, Setembro 2008.

[SM07] Jeffrey Stylos e Brad Myers. Mapping the space of API design decisions. *Human-Computer Interaction Institute*, Janeiro 2007.

[Wor95] Michael F. Worboys. *GIS: A Computing Perspective*. Taylor&Francis, 1995.

[WYF03] Hironori Washizaki, Hirokazu Yamamoto, e Yoshiaki Fukazawa. A metrics suite for measuring reusability of software components. In *Proceedings of the 9th International Symposium on Software Metrics*, pág. 211–223, Washington, DC, USA, 2003. IEEE Computer Society.

[XAC09] Richard Fung Túlio Souza Xiang 'Anthony' Chen, Matthew Dunlap. How do developers choose APIs? Technical report, University of Calgary, 2009.

[ZER11] Minhaz F Zibran, Farjana Z Eishita, e Chanchal K Roy. Useful, but usable? factors affecting the usability of APIs. In *2011 18th Working Conference on Reverse Engineering (WCRE)*, pág. 151–155. IEEE, Outubro 2011.

[Zib08] Minhaz Zibran. What makes APIs difficult to use? *International Journal of Computer Science and Network Security*, 8(4):255–261, 2008.