U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Desenvolvimento de um *web service* para apoio ao cálculo de estruturas metálicas

RUI MANUEL SANTOS BARROS

Dissertação submetida para satisfação parcial dos requisitos do grau de

MESTRE EM ENGENHARIA CIVIL — ESPECIALIZAÇÃO EM ESTRUTURAS

Orientador: Professor Doutor José Miguel de Freitas Castro

Co-Orientador: Professor Doutor João Filipe Meneses Espinheira Rio

JULHO DE 2013

MESTRADO INTEGRADO EM ENGENHARIA CIVIL 2012/2013
DEPARTAMENTO DE ENGENHARIA CIVIL

Tel. +351-22-508 1901

Fax +351-22-508 1446

✉ miec@fe.up.pt

Editado por

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Rua Dr. Roberto Frias

4200-465 PORTO

Portugal

Tel. +351-22-508 1400

Fax +351-22-508 1440

✉ feup@fe.up.pt

🌍 http://www.fe.up.pt

Reproduções parciais deste documento serão autorizadas na condição que seja mencionado o Autor e feita referência a *Mestrado Integrado em Engenharia Civil - 2012/2013 - Departamento de Engenharia Civil, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal, 2013*.

As opiniões e informações incluídas neste documento representam unicamente o ponto de vista do respectivo Autor, não podendo o Editor aceitar qualquer responsabilidade legal ou outra em relação a erros ou omissões que possam existir.

Este documento foi produzido a partir de versão eletrónica fornecida pelo respectivo Autor.

Aos engenheiros do meu percurso:

a minha família e amigos.

"*The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom."*

Isaac Asimov

AGRADECIMENTOS

Sem o apoio incondicional das seguintes pessoas, às quais uma simples palavra de agradecimento peca por ser escassa, a realização deste trabalho jamais havia sido conseguida:

Ao meu orientador, o professor José Miguel Castro, pelo entusiasmo e atenção demonstrados em todas as horas que trabalhamos.

Ao meu coorientador, o professor João Filipe Rio, pelo constante perfecionismo demonstrado, onde nenhum pormenor foi deixado de parte nas suas análises.

Ao João Granado por prontamente se ter disponibilizado a colocar em prática as novas funcionalidade do *Flange+Web*.

Aos meus amigos, em especial aos que me acompanharam nas horas boas e nas menos boas, deste nosso já longo caminho que percorremos lado a lado.

À Ariana.

Aos meus pais e à minha irmã por tudo.

RESUMO

O âmbito deste trabalho prende-se com a necessidade de adaptar o mundo da engenharia civil aos novos paradigmas das tecnologias de informação. Tentando colmatar as carências identificadas, que passam pela criação ferramentas web para o cálculo estrutural, sem recorrência a programas comerciais, foi desenvolvido uma API *(Application Program Interface)* alocada nos servidores da FEUP e disponível através do portal *OpenG*. A API concretizou-se com o recurso à linguagem de programação *Python*, com o apoio da *framework Flask* e de bases de dados *SQL*.

O objetivo da criação da API surge da ambição de criar uma plataforma com um potencial de expansibilidade assinalável, capaz de cobrir no futuro uma grande parte dos campos da engenharia estrutural, e de serem criadas aplicações *web* com base neste trabalho. Iniciou-se a criação da API no contexto do cálculo de secções de perfis metálicos, onde através de pedidos HTTP, o utilizador tem acesso a um vasto leque de informação relativa às secções. Foram incluídos dois tipos de secções metálicas na API: perfis comerciais laminados a quente das gamas europeias IPE e HE, e também da gama britânica UC e UB, assim como perfis compostos por placas soldadas também denominados por *built up*.

Todos os cálculos e algoritmos presentes no trabalho têm por base o Eurocódigo 3, que regula a construção metálica em Portugal e em grande parte da Europa. Além disso foram propostas alternativas de cálculo a esta norma com base em gráficos de curvas de interação de esforço axial com momento fletor, que foram impulsionadas pelas potencialidades do *Python*.

Palavras-chave: API, Engenharia estrutural, Perfis metálicos, *Python*, Tecnologias de Informação

ABSTRACT

The scope of this work concerns the need to adapt the civil engineering world to the new paradigms of the information technologies. Intended to close the existing shortage of web tools for structural calculus, without commercial software, an API (Application Program Interface) was developed. Allocated in FEUP's servers and available by the OpenG portal, the API was developed with the Python programming language, with the support of Flask framework and SQL databases.

The main goals of this API emerged from the ambition to create a platform with a remarkable expandability potential, able to embrace the majority of structural engineering fields, and a future creation of web applications with this work as a basis. The API began in the context of steel cross sections calculus that can be accessed by HTTP requests, containing a wide range of information related to the cross sections. Two kinds of steel cross sections were included in the API: hot rolled commercial steel profiles from the European ranges IPE and HE, and also from the British ranges UC and UB, as well as profiles composed by welded plates also known as built up profiles.

The Eurocode 3, which is the standard that regulates the steel construction in Portugal and in a big part of Europe, was the basis of all the calculus and algorithms presented in this work. Furthermore, alternatives of calculus to this standard were proposed, which are some interaction curves of axial strength combined with bending moment, boosted by the Python potential.

Keywords: API, Information technologies, Python, Steel profiles, Structural Engineering

ÍNDICE GERAL

AGRADECIMENTOS I

RESUMO III

ABSTRACT V

1 INTRODUÇÃO 1

1.1. CONSIDERAÇÕES GERAIS 1

1.2. MOTIVAÇÃO 1

1.3. ÂMBITO E OBJETIVO 1

1.4. ESTRUTURA DA DISSERTAÇÃO 2

2 AS TECNOLOGIAS DE INFORMAÇÃO E A ENGENHARIA CIVIL 3

2.1. TECNOLOGIAS DE INFORMAÇÃO WEB 3

2.1.1. O *PERSONAL COMPUTER* 3

2.1.2. A INTERNET 4

2.1.3. ATUALIDADE – APLICAÇÕES WEB 5

2.2. TECNOLOGIAS DE INFORMAÇÃO E ENGENHARIA DE ESTRUTURAS 8

2.2.1. INTRODUÇÃO 8

2.2.2. CAD, BIM E IFC 9

2.2.3. TECNOLOGIAS *WEB* NA ENGENHARIA ESTRUTURAL 10

2.3. CONSIDERAÇÕES FINAIS 12

3 TECNOLOGIAS ABORDADAS 13

3.1. INTRODUÇÃO 13

3.2. PROGRAMAÇÃO ORIENTADA A OBJETOS 13

3.3. LINGUAGENS DE PROGRAMAÇÃO 14

3.4. PHP *VS. PYTHON* 15

3.4.1. UTILIZAÇÕES DO PHP 15

3.4.2. *PYTHON* 16

3.4.3. COMPARAÇÃO ENTRE AS DUAS LINGUAGENS 16

3.4.4. COMPONENTES DO CÓDIGO *PYTHON* 16
3.4.5. *SCIPY*, *NUMPY* E *MATPLOTLIB* 21
3.5. BASE DE DADOS 22
3.5.1. *SQL – STRUCTURED QUERY LANGUAGE* 22
3.5.2. BASE DE DADOS DO *FLANGE+WEB* 23
3.6. APIS E *WEB SERVICES* EM APLICAÇÕES *WEB* 23
3.6.1. CONCEITOS 23
3.6.2. REST - *REPRESENTATIONAL STATE TRANSFER* 26
3.6.3. EXEMPLOS NA ENGENHARIA ESTRUTURAL 27
3.7. *FLASK MICROFRAMEWORK* 27
3.7.1. *HTTP - HYPERTEXT TRANSFER PROTOCOL* 27
3.7.2. DE API A *WEB SERVICE* 28
3.7.3. *MODEL-VIEW-CONTROLLER* 28
3.8. *WEB SERVICE* 29
3.9. CONSIDERAÇÕES FINAIS 30

4 CÁLCULO DE SECÇÕES METÁLICAS SEGUNDO O EUROCÓDIGO 3 31

4.1. INTRODUÇÃO 31
4.2. ANÁLISE E DIMENSIONAMENTO DE ESTRUTURAS METÁLICAS 32
4.2.1. O MATERIAL AÇO 32
4.2.2. MÉTODOS DE ANÁLISE ESTRUTURAL 33
4.2.3. BASES PARA A CLASSIFICAÇÃO DE SECÇÕES METÁLICAS 33
4.3. VERIFICAÇÕES DE SEGURANÇA DO EC3 37
4.3.1. ESFORÇO AXIAL DE TRAÇÃO 38
4.3.2. ESFORÇO AXIAL DE COMPRESSÃO 39
4.3.3. MOMENTO FLETOR 39
4.3.4. ESFORÇO TRANSVERSO 40
4.3.5. INTERAÇÃO DE ESFORÇOS 41
4.4. CURVAS DE INTERAÇÃO 43
4.4.1. CURVA DE INTERAÇÃO ELÁSTICA 44
4.4.2. CURVA DE INTERAÇÃO PLÁSTICA 45
4.4.3. CURVA COM BASE NOS CRITÉRIOS DE CLASSIFICAÇÃO DO EC3 47

4.4.4. CONFRONTAÇÃO COM AS CURVAS OBTIDAS DAS FÓRMULAS DO EC3 49
4.5. CONSIDERAÇÕES FINAIS 50

5 IMPLEMENTAÇÃO DO WEB SERVICE 51
5.1. INTRODUÇÃO 51
5.2. ABORDAGEM LÓGICA 51
5.2.1. *BUSINESS LOGIC* 51
5.2.2. MVC 53
5.2.3. ESTRUTURA DO URL 55
5.2.4. ARQUITETURA FINAL DO MODELO DE EXPANSÃO 56
5.3. DOCUMENTAÇÃO 57
5.4. EXEMPLOS DE APLICAÇÃO 58
5.5. CONSIDERAÇÕES FINAIS 61

6 CONCLUSÕES E FUTUROS DESENVOLVIMENTOS 63

BIBLIOGRAFIA 65

ANEXOS 67
ANEXO A 69
ANEXO B 73

ÍNDICE DE FIGURAS

Fig. 2.1 - Primeiro PC - Apple II 4
Fig. 2.2 - Primeiro Website 6
Fig. 2.3 - Aplicação web Flange+Web 8
Fig. 2.4 - Aplicação para o cálculo do MEF [12] 11
Fig. 2.5 - Aplicação *ifcserver* carregada em *browser* 11
Fig. 2.6 - Aplicação *ifcserver* carregada em *smartphone* 12
Fig. 3.1 - Articulação entre *packages* e módulos em *Python* 17
Fig. 3.2 - *Python*: Variáveis [19] 18
Fig. 3.3 - *Python: Strings* [19] 18
Fig. 3.4 - *Python: Tupples* [19] 18
Fig. 3.5 - *Python: Listas* [19] 19
Fig. 3.6 - *Python*: Dicionários [19] 19
Fig. 3.7 - *Python*: Funções [19] 19
Fig. 3.8 - *Python: If Statement* [19] 20
Fig. 3.9 - *Python*: Ciclo *For* [19] 20
Fig. 3.10 - *Python*: Classes [19] 20
Fig. 3.11 - Python: Geração de objetos [19] 20
Fig. 3.12 - *Python*: Construtor [19] 21
Fig. 3.13 - *Python*: Exemplo mais complexo de geração de objetos [19] 21
Fig. 3.14 - Python: Instâncias das classes [19] 21
Fig. 3.15 - Python: Métodos das classes [19] 21
Fig. 3.16 - Exemplo de um array Numpy usado no web service 22
Fig. 3.17 - Interação entre o cliente e as APIs 24
Fig. 3.18 - Consola *Google* para a gestão do uso de APIs da *Google* numa aplicação 25
Fig. 3.19 - Processo de desenvolvimento de APIs [23] 26
Fig. 3.20 - Estruturas de dados. À esquerda JSON, e à direita XML 27
Fig. 3.21 - Exemplo de endereço URL, elucidativo de um pedido GET 28
Fig. 4.1 – Aproveitamento da hiperestaticidade das estruturas através de uma análise plástica [32] (adaptado) 34
Fig. 4.2 - Relação tensão-extensão bilinear do aço 35
Fig. 4.3 – Gráfico demonstrativo da diferença na capacidade resistente e rotacional, de secções de diferentes classes [32] (adaptado) 36
Fig. 4.4 - Secção Tipo – IPE 37

Fig. 4.5 - Curva de interação elástica para o perfil IPE 750 x 137, para flexão em torno do eixo dos *yy*, obtida através do *Matplotlib* 45
Fig. 4.6 - Representação de um perfil IPE sujeito a um esforço axial, e o seu momento resistente 46
Fig. 4.7 - Curva de interação plástica para o perfil IPE 750 x 137, para flexão em torno do eixo dos *yy*, obtida através do *Matplotlib* 47
Fig. 4.8 - Representação da curva de interação, de acordo com as normas de classificação do EC3 48
Fig. 4.9 - Curvas de interação de alguns perfis da gama IPE 49
Fig. 4.10 - Detalhe da curva de interação referente à zona onde o EC3 não é conservativo na sua simplificação 50
Fig. 5.1 - Esquematização interna do *model* do modelo MVC 53
Fig. 5.2 - Exemplo de uma aplicação minimalista através da *framework Flask* [28] 54
Fig. 5.3 - Uso de variáveis nos URLs. [28] 54
Fig. 5.4 - Articulação entre os vários componentes do modelo MVC 55
Fig. 5.5 - Estruturação do URL 56
Fig. 5.6 - Articulação entre os vários agentes de processo de desenvolvimento 57
Fig. 5.7 - Exemplo de um pedido de um perfil *built up* 58
Fig. 5.8 - Exemplo de um pedido de um perfil IPE300 59
Fig. 5.9 - Exemplo de um pedido da área de um perfil IPE300 59
Fig. 5.10 - Exemplo de aplicação do *web service* em folha de cálculo Excel 60
Fig. 5.11 - Código *Visual Basic* que possibilita o pedido HTTP 61

ÍNDICE DE TABELAS

Tabela 1 - História das APIs [22] 24
Tabela 2 - Resistências para cada classe de aço estrutural segundo EN10025-2 32

SÍMBOLOS E ABREVIATURAS

API – Application Programming Interface
BIM – Building Information Modeling
CAD – Computer Aided Design
CEN - Comité Europeu de Normalização
CSS – Cascading Stryle Sheets
EC3 – Eurocódigo 3
FEUP – Faculdade de Engenharia da Universidade do Porto
HTML – Hypertext Markup Language
IFC – Industry Foundation Classes
JSON – Javascript Object Notation
MEF – Método dos Elementos Finitos
MVC – Model-View-Controller
OOP – Object Oriented Programming
REST - Representational State Transfer
SQL – Structured Query Language
TI – Tecnologias da Informação
URL – Uniform Resource Locator
WS – Web Service
XML – eXtensible Markup Language

A – Área bruta da secção
A_{eff} – Área efetiva da secção
A_v - Área de corte
b – Largura do banzo
d – Comprimento da alma
E – Módulo de Elasticidade
f_u – Tensão de cedência do aço
f_y – Tensão de cedência do aço
h_i – Altura interna da secção
r – Raio da zona de ligação da alma com os banzos
R_d – Valor de cálculo
R_k – Valor característico

t_f – Espessura do banzo

t_w – Espessura da alma

W – Módulo de flexão da secção

W_{eff} – Módulo de flexão efetivo da secção

γ_{Mi} – Coeficiente parcial de segurança

ε_u - Extensão correspondente à tensão de rotura

ε_y - Extensão correspondente à tensão de cedência

$\sigma_{x,Ed}$ – Valor de cálculo da tensão longitudinal atuante no eixo dos *xx* , no ponto considerado

$\sigma_{z,Ed}$ – Valor de cálculo da tensão transversal atuante no eixo dos *zz,* no ponto considerado

τ_{Ed} - Valor de cálculo da tensão tangencial atuante no ponto considerado

1 INTRODUÇÃO

1.1. CONSIDERAÇÕES GERAIS

O tema desta dissertação surge da preocupação em criar uma ligação, na opinião do autor quase inexistente, entre o mundo da engenharia civil, mais concretamente o cálculo estrutural, e os novos paradigmas de comunicação existentes na sociedade atual. Tendo a internet vindo revolucionar a forma como interagimos com o mundo, e sendo essa a nossa principal fonte de informação, torna-se pertinente e quase obrigatório fundir um pouco da engenharia estrutural com este poço de informação sem fim à vista, e colocar conhecimento relativo à nossa área de formação ao alcance de um clique.

1.2. MOTIVAÇÃO

Sendo o mundo das aplicações *web* uma verdade atual das tecnologias de informação, e estando o mundo da engenharia civil restrito a um grupo de pessoas que na maioria dos casos não dominam tais tecnologias, torna-se obrigatório a criação de sinergias para unir ambas as partes. No decorrer deste trabalho foi criada uma API *(Application Programming Interface)* com o objetivo de estimular a produção de aplicações *web*, para uso no seio da engenharia civil, nomeadamente pelos engenheiros de estruturas.

O trabalho decorre na continuação de uma aplicação *web* designada por *Flange+Web*, e desenvolvida por Garcia [1]. Esta aplicação que fornece informação relativa a perfis metálicos foi um excelente ponto de partida, porém era evidente que a aplicação necessitava de uma base mais sólida, com a finalidade de que o projeto fosse continuado e expandido com sucesso.

1.3. ÂMBITO E OBJETIVO

Tal como o nome indica, um *web service* tem como função a disponibilização de serviços através da internet. Existem já vários exemplos de indústrias que recorrem este tipo de serviços para satisfazer a sua produtividade, de tal modo que se tornaram numa ferramenta chave de trabalho. No caso da engenharia civil, o uso de programas de cálculo automático comerciais é ainda na maioria dos casos o principal método para a obtenção de soluções. Estando o mundo do conhecimento a migrar para um tipo de conhecimento remoto, onde a informação está disponível de um modo permanente e aberto, coloca-se a questão: fará sentido continuar a indústria da engenharia civil alheia a estas mudanças? Com a pluralidade de saber existente em todo o mundo, fará sentido continuar-se num paradigma onde o código é imutável sem que ninguém acrescente nada ao conhecimento que se idealiza como

absoluto? Continuará a fazer sentido manter-se a informação estanque e imutável, em vez de atualizada ao segundo?

O objetivo principal desta dissertação consiste na criação de uma plataforma onde cresçam funcionalidades uteis para os profissionais da área da engenharia de estruturas, que se encontrem disponíveis *online* juntamente com a respetiva documentação, e em formato *open source* para que este projeto possa no futuro crescer cada vez mais.

1.4. ESTRUTURA DA DISSERTAÇÃO

No segundo capítulo deste documento far-se-á uma descrição do estado da arte no domínio dos *web services*, e dar-se-á alusão a alguns exemplos de aplicações *web* criadas até agora. No terceiro capítulo serão abordadas com maior profundidade as tecnologias utilizadas no desenvolvimento *web*, discutir-se-ão as linguagens de programação, as *framewoks* e a sua aplicabilidade. No quarto capítulo serão descritos os algoritmos para o cálculo estrutural abordado, a sua contextualização no âmbito das normas em vigor, nomeadamente o Eurocódigo 3, e será dada uma explicação de como foram obtidas curvas de interação de esforço axial/momento fletor, de uma determinada secção metálica. No quinto capítulo será abordada a utilização do *web service* no contexto da aplicação *Flange+Web*, assim como em futuras aplicações. No sexto e último capítulo serão expostas as conclusões desta dissertação, e serão propostos alguns desenvolvimentos para o futuro.

2
AS TECNOLOGIAS DE INFORMAÇÃO E A ENGENHARIA CIVIL

2.1. TECNOLOGIAS DE INFORMAÇÃO WEB

A inovação nas tecnologias de informação (TI) alterou radicalmente o modo como interagimos com o mundo. Hoje em dia a forma como a informação chega às pessoas sofreu uma mutação, que permitiu uma melhoria na qualidade e uma poupança no tempo em que essa informação chega aos potenciais interessados. Hoje já não temos de esperar pelas notícias pois estas são instantâneas e chegam a todos os nossos dispositivos.

2.1.1. O *PERSONAL COMPUTER*

Façamos em primeiro lugar uma contextualização histórica destas tecnologias para entendermos realmente o que motivou o Homem a tomar este caminho evolutivo. Eischen define a génese de TI através do conceito de algoritmo [2]. Este autor refere Leibniz, que acreditava que tudo no mundo pode ser modelado matematicamente através de algoritmos que produzem *outputs* percetíveis aos humanos. Porém, este processo de otimização dos algoritmos levou seculos até chegarmos ao paradigma de hoje, começando no desejo por um "computador máquina" utilizado nas indústrias, passando pela codificação binária que ainda hoje é utilizada, até à internet como hoje a conhecemos.

Durante a revolução industrial, entre o final do século XIX e o início do século XX, esta ideia de algoritmo para atingir determinado fim esteve bem presente no ímpeto das figuras da época. Nessa altura começaram as surgir as primeiras máquinas industriais e, com elas, as primeiras tentativas em colocar nas ditas máquinas inteligência suficiente para que completassem processos sem a interferência humana. Foi também nesta época que começaram a surgir as primeiras tentativas da criação de máquinas multitarefas que automatizassem procedimentos e que gerissem a informação. Porém, só em 1944, em plena II Guerra Mundial, foi conseguida a primeira máquina totalmente programável denominada computador, criada para a marinha Norte-Americana. Esta tecnologia, implementada pela IBM, consistia numa máquina de cinco toneladas que lia cartões perfurados. Os Estados Unidos tiraram então partido das unidades de investigação das suas universidades, muito avançadas para a altura, para produzirem ferramentas que pudessem ser vantajosas durante a guerra. Ainda no ano de 1944 foi lançado também o primeiro computador eletrónico, com uma capacidade de cerca de 5000 operações por minuto. Nas décadas seguintes a indústria tentou desenvolver soluções com base nas máquinas anteriormente referidas, obtendo resultados não numa perspetiva de produção em série, mas com o enfoque num cliente em especifico, muito devido ao facto da impossibilidade de

interligação entre sistemas tanto a nível de *hardware*, como de *software*. Contudo já em 1959 havia sido descoberto o circuito integrado (CI), que mais tarde viria a revolucionar toda a história da computação. Esta descoberta permitiu que informação que antigamente teria de ser guardada num espaço físico enorme, passa-se a poder ser alocada em pequenos *chips*. Este avanço, combinado com o aparecimento em 1971 dos microprocessadores por parte da Intel, que permitia a um *chip* apenas desempenhar várias funções, foram o ponto de partida para o surgimento dos primeiros *personal computers* (PCs). O primeiro aparelho desta gama foi o Apple II em 1977 (Fig. 2.1).

Fig. 2.1 - Primeiro PC - Apple II

Ainda assim, um dos aspetos que muito influenciou o desenvolvimento das TI, foi o aparecimento em 1964 de sistemas que permitiam a troca de *software* entre todos os computadores da mesma gama, o que hoje conhecemos como Windows ou Linux. Esta inovação por parte da IBM permitiu separar a aquisição de *software* da aquisição de hardware por parte dos consumidores, e de igual modo começaram a aparecer empresas mais ligadas somente a uma das partes, o que não acontecia até à época. Já no início da década de 90, com a expansão dos PCs em geral e da internet em particular, proliferaram as grandes empresas de distribuição de *software*. Na referida década a expansão da gama de *software* disponível nos mercados, aliada a uma cada vez maior robustez do *hardware*, representaram uma transformação de larga escala no mundo da informática.

2.1.2. A INTERNET

Nos finais dos anos 60, nos Estados Unidos, algumas universidades já estudavam a possibilidade do transporte da informação a longa distancia, no que viria a tornar-se mais tarde na internet. Em 1969 foi conseguida a primeira ligação que conectava a UCLA, a UCSB, o SRI International e a Universidade de Utah. Vários problemas apareceram logo à partida, como por exemplo problemas de compatibilidade, já que as máquinas apresentavam diferenças entre si, tanto a nível de *hardware* como de *software*. A solução foi criar um filtro intermédio, o que hoje conhecemos como servidores, na altura denominados de IMPs *(Interface Message Processors)*, que tinham como função fazer a gestão da informação que circulava na rede. Outra problemática existente era o facto de, tal como numa chamada telefónica, o conteúdo da informação era apenas exclusivo dos intervenientes do telefonema,

o que contrapõe à idealização inicial do conceito de internet, onde a informação estaria disponível de um modo permanente. A juntar a isto, havia ainda uma perda de dados nas ligações. Estas questões foram resolvidas com o recurso ao conceito de *packet-switching architecture*, conceito este que implicava a informação estar dividida em pequenas partes de modo a serem transmitidas a cada computador. Esta estrutura estabeleceu a *framework* onde as ligações começaram a ser criadas [2].

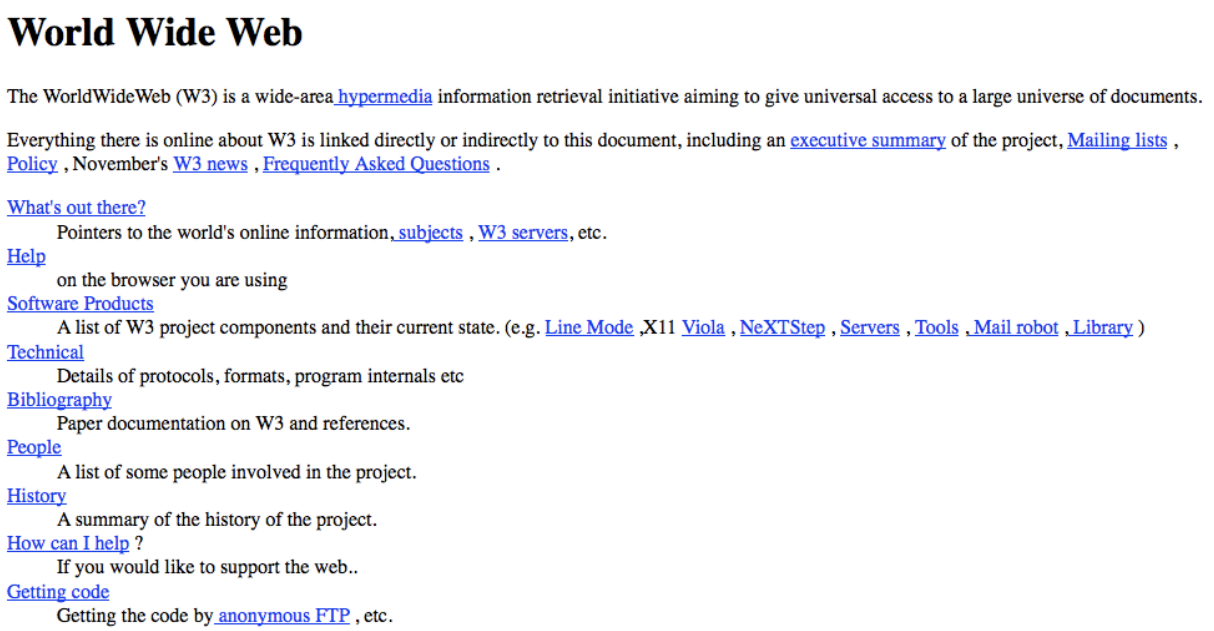
Dois dos desenvolvimentos mais significativos na altura foi o surgimento da *Ethernet* e do protocolo TCP/IC. A *Ethernet*, inventada por Bob Metcalfe, permitiu a criação de áreas de rede locais (*local area networks* - LAN), que possibilitou pela primeira vez a ligação dos computadores dentro das universidades. Já o *transmission control protocol* (TCP) foi desenhado por Vint Cerf e Bob Kahn em 1974. Este protocolo veio uniformizar a linguagem de comunicação entre as redes e, juntando a isto o conceito de *internet protocol* (IP) introduzido em 1978, foram criadas condições para a conexão de grande parte dos computadores a nível mundial.

Enquanto nos anos 80 a internet era essencialmente utilizada no meio académico, militar e governamental, na década seguinte com o aparecimento do protocolo HTTP *(HyperText Transfer Protocol)* e com o desenvolvimento dos primeiros *browsers*, o mundo assistiu a uma massificação desta tecnologia.

Contudo, existiam ainda alguns entraves à fácil distribuição da informação. Os endereços IP eram (e ainda o são) constituídos por números pouco intuitivos ao ser humano e apenas identificam um computador. Foi então criado em 1991 o *Uniform Resource Locator* (URL) por Tim Berners-Lee juntamente com um grupo do CERN. A informação contida num URL é de fácil compreensão, e existe a possibilidade de manipulação deste URL para a especificação de páginas mais ou menos específicas. Além disso, a equipa foi também responsável pelo desenvolvimento do *HyperText Markup Language* (HTML), uma linguagem que normaliza conteúdo das páginas da internet lidas por cada *browser*. Este conjunto de inovações por parte da referida equipa deu origem ao *World Wide Web*, sistema de informação que permite a conexão entre computadores através de uma linguagem própria de informação [2].

2.1.3. ATUALIDADE – APLICAÇÕES WEB

Tim Berners-Lee e a sua equipa do CERN criaram o primeiro *web site* do mundo (Fig. 2.2), que consistia numa coleção de documentos estáticos em formato HTML. Desde então o mundo da web assistiu a uma evolução vertiginosa no que à interatividade das páginas diz respeito. No início a maioria das páginas não autenticava os seus utilizadores porque não havia necessidade – cada utilizador era tratado da mesma forma e a informação era disponibilizada de igual modo para todos [3]. O aperfeiçoamento das várias tecnologias envolvidas possibilitou a migração das páginas estáticas que não passavam de repositórios de informação, como de resto é representado na Fig. 2.2, para aplicações *web* onde é possível a execução de comandos, como por exemplo a autenticação [4].



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Fig. 2.2 - Primeiro Website

Kappel et al. definem o conceito de aplicação web do seguinte modo:

"Uma aplicação web é um software baseado em tecnologias e normas do World Wide Web Consortium (W3C), e que disponibiliza recursos como conteúdos e serviços através da interface com o utilizador – o web browser". [5]

Hoje em dia assistimos à massificação de aplicações *web* em todas as áreas, entre as quais o *eCommerce*, plataformas bancárias, redes sociais, não só ao nível do *browser* como também ao nível de *smartphones* ou *tablets*. [4]. Estas aplicações requerem muitas vezes um registo e autenticação, e contemplam igualmente conteúdos privados do utilizador ou até mesmo operações financeiras. [3].

As características que distinguem uma aplicação *web* são, ainda segundo Casteleyn [4] as seguintes:

- Mais acessibilidade à informação e serviços: deve ser possibilitado o acesso ao maior número possível de utilizadores através da criação de diferentes *layouts*.
- Abordagem *document-centric*: o desenvolvimento de aplicações web ainda é considerado centrado numa página, apesar da diversidade de *links* e gráficos existentes nas aplicações;
- Gestão de dados: formatos de transferência de dados (XML – *Extensible Markup Language* e JSON – *JavaScript Object Natation*) e bases de dados são necessários para ser criado um fluxo de informação e para esta ser devidamente armazenada;
- Multiplicidade de formatos de apresentação: com o elevado número de *browsers* e dispositivos onde pode ser apresentada a informação existentes no mercado, torna-se imperativo criar compatibilidade entre todos estes formatos;
- Complexidade da arquitetura: com a elevada acessibilidade às tecnologias, deve ser considerado o uso das mesmas por uma gama de utilizadores elevada, incluindo até

aqueles pouco familiarizados com as TI, o que promove uma grande complexidade interna das aplicações;

São muitos os exemplos claros desta migração. Entre eles estão os serviços de *email*, os processadores de texto, e folhas de cálculo. Hoje em dia, *softwares* como o *Microsoft Outlook*, foram substituídos por aplicações *web* como o *Outlook Web Access*, enquanto de igual modo foram encontradas alternativas aos programas *"Office"* tradicionais como o *Google Docs* ou o *Google Spreadsheets*. Esta tecnologia permitiu a substituição em muitos casos do *software* tradicional, para o paradigma de aplicação *web*:

"O fator diferenciador das aplicações web quando comparadas com o software tradicional, é a maneira como a web é utilizada, isto é, as tecnologias são utilizadas tanto como plataforma de desenvolvimento, como plataforma de utilização ao mesmo tempo". [5]

Estes autores separam ainda mais os dois tipos de *software* na ótica dos programadores, ao referirem que devido às características especiais da web, o desenvolvimento de aplicações tradicionais em quase nada toca o desenvolvimento de aplicações *web*.

Stuttard e Pinto apontam as vantagens das aplicações *web* [3]*:*

- O protocolo HTTP possibilita uma fácil transferência de informação entre servidores e clientes;
- Sem dúvida a grande vantagem das aplicações *web* em comparação com o *software* tradicional é a acessibilidade. Neste caso todo o conteúdo da aplicação é mostrado ao utilizador através do *browser*, não existindo o requisito de mais nenhuma instalação;
- Hoje em dia a funcionalidade dos *browsers* permite uma interface com o utilizador bastante funcional. Esta funcionalidade é em muito devida à uniformização das linguagens *"client-side scripting"* como o HTML ou o *Javascript*;
- As tecnologias e linguagens utilizadas no desenvolvimento de aplicações são de aprendizagem relativamente fácil. Existe uma grande e emergente panóplia de plataformas *online* (MOOC – *Massive Online Open Courses*) como para o ensino destas tecnologias, por exemplo o *Udacity*. [6]. Existem ainda *frameworks* que facilitam o desenvolvimento e ainda código *open source* que também pode ser útil.

Na Fig. 2.3 é apresentado o *layout* da aplicação *web Flange+Web*, um protótipo desenvolvido pelo grupo de investigação no qual se desenvolve esta dissertação.

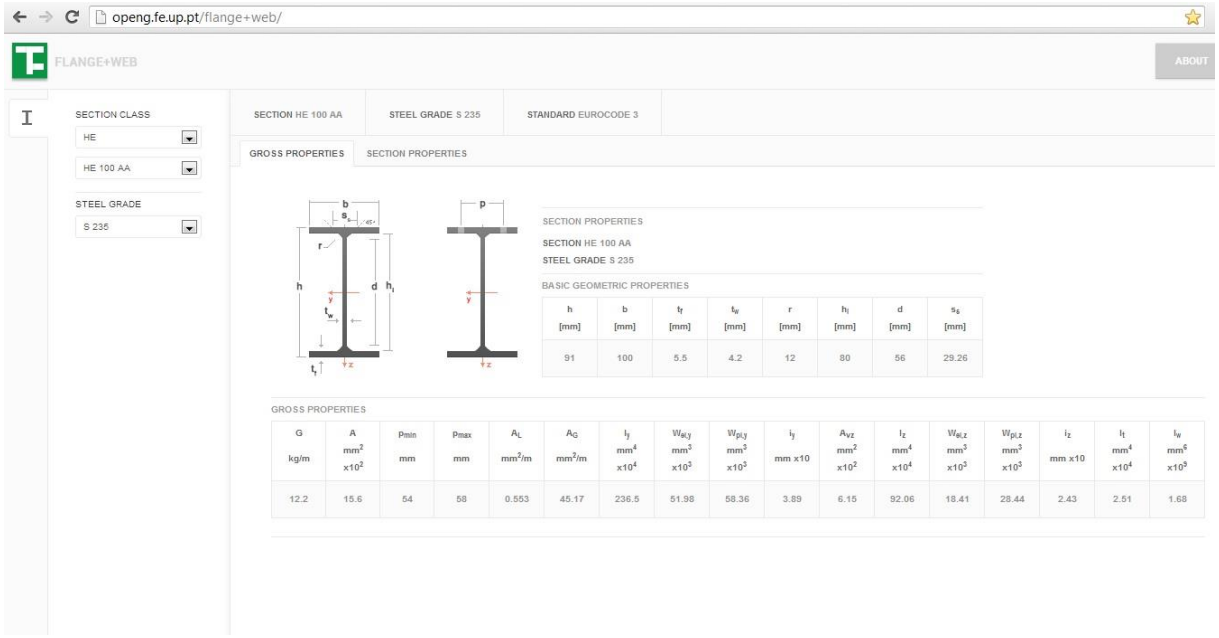


Fig. 2.3 - Aplicação web Flange+Web

2.2. TECNOLOGIAS DE INFORMAÇÃO E ENGENHARIA DE ESTRUTURAS

2.2.1. INTRODUÇÃO

Desde cedo que, devido à complexidade da área, a engenharia estrutural recorre a processos automáticos para a obtenção de resultados. Na realidade, a evolução da engenharia estrutural está umbilicalmente relacionada com a evolução da informática, pois análises mais robustas foram possíveis assim que começaram a surgir computadores com uma capacidade assinalável.

Tal como foi feito anteriormente para as TI, é possível abordar historicamente o tema da engenharia estrutural e fazer uma contextualização do paradigma atual. Seria impossível falar da engenharia estrutural sem ao mesmo tempo contemplar o ramo da construção, pois um trata da concretização do outro.

Civilizações já extintas como a Egípcia, a Grega, ou a Romana construíram grandes obras de engenharia que ainda hoje perduram. Porém, os dimensionamentos, os materiais e os métodos construtivos aplicados tiveram apenas por base o empirismo de vários séculos de cíclica experimentação. Foi no século XIX que surgiram os primeiros modelos de cálculo, entre eles o método das forças, o método dos deslocamentos, e os teoremas de energia. Gauss, Cauchy, Navier, Maxwell, Castigliano, entre outros, foram os responsáveis por esta evolução no pensamento dos edifícios e outras estruturas. Na primeira metade do século XX surgiu o conceito de diferenças finitas com Cross, Grinter e Southwell, e foi ainda no neste século que surgiu o método dos elementos finitos (MEF). Este método, devido à sua complexa abordagem do cálculo estrutural, potenciou o desenvolvimento de programas de cálculo automático [7].

Desde que o cálculo automático passou a ser uma constante no mundo da engenharia estrutural que os profissionais perceberam que, apesar de todas as vantagens que esta evolução representa, o seu espirito critico e intuição deveriam ser apurados recorrendo à sua experiencia, dado que os programas tinham pouca maturidade, e ainda careciam de um olhar mais conservador. Hoje, contudo, dada a robustez atual dos programas, os resultados obtidos por estes traduzem relativamente bem os fenómenos físicos observados. Porém, não deixa de ser imperativo criar uma clivagem entre as competências do Homem

e da máquina. Há que compreender que para existir automatismo o Homem terá de estar presente durante todo o processo, apesar das vantagens trazidas pelo avanço da computação. Muitas vezes este processo de interação passa pela utilização de ferramentas CAD *(Computer Aided Design)*, BIM *(Building Modeling Information)*, ou IFC *(Industry Foundation Classes)* [7].

2.2.2. CAD, BIM E IFC

Os conceitos de CAD, BIM e IFC foram consequências naturais da informatização da indústria da construção. Surgiram numa tentativa de automatizar processos, e de ser gerada informação com mais qualidade.

CAD *(Computer Aided Design)* pode ser definido como o uso de sistemas computorizados para assistir a criação, modificação, análise e otimização do projeto. Os *softwares* CAD existentes consistem na disponibilização gráfica por parte do computador da informação transmitida pelo utilizador. Alguns programas de cálculo estrutural automático também podem ser enquadrados no conceito de CAD, já que também comportam a visualização gráfica dos elementos, através do computador [8].

Já o conceito de BIM *(Building Information Modeling)* vai muito além dos conceitos de CAD e de programa de cálculo automático, sendo bastante mais abrangente. A definição desta metodologia está longe de obter um consenso. Contudo o *The American National Institute of Building Sciences* define BIM como: *"Uma representação computacional das características físicas e funcionais de um edifício, e da sua informação relativa ao projeto e ciclo de vida, usando normas da indústria, que permite aos agentes decisores criar maior valor"*. Este conceito tem vindo a ganhar atenção nos últimos anos. Com o avanço das capacidades gráficas dos computadores, houve um aumento de *software* disponível para rever esta metodologia, e prevê-se que num futuro próximo esta seja a plataforma utilizada por todos os profissionais da área da construção tanto engenheiros, como arquitetos, ou até mesmo donos de obra para gerirem a informação da construção entre si.

Paralelamente ao BIM surgiu o IFC *(Industry Foundation Classes)* que, à imagem do BIM, preocupou-se com a comunicação entre os intervenientes do sector da construção. Este conceito estabelece normas para a comunicação entre os vários intervenientes do ramo. É oportuna a referência ao IFC neste trabalho dado que este conceito está em tudo ligado à programação orientada a objetos, a qual foi um dos paradigmas chave na elaboração do *web service* desenvolvido no âmbito deste trabalho, e será abordado em maior detalhe no próximo capítulo.

A mitologia da torre de Babel que se acredita poder ter sido construída por volta do ano 5000 A.C. tinha como objetivo atingir os céus. Não chegou a ser concluída como refere a mitologia, devido às diferentes origens dos povos que trabalhavam na sua construção que, como falavam em línguas distintas, resultaram em graves falhas de comunicação que tornaram inviável a sua conclusão. Independentemente deste facto ser verdade ou ficção, podemos tomar esta alegoria como ponto de partida para a discussão da necessidade de uma linguagem comum no mundo da construção.

Desde então temos vindo a tentar comunicar e a uniformizar as linguagens, contudo, nem sempre da melhor forma. É frequente encontrarmos exemplos ainda hoje em dia de divergências desta ordem entre engenheiros, arquitetos. Como foi referido, as tecnologias BIM tentam colmatar estas falhas, numa tentativa de aproximar todas as partes. Com a proliferação das tecnologias BIM e CAD multiplicaram-se os *softwares* dedicados a cada uma das especialidades, é então que surge o conceito de IFC, que tenta não só articular os agentes que intervêm na construção, como também criar uma linguagem própria para a melhor compreensão entre todos [9].

O conceito encontra-se atualmente normalizado através da norma ISO/PAS 16739 [10]. Esta norma tem como objetivo formalizar as classes de objetos, todas as suas definições e as ligações entre estes. Na prática, o objetivo foi criar uma livraria robusta em que estão detalhados todos os elementos da construção como vigas, pilares ou paredes, e juntar todos estes componentes numa base de dados. Parecendo redutor à primeira vista, este conceito torna-se extremamente útil, se pensarmos que todos os tipos de *softwares* anteriores ao aparecimento do IFC utilizavam cada um a sua estrutura de classes, não existindo uma uniformização. O IFC pretende permitir que todos os *softwares* usem uma base comum e, mais do que isso, que exista um consenso entre todos os agentes da indústria da construção, nas várias fases da vida de uma obra. Isto traduz-se em ganhos de tempo e consequentes ganhos financeiros [9].

2.2.3. TECNOLOGIAS *WEB* NA ENGENHARIA ESTRUTURAL

Com a evolução da internet para o paradigma de aplicações *web*, foi aberta a porta à criação plataformas que fizessem a interligação das técnicas de cálculo estrutural, para um mais vasto universo de utilizadores num ambiente colaborativo.

Esta "ponte" foi criada a partir da criação dos chamados *web services* (WS), que possibilitaram esta partilha de inteligência. Em termos gerais, podemos definir WS como *"uma função específica disponível na internet para disponibilizar um determinado serviço, ou para fornecer uma determinada informação"*. A palavra-chave para o uso de WS é integração. Diferentes agentes em diferentes espaços físicos podem partilhar informação através desta tecnologia, promovendo assim a integração de todos. A ideia de integração também se estende no sentido de que podemos integrar os WS no contexto das aplicações *web* [11]. Mais à frente neste documento irá ser dada uma explicação de como o WS produzido neste trabalho foi adotado pela aplicação *web Flange+Web.* No próximo capítulo será feita uma apresentação mais exaustiva das tecnologias dos WS.

Apesar de noutras áreas o conceito de WS já estar bastante maduro, no mundo da engenharia civil ainda há muito trabalho pela frente até ser atingido o ponto em que a indústria tire completo partido do mundo da web. Ainda assim, já existe algum desenvolvimento consumado, podendo-se destacar trabalhos nas áreas do cálculo por elementos finitos e em IFC/BIM.

No trabalho de [12] foi idealizado o *Web-FEM*, que se trata de uma *framework* para cálculo estrutural em elementos finitos em funcionando num formato WS. Este trabalho surgiu da constatação de que, apesar de já existirem alguns modelos nesta área, nenhum deles oferecia uma plataforma de visualização 3D em suporte de *browser* (aplicação *web*) que os tornasse populares (Fig. 2.4). A plataforma foi elaborada nas linguagens de programação *Java* e *C++*, que comportam a programação orientada a objetos, que também no caso do MEF é um conceito essencial.

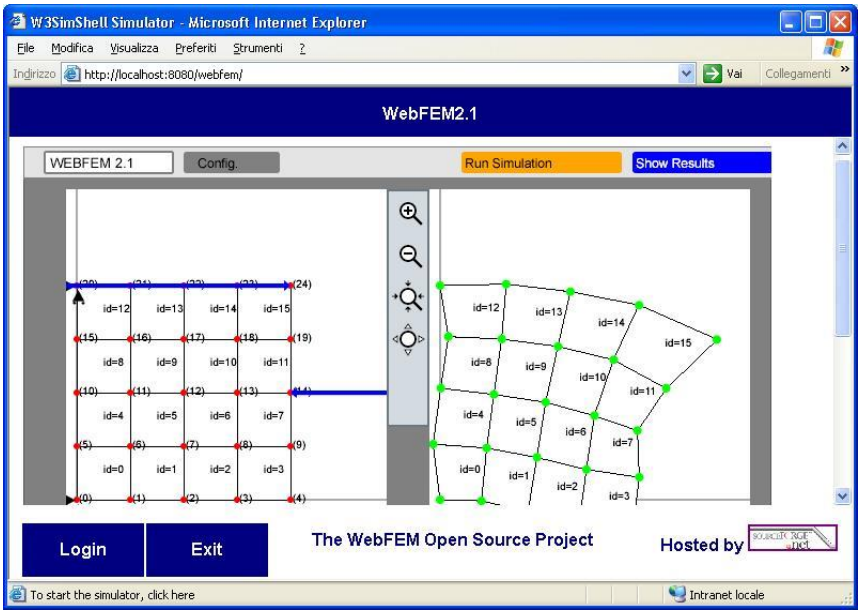


Fig. 2.4 - Aplicação para o cálculo do MEF [12]

Já na área do IFC esta aproximação entre os WS e mundo da engenharia civil tornou-se imperativa, dado que a internet possibilita uma maior integração de todos os agentes no processo da construção. Muitos trabalhos foram já realizados nesta área, porém ir-se-á destacar apenas o projeto *ifcwebserver* (Fig. 2.5 e Fig. 2.6). Apesar de não ter muitas funcionalidades disponíveis, esta aplicação *web* destaca-se por ter em si um WS que permite a criação de objetos de classes normalizadas de IFC, e alem disso a sua visualização com gráficos de qualidade assinalável, dada a utilização da tecnologia *WebGL*. [13].

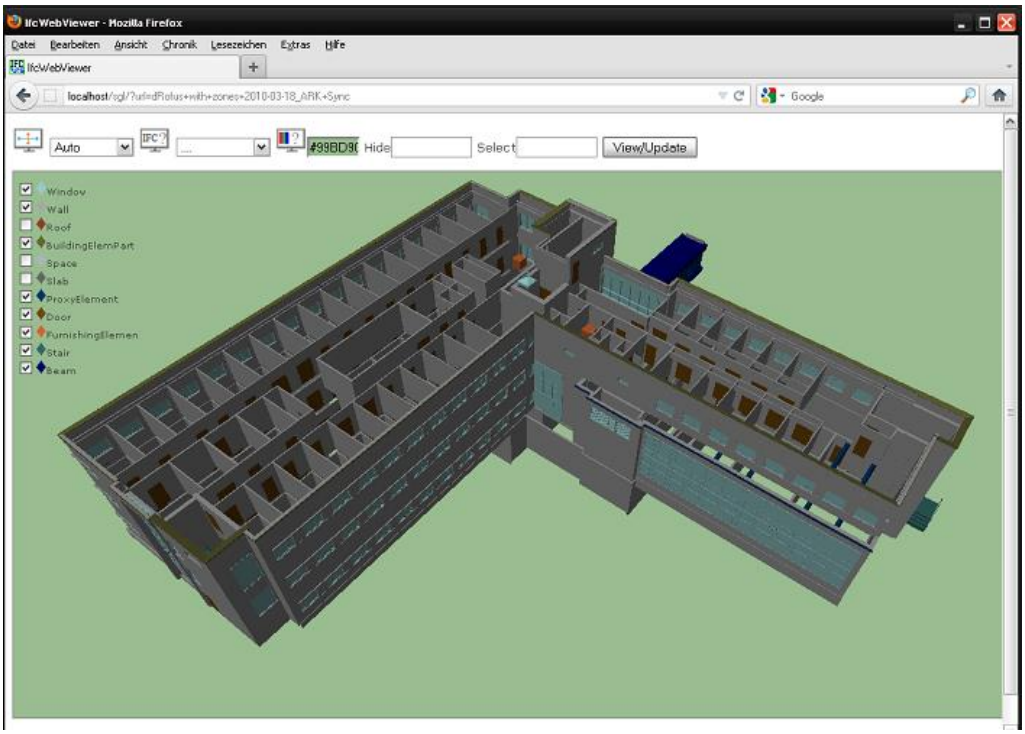


Fig. 2.5 - Aplicação *ifcserver* carregada em *browser*

Fig. 2.6 - Aplicação *ifcserver* carregada em *smartphone*

2.3. CONSIDERAÇÕES FINAIS

Neste capítulo procurou fazer-se uma contextualização da dissertação, dando a entender a necessidade da aproximação da engenharia civil às tecnologias de informação atualmente emergentes. Foi feita uma abordagem histórica, terminando com a contextualização atual do tema, dando referência a alguns exemplos de aplicações práticas já implementadas. No próximo capítulo abordar-se-ão em maior pormenor as tecnologias utilizadas na produção do WS realizado no âmbito desta dissertação.

3
TECNOLOGIAS ABORDADAS

3.1. INTRODUÇÃO

No presente capítulo serão introduzidos os principais conceitos e tecnologias utilizadas na concretização do *web service* (WS) para apoio no dimensionamento de estruturas metálicas. Será feita uma descrição detalhada das tecnologias utilizadas, explicando todos os conceitos e a justificação do seu uso.

3.2. PROGRAMAÇÃO ORIENTADA A OBJETOS

Existem inúmeros tipos de paradigmas de programação, uns mais conhecidos por terem sido implementados por linguagens que se tornaram mais famosas, outros menos conhecidos por serem de mais difícil aprendizagem. Um dos mais utilizados é a programação funcional, isto é, a definição de funções normalmente matemáticas com *inputs*, cuja função gera *outputs*. Este tipo de programação é normalmente o eleito pelos profissionais de engenharia. Tal facto é de natural compreensão, dado que a sua formação é orientada para o raciocínio matemático, onde são declaradas variáveis, que são utilizadas em funções que geram novas variáveis.

A programação orientada a objetos (em inglês *Object Oriented Programming* - OOP), paradigma surgido em finais dos anos 60 [14], toca em conceitos como classes ou objetos, pouco familiares no campo da engenharia civil. Dentro da OOP importa clarificar conceitos como o de objeto, classe, propriedades e métodos do objeto, hereditariedade, entre outros.

Pode definir-se objeto como um "individuo" que identifica um item, podendo esse item ser real ou abstrato. Cada objeto contém informação acerca de si próprio, e informações sobre a sua manipulação [14]. É oportuno então, saber como é gerado e definido cada objeto. Um objeto tem obrigatoriamente de pertencer a uma classe. Uma classe trata-se da representação de um determinado tipo de objetos, descreve todos os detalhes deste como forma de um diagrama, e é composta por três componentes: um nome, atributos e métodos [15].

Como referido anteriormente, podemos aplicar estes conceitos numa perspetiva real, e dando um exemplo prático podemos imaginar por exemplo a nossa mão direita como um objeto da classe mão. O corpo humano tem dois objetos da classe mão, um com o nome "mão direita" e o outro com o nome "mão esquerda". Repare-se que não foi necessário repetir a classe mão, pois os dois objetos têm as mesmas propriedades e funções, apesar de terem nomes diferentes. Um exemplo de uma propriedade da classe mão é por exemplo o facto de uma mão ter cinco dedos, e de um modo similar podemos definir um método da classe mão como por exemplo o ato de "fechar a mão" [15].

Outro conceito interessante da OOP é o conceito de hereditariedade. Imaginemos que existe agora uma classe denominada "membros do corpo humano", com a propriedade "tem células", e o método "receber sangue". Podemos afirmar que a classe "mão" e a classe "membro do corpo humano têm uma relação unívoca, pois todas as mãos são membros do corpo humano, mas nem todos os membros são mãos, logo a classe "mão" é uma subclasse da classe "membro do corpo humano". Podemos também estabelecer esta relação unívoca para as propriedades e métodos das classes, ou seja a classe "mão" terá igualmente a propriedade "tem células" e o método "receber sangue".

Como referido cada objeto é individual e singular depois de ser criado, devendo as suas propriedades serem inalteráveis, embora existam linguagem de programação que não o obriguem, conceptualmente esta a forma correta de utilizar a OOP. Continuando com a analogia com o corpo humano, imaginemos um objeto da classe "mão", em que esse objeto tem a propriedade "impressão digital". Como sabemos, cada impressão digital é irrepetível, e jamais essa propriedade pode ser alterada. Esta ideia transporta-nos para o conceito de encapsulamento, ou seja, de o objeto se isolar do resto do universo não deixando que nada o altere, sendo a sua única interação com o resto do universo a adição de propriedades e o uso dos seus métodos.

Esta vantagem associada à OOP da não repetição do código, e da sua reutilização infinita através da criação de objetos de uma determinada classe torna-se evidente no âmbito da informática. Como ficará claro no capítulo 5 desta dissertação, foi criada uma classe mãe *"Section"* de onde derivam algumas subclasses como secções de perfis comerciais ou soldados. Esses perfis soldados por sua vez são compostos por vários objetos da classe *"Plate"*. Esta abstração até à classe mãe também é uma das particularidades da OOP, em que é feita uma abstração da realidade até ao máximo patamar possível.

No capítulo anterior foram efetuadas inúmeras referências à OOP, nomeadamente aquando da introdução do conceito de IFC. Torna-se agora claro o porquê desta referência e, mais do que isso, o porquê do uso da OOP, dado que muitas das vantagens trazidas pelo IFC são totalmente válidas no contexto do WS produzido no âmbito desta dissertação, isto porque analogamente às classes estabelecidas pelo modelo IFC, também foram criadas classes de diferentes secções, e componentes das secções, que se articulam entre si.

3.3. LINGUAGENS DE PROGRAMAÇÃO

Uma linguagem de programação difere de uma linguagem natural apenas pelos seus intervenientes. Enquanto uma linguagem natural serve para a comunicação entre seres humanos, uma linguagem de programação serve fundamentalmente para a comunicação entre o Homem e a máquina. Essa comunicação pode ser mais ou menos descritiva das tarefas a executar pelo computador, e podemos dividir as linguagens de programação em dois grupos diferentes: as linguagens de alto e baixo nível.

O que difere estes dois grandes grupos é o nível de abstração. Por um lado, as linguagens de baixo nível estão sobretudo ligadas à performance do *hardware*, fazendo por exemplo a gestão da memória do computador, o que corresponde a um baixo nível de abstração. Nos primórdios da computação apenas este tipo de linguagem estava contemplado, que consistia numa sequência de bits que controlava diretamente o processador, adicionando, comparando ou movendo informação. Como se pode calcular, esta era uma tarefa árdua, que resultava muitas vezes em sucessivos erros. Este tipo de linguagens não lida com os conceitos, que hoje para nós são triviais, por exemplo de variáveis ou funções [16].

Desde os primórdios da computação houve a necessidade de criar uma forma de automatizar certas operações que os programadores exigiam às máquinas, e foi então que surgiram as linguagens

assembladas *(assembly languages)*. Contudo, ainda assim era necessário executar todas estas operações para cada computador em específico, pois a programação antes de surgirem os primeiros PCs, era centrada nas máquinas e não no Homem como hoje acontece. Foi então na década de 50 que surge a primeira linguagem de alto nível, a linguagem *Fortran.* Estas linguagens foram criadas com um sentido de usabilidade associado, onde é patente a preocupação em que um utilizador comum aprenda essa linguagem de uma forma simples, e onde existam uma série de funções já programadas facilmente acedidas pelos programadores (funções *built-in*). Com a evolução da tecnologia, começaram então a surgir as noções de ciclo, *if statement*, programação orientada a objetos, entre outros conceitos que ainda hoje são universalmente utilizados [16].

No caso da elaboração do WS, o que se procurava era naturalmente uma linguagem de alto nível que pudesse ser implementada do lado do servidor *(server side)*. Essa linguagem deveria ter a possibilidade de ser orientada a objetos, de ser de aprendizagem rápida e de fácil manuseamento, e sobretudo de ter disponível uma boa e extensa documentação e uma forte comunidade *online*.

Para além de satisfazer os requisitos anteriores, a linguagem deveria ser de interpretação e não compilada, que é o que acontece nas linguagens que são executadas em servidores *web*. A diferença entre estes dois paradigmas está na leitura do código. Enquanto nas linguagens compiladas o código é transformado em código binário e lido pela máquina, nas linguagens interpretadas é lida uma linha de código de cada vez existindo uma maior flexibilidade de leitura entre máquinas diferentes. Uma desvantagem da abordagem da interpretação, é que o código torna-se de leitura mais lenta [16].

Neste momento pode dizer-se que as três linguagens de programação *web* mais utilizadas e difundidas são o PHP, o *Python* e o *Ruby* [17]. Por uma questão de tempo, optou-se por uma focalização nas duas primeiras linguagens. Ambas trazem vantagens e desvantagens e não é consensual o uso de uma em detrimento da outra. Não é o objetivo deste documento dissecar cada uma destas linguagens e fazer uma abordagem do ponto de vista da ciência computacional, porém far-se-á de seguida fazer uma breve descrição da linguagem PHP e também da linguagem *Python*.

3.4. PHP *VS. PYTHON*

3.4.1. UTILIZAÇÕES DO PHP

Neste momento existem milhões de aplicações e páginas na internet implementadas em PHP. A principal vantagem que se pode extrair desta linguagem é a possibilidade de esta estar integrada em servidores HTTP Apache e por isso possibilitar um dinamismo na geração de páginas muito elevado. Além disso, esta linguagem detém uma vasta gama de bibliotecas e muitas delas que permitem uma conexão, pedidos e respostas a bases de dados SQL de uma forma muito simplificada, ao contrário de outras linguagens. Com a expansão e as potencialidades do PHP surgiu também o desenvolvimento de diversas *web frameworks* em PHP, que possibilitam o rápido desenvolvimento de aplicações web, como por exemplo o *Codeigniter* e o *Symfony*. Um exemplo disto mesmo foi a elaboração do *Flange+Web* por [1] que foi implementado em *Codeigniter*. Trata-se de uma forma muito rápida de obter aplicações web, que conjugandoo conhecimento de *Javascript* para gerar interatividade nas páginas, e de CSS *(Cascading Stryle Sheets)* que trata da parte gráfica das aplicações, consegue resultados bastante significativos num relativo curto espaço de tempo.

Existem largos exemplos de projetos criados em PHP. Os mais famosos são a rede social *Facebook*, a enciclopédia *online Wikipedia*, o *Wordpress* e o *Moodle*.

3.4.2. *PYTHON*

"O Python tem sido uma importante parte da Google desde o início, e continua a ser com o crescimento e evolução do sistema.". Segundo Peter Norvig, diretor da qualidade da pesquisa na Google, Inc. [18], hoje dezenas de engenheiros da Google usam o Python, e a empresa está à procura de mais pessoas com domínio na linguagem de programação.

Embora não tendo tantos seguidores como o PHP, o *Python* assume-se como uma das linguagens de programação mais em voga e expansão no mundo das aplicações web. Não sendo o mundo da internet o seu único destinatário, devido a inúmeros fatores, a linguagem de programação adotada pela Google e pelo Youtube assumiu-se como uma linguagem muito apelativa para o *server side scripting*. Entre esses fatores podemos destacar a facilidade de aprendizagem que o *Python* possibilita.

O panorama atual de aprendizagem da programação tem mudado radicalmente. Enquanto há uma década atrás o ensino da programação era feito sobretudo nas universidades, hoje assistimos a um aumento progressivo de cursos *online* de programação (e não só), em que inúmeras vezes têm o carimbo de universidades conceituadas. Entre eles podemos destacar o *coursera*, o *edX*, o *Udacity*, entre outros. Na plataforma *Udacity*, existe um curso denominado *"Intro to Computer Science"* onde é utilizado o *Python* para fazer uma introdução à programação. A linguagem *Python* é tão simples e intuitiva que um principiante da programação não terá qualquer dificuldade em ultrapassar a maioria dos desafios propostos. Este potencial de usabilidade, associado a uma vasta documentação e ao mesmo tempo uma grande comunidade *online*, tornam o *Python* uma linguagem bastante atrativa para novos programadores.

De modo similar ao PHP, foram desenvolvidas web *frameworks* para *Python*, sendo que a mais conhecida e poderosa é o *Django*. Mais à frente vamos abordar este tema, através do uso de uma *framework* denominada *Flask*.

3.4.3. COMPARAÇÃO ENTRE AS DUAS LINGUAGENS

Ambas as linguagens cumpririam todos os requisitos previamente estabelecidos para serem a linguagem escolhida para o *web service*. Contudo foram vários os fatores chave que levaram à escolha do *Python* em detrimento do PHP. Embora ambas fossem compatíveis com a programação orientada a objetos, que seria um critério de exclusão dado que havia a necessidade de serem criadas classes de diferentes secções no código, e também com a programação funcional que possibilita a criação de funções com parâmetros de entrada e saída, foi escolhido o *Python* sobretudo por três particularidades: em primeiro lugar, como já referido, a linguagem teria de ser de rápida aprendizagem, teria de ter poderosas ferramentas matemáticas disponíveis (como por exemplo o *SciPy*, o *Numpy* e o *Matplotlib*), e teria de existir uma *framework* adequada e também fácil de utilizar que permitisse uma rápida implementação do *web service*.

3.4.4. COMPONENTES DO CÓDIGO *PYTHON*

Para melhor compreensão de toda estrutura interna de dados do *web service*, que será abordada em maior detalhe no Capítulo 5, é importante perceber-se primeiro como esta foi implementada em *Python*. Neste subcapítulo será dada uma breve introdução acerca de cada componente utilizado desta linguagem.

3.4.4.1. Módulos e *Packages*

Existem várias estruturas de dados em *Python* que servem diversos propósitos. Primeiro de tudo é relevante indicar, que o código *Python* está contido dentro de ficheiros com uma extensão própria - *.py* se se tratarem de ficheiros para serem interpretados ou *.pyc/.pyo* no caso dos ficheiros terem sido compilados. No âmbito deste trabalho só interessam os ficheiros interpretados com extensão *.py*.

Contudo os ficheiros *Python* não são denominados por ficheiros, mas sim módulos. É possível dentro de cada módulo executar código existente em outro módulo através da expressão *import*. A divisão do código em diferentes módulos é essencial para uma programação organizada, onde o código é "partido" em fragmentos, para assim ser de melhor compreensão e de melhor correção de falhas. Porém, sem o conceito de *package* apenas é possível executar códigos que estejam em módulos da mesma pasta, ou seja, que estejam na mesma raiz. Esta dificuldade é ultrapassada com os *packeges*, que não são mais que uma pasta com módulos e subpastas inseridos. Porém, para a pasta funcionar como *package* deve conter um módulo com o nome *__init__.py*. Desta maneira todo o código contido em cada módulo dentro de cada *package* pode ser executado noutro módulo, através da expressão *"import package.módulo"* ou então *"from package import módulo"*. A Fig. 3.1 é elucidativa desta organização do código.

```
package/
    __init__.py
    file.py
    file2.py
    file3.py
    subpackage/
        __init__.py
        submodule1.py
        submodule2.py
```

Fig. 3.1 - Articulação entre *packages* e módulos em *Python*

Assim pode concluir-se que sem conceito de *package*, o módulo *file.py* não poderia "importar" código do módulo *submodule1.py* e vice-versa.

Este é o método para interligar o código em *Python*. Falta agora referir como é gerada a informação dentro de cada módulo, através dos conceitos habituais em programação de variável, função, classe, entre outros.

3.4.4.2. Variáveis

Tanto em Python como na maioria das linguagens de programação de alto nível, as variáveis são os elementos mais básicos onde pode ser alojada a informação. Em Python podem ser definidas variáveis como exemplificado na Fig. 3.2.

```
>>> x = y = z = 0  # Zero x, y and z
>>> x
0
>>> y
0
>>> z
0
```

Fig. 3.2 - *Python*: Variáveis [19]

Como é óbvio, este exemplo é abrangente e podiam ser definidas as variáveis uma a uma. Além disso pode dividir-se as variáveis em diversos tipos, sendo que os mais usuais são os tipos *integer* (um número inteiro), *float* (um número real) ou uma *string* (um conjunto de caracteres). Na Fig. 3.3 estão alguns exemplos de manipulação de strings.

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

Fig. 3.3 - *Python: Strings* [19]

Além disso podemos usar uma variável para definir mais do que um valor como na Fig. 3.4. A isto chamamos *tuple.*

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Fig. 3.4 - *Python: Tupples* [19]

Existem ainda mais duas estruturas de dados bastantes úteis: as listas e os dicionários *Python.* As listas correspondem a uma sequência de variáveis alojadas em diferentes posições da lista, que podem ser obtidas sabendo o valor do seu índice. Já um dicionário como o próprio nome indica, faz a correspondência entre uma chave e um ou mais valores. Nas Fig. 3.5 e Fig. 3.6 apresentam-se exemplos de listas e dicionários respetivamente:

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Fig. 3.5 - *Python: Listas* [19]

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Fig. 3.6 - *Python*: Dicionários [19]

3.4.4.3. Funções

A forma como são definidas funções (métodos) em *Python* está definida na Fig. 3.7.

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print a,
...         a, b = b, a+b
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Fig. 3.7 - *Python*: Funções [19]

As funções têm como característica a entrada de *inputs*, e a saída de *outputs*. Porém, isto nem sempre é verdade e podem existir funções sem nenhum dos dois conceitos referidos, ou com apenas um, existindo a função apenas para modificar o sistema, sem entrada e saída de variáveis. Na função anterior pode verificar-se que ela devolve a sequência de *Fibonacci*.

3.4.4.4. If statements

No mundo da programação é indispensável o recurso às estruturas de decisão com expressões Booleanas, e o *Python* não é uma exceção. A Fig. 3.8 ilustra bem a sua aplicação.

```
>>> x = int(raw_input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...      x = 0
...      print 'Negative changed to zero'
... elif x == 0:
...      print 'Zero'
... elif x == 1:
...      print 'Single'
... else:
...      print 'More'
...
More
```

Fig. 3.8 - *Python: If Statement* [19]

3.4.4.5. Ciclos *For* e *While*

Inúmeras vezes é indispensável o recurso a ferramentas que percorram um conjunto de informação de um modo iterativo, e isso é conseguido através dos ciclos *For* e *While* (Fig. 3.9 e Fig. 3.7 respetivamente).

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print w, len(w)
...
cat 3
window 6
defenestrate 12
```

Fig. 3.9 - *Python*: Ciclo *For* [19]

3.4.4.6. Classes e Objetos

Como foi referido, um dos fatores decisivos na adoção do *Python* como linguagem utilizada no *web service*, foi a possibilidade de se poder trabalhar com o paradigma de programação orientada a objetos. Pelas vantagens apresentadas por este tipo de programação, tornou-se obrigatório recorrer a criação de diversas classes para melhor manipular a informação. Em Python, a forma como é criada uma classe de objetos está representada na Fig. 3.10.

```
class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world'
```

Fig. 3.10 - *Python*: Classes [19]

Para ser gerado um objeto dessa classe é necessário o código da Fig. 3.11.

```
x = MyClass()
```

Fig. 3.11 - Python: Geração de objetos [19]

Este é um exemplo minimalista do que é uma classe de objetos. Como se pode verificar, não é introduzido nenhum parâmetro dentro da classe. O código pode ficar mais complexo quando se pretende atribuir características ao objeto em especifico, para isso é necessário colocar uma função

especial na classe, a função construtora: *__init__()*. Esta função define que parâmetros devem ser introduzidos no momento da geração do objeto como se ilustra na Fig. 3.12.

```
def __init__(self):
    self.data = []
```

Fig. 3.12 - *Python*: Construtor [19]

Deste modo é possível a geração de objetos com um grau de complexidade mais elevado à imagem da Fig. 3.13.

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Fig. 3.13 - *Python*: Exemplo mais complexo de geração de objetos [19]

É todavia essencial a referência a mais duas particularidades da OOP, o uso de instâncias e de métodos de cada objeto/classe. Pode-se pensar numa instância de uma classe como uma característica dessa classe que está associada a um determinado valor (Fig. 3.14).

```
x.counter = 1
while x.counter < 10:
    x.counter = x.counter * 2
print x.counter
del x.counter
```

Fig. 3.14 - Python: Instâncias das classes [19]

Como se pode verificar, foi atribuída uma instância ao objeto "x" anteriormente gerado com o nome de *"counter"*, e é-lhe atribuído o valor inicial "1", sendo que de seguida essa instancia do objeto é repetidamente invocada. Os métodos das classes podem ser analogamente invocados no código como representado na Fig. 3.15.

```
xf = x.f
while True:
    print xf()
```

Fig. 3.15 - Python: Métodos das classes [19]

3.4.5. *SCIPY*, *NUMPY* E *MATPLOTLIB*

Segundo James A. Hendler, professor da Universidade de Maryland:

"Eu tenho alunos a aprenderem Python nos cursos de programação web. Porquê? Basicamente porque não existe mais nada com a mesma flexibilidade e bibliotecas web" [18].

Para o cálculo das secções metálicas são necessários cálculos matemáticos algo elaborados e também a representação gráfica da curva de interação de esforço axial/momento fletor. No PHP não encontramos bibliotecas ou extensões que nos satisfaçam estas necessidades já que, como já foi referido, trata-se de uma linguagem direcionada para a geração de páginas web e não tanto para o desenvolvimento do *business logic* pretendido neste trabalho.

No entanto para o *Python* encontramos o *Scipy* e o *Numpy* que nos oferecem toda uma série de funções matemáticas, e o *Matplotlib* que possibilita a geração de gráficos de elevada qualidade através de comandos em *Python*. Estas três extensões colocam o *Python* num patamar semelhante ao do *MATLAB* no que toca ao cálculo matemático avançado, apesar de existirem algumas diferenças.

O *Scipy* foi criado para alargar a biblioteca matemática do *Python* e adiciona-lhe ferramentas por exemplo de álgebra linear e de integração. Já o *Numpy* permite a definição de *arrays* e matrizes de uma forma similar ao MATLAB, como mostra a Fig. 3.16, o que também ajuda no tempo de processamento de matrizes, quando comparamos os *arrays Numpy* com a alternativa que são as próprias listas do *Python*. O *Matplotlib* é uma biblioteca de geração de gráficos de todos os tipos, desde histogramas até dispersão de pontos [20]. Estas ferramentas combinadas foram um trunfo fundamental no rápido desenvolvimento da parte lógica do *web service*.

```
[[  6.21250000e+03  -5.53670960e+00             nan             nan             nan  -5.53670960e+00   0.00000000e+00   0.00000000e+00]
 [  4.97000000e+03   4.58228589e+02             nan             nan             nan   4.58228589e+02   3.01466000e+02   4.52257636e+02]
 [  3.72750000e+03   9.13726303e+02             nan             nan             nan   9.13726303e+02   6.02932000e+02   9.04515272e+02]
 [  2.48500000e+03   1.34898012e+03             nan             nan             nan   1.34898012e+03   9.04398000e+02   1.35677291e+03]
 [  1.24250000e+03   1.63259425e+03             nan             nan             nan   1.63259425e+03   1.20586400e+03   1.72707500e+03]
 [  0.00000000e+00             nan  1.72713229e+03             nan             nan   1.72713229e+03   1.50733000e+03   1.72707500e+03]
 [ -1.24250000e+03             nan             nan             nan  1.14822298e+03   1.63259425e+03   1.20586400e+03   1.14822298e+03]
 [ -2.48500000e+03             nan             nan             nan  7.89115962e+02   1.34898012e+03   9.04398000e+02   7.89115962e+02]
 [ -3.72750000e+03             nan             nan             nan  4.30008943e+02   9.13726303e+02   6.02932000e+02   4.30008943e+02]
 [ -4.97000000e+03             nan             nan             nan  7.09019236e+01   4.58228589e+02   3.01466000e+02   7.09019236e+01]
 [ -6.21250000e+03             nan             nan             nan  0.00000000e+00  -5.53670960e+00   0.00000000e+00   0.00000000e+00]]
```

Fig. 3.16 - Exemplo de um array Numpy usado no web service

3.5. BASE DE DADOS

Durante o desenvolvimento surgiu a necessidade da informação estar alojada de uma forma permanente, isto devido ao facto de existir informação que não necessita de ser calculada cada vez que é pedida pelo utilizador, como por exemplo informação relativa a perfis metálicos comerciais, dado que esta é uma informação tabelada e imutável. Esta poupança ao nível do cálculo estrutural reflete-se também no tempo de resposta.

3.5.1. *SQL – STRUCTURED QUERY LANGUAGE*

Existindo todavia outros tipos de bases de dados, as bases de dados do tipo SQL são as mais utilizadas no desenvolvimento de plataformas web. As bases de dados SQL são organizadas em diferentes tabelas que podem ter ou não dependências entre si. Cada tabela tem um determinado conjunto de campos, em que o primeiro é a identificação da linha (id), ou seja um número inteiro, e os restantes podem ter vários formatos como por exemplo *strings*, *floats*, data, hora, entre outros. Cada linha representa uma entrada de informação da tabela e pode haver ou não o requisito de serem preenchidos todos os campos. Esta forma de organizar a informação é sem dúvida uma das vantagens do SQL.

Podemos pensar em SQL como um tipo de bases de dados, mas também como uma linguagem de manipulação da informação. A operação mais comum que uma base de dados recebe é um pedido de informação *(query)*. Porém, existem ainda outras funcionalidades como inserir, alterar ou apagar dados. Em linguagem SQL fazer um *query* à base de dados corresponde a uma série de comandos simples e intuitivos como por exemplo: *"SELECT author, year, price FROM books WHERE title = "A Jangada de Pedra" ".* Neste caso existe na base de dados uma tabela denominada *books*, onde queremos saber informações sobre o autor, o ano da publicação e o preço do livro de José Saramago. Repare-se que não é imperativo que a tabela apenas contenha estas infirmações acerca dos livros, podendo existir outras como o número de páginas ou a cor da capa. Se quiséssemos toda a informação seria possível substituir *"author, year, price"* por *" * "*, e obteríamos todos os dados relativos ao livro

existentes na tabela. Um outro exemplo demonstrativo do potencial desta linguagem é o seguinte: *"SELECT title FROM books WHERE author = "José Saramago" AND year > 1985 ORDER BY title".* Neste caso seriam obtidas todas as obras do autor português desde o ano de 1985 ordenadas pelo seu título por ordem alfabética.

Toda esta metodologia SQL está disponível em vários softwares de bases de dados. Entre elas podemos destacar o *MySQL* e o *SQLite*.

3.5.2. BASE DE DADOS DO *FLANGE+WEB*

A base de dados referente aos perfis metálicos já havia sido criada para o protótipo *Flange+Web* em formato *MySQL*, não havendo necessidade de a replicar. Durante todo o desenvolvimento do WS, foi utilizada esta base de dados. Contudo, verificou-se que surgiriam vários problemas na continuidade da adoção deste tipo de base de dados. Como em futuro desenvolvimento esta pretende ser uma plataforma *open source* múltiplos indivíduos podem fazer alterações à base de dados. O problema está relacionado com o facto de a plataforma *MySQL* prever o alojamento da base de dados num servidor, o que acarretaria dificuldades na ótica de querermos que o desenvolvimento seja feito com a maior mobilidade possível, já que esse servidor pertence à FEUP, onde foram criadas *firewall* que não permitem modificações a quem não tem permissões para tal, tanto ao código como a bases de dados. A solução encontrada passaria por encontrar uma forma de a base de dados ser transportada através de um ficheiro, e isso foi alcançado através da conversão da base de dados do formato *MySQL*, para o formato *SQLite*. Esta passagem foi simples e não surgiram grandes contrapartidas ao nível de alterações no código *Python*, já que as bases de dados *MySQL* e *SQLite* partilham uma linguagem comum [21].

3.6. APIS E *WEB SERVICES* EM APLICAÇÕES *WEB*

3.6.1. CONCEITOS

Nos finais da década de 90 do século passado, era quase impreterível a um negócio de grande escala ter um *website* disponível para consulta dos seus clientes. Era uma época em que um negócio que não tivesse esta plataforma quase certamente não alcançaria bons resultados. Isto devido, talvez à curiosidade nesta tecnologia, os consumidores iam ao encontro do serviço, e não o contrário.

Embora esta tendência se mantenha, o mundo tornou-se mais complexo desde então, tornando-se mais fragmentado. Devido à proliferação dos meios de comunicação e dos dispositivos existentes no mercado das comunicações, neste momento existe um excedente de oferta. Torna-se portanto essencial uma aposta na qualidade e na pluralidade dos serviços fornecidos, isto para cativar cada vez mais audiência. Deste modo, passamos de um paradigma onde com baixos recursos teríamos clientes garantidos, para um paradigma onde a audiência é cada vez mais exigente e faz uma triagem mais seletiva dos serviços utilizados.

Caminhamos então num rumo, onde como expoente máximo, estará a livre (gratuita) utilização dos serviços, em termos de tempo, espaço e qualidade.

Antes da introdução ao conceito de API, é interessante fazer-se uma analogia com outra indústria para se compreender melhor a importância das APIs. Imagine-se então um veículo. Antigamente os veículos eram fabricados apenas por uma só marca que depois o vendia. Nos dias de hoje isto já não sucede. Como é sabido, um veículo corresponde a um conjunto bastante alargado de componentes, componentes esses que muitas vezes são fabricados por companhias distintas. Isto acontece porque os

fabricantes tendem em especificar-se numa área em específico, e não no todo, porque perceberam que tinham vantagens em trabalhar focados apenas num componente, e posteriormente fazerem uma assemblagem. Tirando partido da informação dos outros, é assim criado um produto final próprio, com ganhos para todos os intervenientes.

Este fenómeno é visto em muitas outras indústrias, e tem real impacto também na informática. Surge então o conceito de API *(Application Programming Interface)* que, como o próprio nome indica, está definido como um conjunto de regras e especificações que um *software* pode seguir para aceder e fazer uso de serviços e recursos providos por outro *software* onde está implementada a API. Uma API funciona como a interface entre *softwares* e facilita a sua interação, de modo similar à interface entre computadores e as pessoas. Um *web service* representa um caso particular de uma API, onde o referido conjunto de regras e especificações são transmitidas via web, tratando-se de uma API remota como demonstrado na Fig. 3.17. Contudo, o conceito de API é anterior à massificação da Internet.

Fig. 3.17 - Interação entre o cliente e as APIs

Numa primeira fase, as APIs apareceram no contexto dos sistemas operativos. Como as grandes companhias de *software*, como a *Microsoft*, pretendiam ganhar a hegemonia do mercado de sistemas operativos, foi feito um grande esforço no sentido de serem desenvolvidas APIs para servirem de base à criação de aplicações para esses sistemas operativos, tendo como objetivo ganhar mercado, pois cada aplicação só funcionaria naquele sistema operativo, um pouco à imagem do que acontecia com os automóveis no exemplo dado.

Só mais tarde, com a expansão da *web*, surgiram as *web* APIs começando igualmente pelas grandes companhias como a *eBay*, *Amazon*, ou a *Google*. Porém, o fenómeno alastrou-se e hoje assistimos ao surgimento de ideias de negócio com base na criação de uma API. Essa pode estar acessível *online*, passando então a poder-se denominar de *web service* [22]. A Tabela 1 mostra a evolução das APIs ao longo dos anos.

Tabela 1 - História das APIs [22]

Categoria da API	Exemplo	Data
Sistema operativo	API para o Microsoft Windows	1985
	API para o Apple Mac OS X	2001
Linguagens de programação	API para Java	1995
Application Services	API para SAP	1990
Infrastructure Services	API para o Amazon Web Service	2002
Web Services	API para o Twitter	2006

Uma boa API deve deixar a complexidade para quem a programou e apenas deixar para os utilizadores funções que sejam de fácil manejo para os seus próprios desenvolvimentos. Alem disso, a API deve possuir uma clara e extensa documentação para futuros desenvolvimentos. Existem vastos exemplos de sucesso de APIs como por exemplo as APIs da *Google* como a do Youtube que permite a integração de vídeos em qualquer aplicação, ou a API do *Google Maps* que faz com que possam ser visualizados mapas e pedir direções. Também o *Facebook* apostou numa API que permitisse o desenvolvimento paralelo de aplicações, o que ajudou em muito a expansão do projeto. Já a API disponibilizada pela plataforma de meteorologia *Wheather Underground* sempre foi uma referência no desenvolvimento da API produzida ao longo desta dissertação, já que a forma como é disponibilizada a informação assemelha-se à idealização que havia à partida.

Hoje existe uma grande oferta de APIs cujos serviços alimentam ideias e projetos que oferecem serviços de uma ordem completamente diferente. Isto transporta-nos à ideia de *mashup*. Este conceito tem-se demonstrado fundamental para o sucesso de muitas aplicações web, que ao fundirem funcionalidades de diferentes APIs conseguem criar produtos inexistentes no mercado. Este benefício para os programadores que agora apenas têm de se preocupar com a própria API, tem vindo exponenciar o número de aplicações *web*. Um exemplo claro de sucesso de *mashups*, são as aplicações criadas sobre a API do *Google Maps*, como por exemplo o *Foursquare*. Esta aplicação usa os mapas e o sistema de localização da *Google* para que os utilizadores possam fazer *"check in"* nos locais por onde passam. A *Google* disponibiliza uma consola que permite manipular as API que estamos a utilizar em determinado projeto, como é representado na Fig. 3.18. A Fig. 3.19 mostra o processo de desenvolvimento de aplicações e APIs desde o seu embrião, até à sua proliferação.

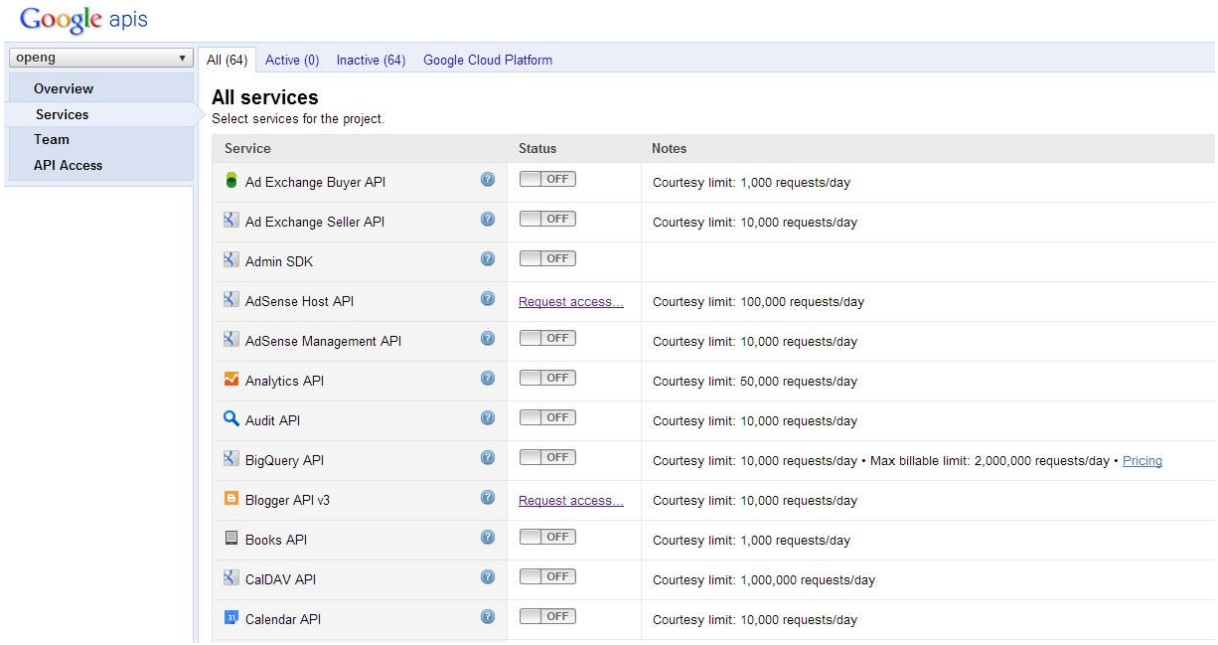


Fig. 3.18 - Consola *Google* para a gestão do uso de APIs da *Google* numa aplicação

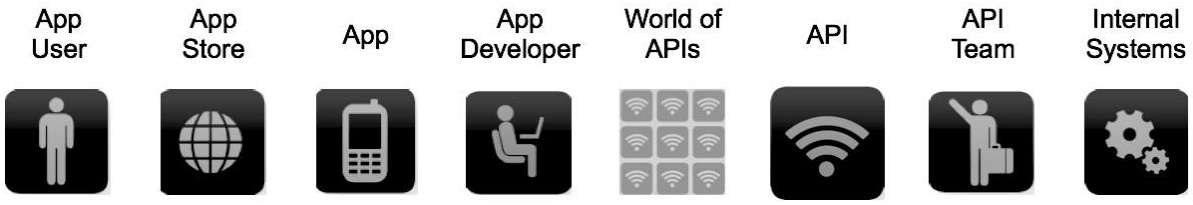


Fig. 3.19 - Processo de desenvolvimento de APIs [23]

3.6.2. REST - *REPRESENTATIONAL STATE TRANSFER*

Pode dizer-se que as APIs servem de base a qualquer tipo de aplicação *web*, ou seja, para uma aplicação *web* funcionar terá de ter pelo menos uma API, ou então um *web service* remoto. A diferença de funcionamento entre os *web services* e as aplicações *web*, é que as aplicações respondem a pedidos HTTP em HTML que é interpretado pelo *browser*, enquanto nos *web services* a informação (texto, listas, imagens, entre outros) é transmitida ou não (pode não ser gerada nenhuma resposta), pelo servidor para o cliente, em formato XML ou em formato JSON, consoante o caso [4].

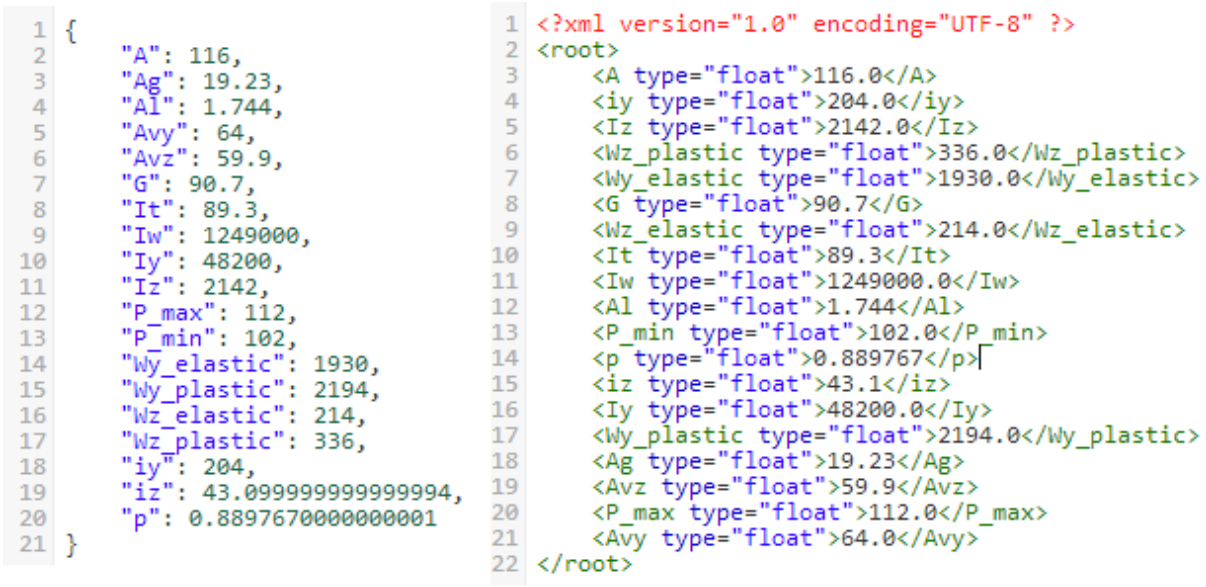
No caso do WS produzido ao longo deste trabalho, procurou-se seguir uma metodologia que nos últimos anos tem vindo a crescer e a ganhar adeptos no desenvolvimento de *web services*, metodologia essa denominada REST *Representational State Transfer*. O REST é um estilo de arquitetura, não é uma norma e permite uma grande flexibilidade. Foi esta flexibilidade que levou à sua adoção no *web service* desenvolvido. O sucesso de uma API/WS pode medir-se na rapidez em que os utilizadores a dominam, para isso há que manter a interface bastante simples e intuitiva, uma das características do REST [23].

Aqui estão algumas características do REST [4]:

- As operações são baseadas em métodos HTTP (mais à frente especificados), como por exemplo *GET* para receber uma informação, *POST* para enviar informação ao servidor ou *DELETE* para apagar informação. Em REST é frequente chamar estes métodos de *verbos*;
- A informação que o utilizador pretende é especificada no URL, sob forma de coleções, no caso do WS deste trabalho "*steelsections*/*comercial"* por exemplo;
- As respostas são dadas através de XML ou JSON, tal como foi implementado no *web service* produzido.

O JSON *(Javascript Object Notation)* é um formato de dados que permite o fácil transporte de informação entre plataformas. É uma estrutura de dados de fácil leitura por parte dos programadores, pois agrupa a informação de uma forma intuitiva, e de igual modo muito fácil de analisar e gerar por parte dos computadores. Como esta estrutura de dados tem origem na linguagem *Javascript*, torna-se bastante exequível a articulação com esta linguagem usada na criação de aplicações *web*. A informação pode ser agrupada de duas formas: numa lista onde é emparelhado o nome e o respetivo valor que por sua vez pode ser outra lista, ou então sob forma de *array*, usado mais para fins matemáticos para representar matrizes [24].

À semelhança do JSON, a estrutura de dados XML serve o transporte de dados entre várias plataformas, sendo que a sua grande vantagem é a fácil articulação com o HTML, pois ambos partilham a mesma notação. Esta articulação é essencial na geração de dinamismo nas páginas *web* [25]. A Fig. 3.20 mostra a confrontação das duas estruturas de dados anteriormente mencionadas.



```json
{
    "A": 116,
    "Ag": 19.23,
    "Al": 1.744,
    "Avy": 64,
    "Avz": 59.9,
    "G": 90.7,
    "It": 89.3,
    "Iw": 1249000,
    "Iy": 48200,
    "Iz": 2142,
    "P_max": 112,
    "P_min": 102,
    "Wy_elastic": 1930,
    "Wy_plastic": 2194,
    "Wz_elastic": 214,
    "Wz_plastic": 336,
    "iy": 204,
    "iz": 43.099999999999994,
    "p": 0.8897670000000001
}
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
    <A type="float">116.0</A>
    <iy type="float">204.0</iy>
    <Iz type="float">2142.0</Iz>
    <Wz_plastic type="float">336.0</Wz_plastic>
    <Wy_elastic type="float">1930.0</Wy_elastic>
    <G type="float">90.7</G>
    <Wz_elastic type="float">214.0</Wz_elastic>
    <It type="float">89.3</It>
    <Iw type="float">1249000.0</Iw>
    <Al type="float">1.744</Al>
    <P_min type="float">102.0</P_min>
    <p type="float">0.889767</p>
    <iz type="float">43.1</iz>
    <Iy type="float">48200.0</Iy>
    <Wy_plastic type="float">2194.0</Wy_plastic>
    <Ag type="float">19.23</Ag>
    <Avz type="float">59.9</Avz>
    <P_max type="float">112.0</P_max>
    <Avy type="float">64.0</Avy>
</root>
```

Fig. 3.20 - Estruturas de dados. À esquerda JSON, e à direita XML

3.6.3. EXEMPLOS NA ENGENHARIA ESTRUTURAL

No campo da engenharia estrutural o conceito de API encontra-se pouco desenvolvido. Os únicos exemplos significativos encontrados são as APIs de alguns programas comerciais como o conhecido *Autodesk Robot Structural Analysis*. Contudo, mesmo nestes casos, nunca houve uma implementação num contexto *web*, ou seja não é possível criar aplicações *web* com base nas análises da API do programa [26].

3.7. *FLASK MICROFRAMEWORK*

Como anteriormente referido o uso de uma *web framework* torna-se no presente caso essencial. Isto porque para passagem do paradigma de API para o de *web service*, existe a necessidade de criar conectividade através da rede, e isso consegue-se no presente caso através do protocolo HTTP.

3.7.1. *HTTP - HYPERTEXT TRANSFER PROTOCOL*

Este protocolo de pedido/resposta, implementado no início da década de noventa tem como objetivo estabelecer regras e padrões para as comunicações na web entre clientes e servidores. Cada pedido ou resposta HTTP divide-se em duas partes: o *header* e o *body*. No *header* vão incluídas informação relativas ao pedido que não são mostradas ao utilizador tais como, o *status* do pedido (os mais conhecidos *"200 OK"* e o *"404 Not Found"*), ou *cookies* que normalmente correspondem a informação privada do utilizador. No *body* vai toda a informação em formato HTML que é interpretada pelo *browser* e disponibilizada ao utilizador.

Através de um *browser* é possível com o URL efetuar pedidos a um servidor através dos métodos HTTP. Entre os métodos disponíveis no protocolo, destacam-se os métodos *GET* e *POST*. O método *POST* é usado normalmente na submissão de formulários em que a informação vai embutida no corpo da mensagem, como por exemplo o ato de fazer login numa determinada página corresponde ao envio da informação através do método *POST* em que detalhes como o *username* e a senha são enviados como *cookies*. O método *GET* apenas faz pedidos no sentido de ser retribuída uma resposta por parte

do servidor com a informação solicitada, ou seja não há inserção de dados no servidor. No método GET, a informação que o servidor necessita vai explicita no próprio URL, onde neste caso podemos introduzir vários parâmetros na denominada *query string* [27]. Um exemplo prático de um pedido GET, é por exemplo uma pesquisa no motor de pesquisa da empresa Google. Ao fazermos a pesquisa de informação relativa às palavras *"engineering web applications"*, podemos facilmente reparar que o URL do pedido é o que podemos verificar na Fig. 3.21.

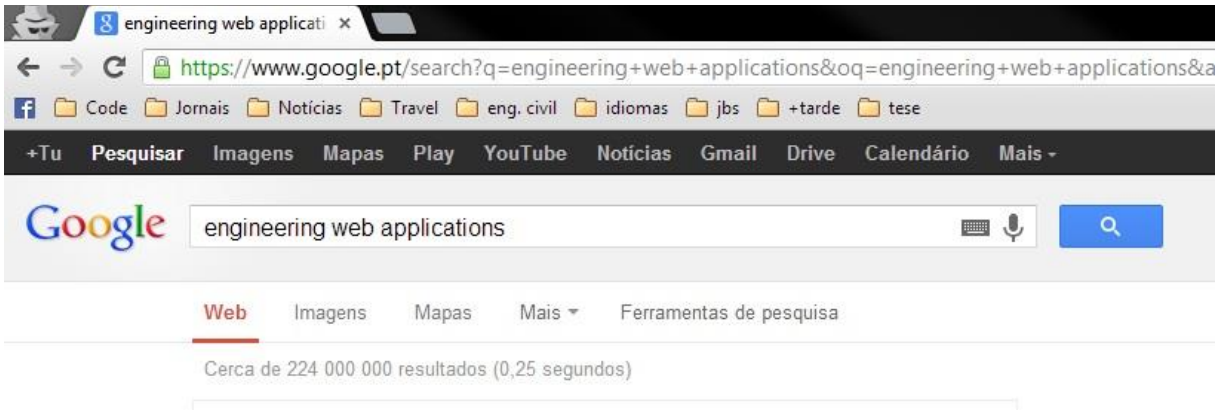


Fig. 3.21 - Exemplo de endereço URL, elucidativo de um pedido GET

Neste pedido GET pode-se identificar o parâmetro "q" que corresponde a uma *string* que é pesquisada na base de dados do *Google*.

3.7.2. De API a *Web Service*

Programar a receção de pedidos e a emissão de respostas sob o protocolo HTTP não seria concordante com o objetivo desta dissertação. Portanto é mais que oportuno o uso de uma *web framework* que nos facilite essa tarefa, caso contrário haveria a necessidade de passar por um processo de aprendizagem para dominar estas tecnologias, que se poderia demonstrar moroso. Pelos mesmos motivos utilizar uma *framework* demasiado complexa poderia igualmente representar uma dificuldade. Tornou-se assim imperativo o uso de uma *framework* em que não fossem necessárias configurações bastante rebuscadas, onde os pedidos e respostas HTTP fossem de rápida integração no código *Python*, e onde acima de tudo imperasse a simplicidade nos processos.

Todos esses pressupostos foram garantidos desde as primeiras interações com a *web framework Flask* [28]. Esta web *microframework* que tem como *slogan "web development one drop at a time"*, é na sua essência simples, intuitiva e tem a vantagem de se tornar complexa apenas se se quiser adicionar extensões de apoio como por exemplo de login, de autenticação, e de validação, entre outros.

3.7.3. *Model-View-Controller*

Embora o *Flask* não esteja padronizado para receber esta arquitetura de dados, esta foi utilizada no web service para implementar o código. *Frameworks* mais robustas como o *Codeigniter* ou o *Django* utilizam esta arquitetura por defeito, devido a esta ser bastante intuitiva para os programadores. No caso do *web service* esta arquitetura foi criada de raiz, devido ao facto já referido de o *Flask* ser uma web *framework* "minimalista", em que tudo o resto são extensões.

Fazendo uma abordagem mais minuciosa acerca desta arquitetura de programação, em primeiro lugar há que dissecar os seus vários componentes.

O *controller* é a ligação entre o utilizador e sistema. A sua função é receber a informação proveniente do utilizador *(input)* e dar o seu seguimento correto. O *model* é onde está armazenada toda a

informação lógica do código. Deste modo este componente recebe informação do *controller* o processa os dados para de seguida serem devolvidos ao utilizador. Para este receber a informação numa forma percetível, os dados necessitam de passar por um filtro que os converta numa linguagem própria de ser interpretada pelo *browser* e de seguida pelo utilizador. A este filtro chamamos *view*, que normalmente nas aplicações web se trata de um *template* HTML que é preenchido com informações providas do *model* [29].

No caso do *web service* produzido a estrutura seguida é na sua essência a exposta acima, apenas com a diferença de que o *view* não é no nosso caso um *template* HTML, mas uma estrutura de dados em *Python* que depois é convertida no formato JSON ou XML. O fluxo de dados é iniciado no *controller* onde são criados objetos de classes pertencentes aos *models* (classes *Commercial* e *I2sym*), que depois são inseridos na *view*. A *view* filtra os atributos dos objetos que são importantes, e com eles constrói um dicionário *Python*, que depois é convertido para formato JSON ou XML para ser disponibilizado ao utilizador. Neste capítulo foi dado a conhecer um pouco da estrutura do código para explicar o conceito de MVC, porem essa estrutura será abordada em maior pormenor no capítulo 5.

Durante o desenvolvimento ficou clara a necessidade de separação da informação relativa a engenharia civil do resto da estrutura de dados. Isto deve-se ao facto existir a possibilidade de futuros desenvolvimentos em que fosse necessário a utilização de objetos de classes já criadas. Para esta utilização não faria sentido o uso em desenvolvimento num contexto web, podendo estes módulos serem acedidos diretamente e usados para novas implantações ao nível da API. Então foi tomada a decisão de colocar o componente *model* fora do contexto da *framwork*, e funcionando como uma biblioteca *Python* que igualmente pode ser acedida através dos *controllers*.

3.8. *WEB SERVICE*

O desenvolvimento do *web service* tem como objetivo criar uma base expansível para a criação de aplicações. Neste momento a API já contém alguma variedade de funções disponíveis. Entre elas estão o cálculo de propriedades de perfis metálicos comerciais (perfis HE e IPE, e também os perfis britânicos UC e UB), e também de perfis soldados. As funções disponíveis também permitem a representação da curva de interação de esforço axial/momento fletor através do *Matplotlib*, o que representa uma inovação no campo das estruturas metálicas. Estas curvas têm em consideração disposições regulamentares previstas no EC3, aspetos que serão abordados no Capítulo 4. Apesar dos resultados alcançados, o potencial da API não tem ainda fim à vista, tendo até várias direções a poder tomar.

Neste momento todos os dados disponibilizados pelo *Flange+Web* são solicitados via *web service*, e não como anteriormente através do próprio código em PHP. Além disso foi adicionada a funcionalidade da visualização das curvas de interação, e do cálculo da verificação de segurança, através da receção da representação gráfica com recurso a bibliotecas *Javascript* que permitem a geração de gráficos deste tipo. Como é óbvio, este não é o percurso natural para uma aplicação *web*, porém desta forma proporcionamos ao *Flange+Web* uma evolução muito mais sustentada através do recurso ao *web service*, isto porque deste modo o desenvolvimento das funções de engenharia e da própria aplicação podem ser feitos em separado, e assim assim a aplicação crescer em paralelo com a biblioteca de funções.

3.9. CONSIDERAÇÕES FINAIS

No decurso deste capítulo foram introduzidos os conceitos que estiveram por detrás do desenvolvimento do *web service*. Fez-se uma descrição das tecnologias utilizadas, e a justificação do seu uso neste contexto em específico, tentando sempre dar exemplos práticos da sua aplicação. Demonstrou-se a importância da programação orientada a objetos para o *web service*, assim como o *Python*, as bases de dados SQL, a arquitetura REST, a *web framework Flask* e a arquitetura interna MVC.

No próximo capítulo será aprofundado o estudo das secções metálicas no contexto do Eurocódigo 3, e serão introduzidas as curvas de interação de esforço axial/momento fletor.

4
CÁLCULO DE SECÇÕES METÁLICAS SEGUNDO O EUROCÓDIGO 3

4.1. INTRODUÇÃO

Os Eurocódigos estruturais surgiram da preocupação de a Comissão Europeia regulamentar o sector da construção. Em 1990 este órgão delegou no CEN (Comité Europeu de Normalização) a criação de normas que padronizassem a indústria da construção no continente europeu, e daí provieram dez documentos:

- EN 1990 Eurocódigo: Bases de Projeto;
- EN 1991 Eurocódigo 1: Ações em Estruturas;
- EN 1992 Eurocódigo 2: Projeto de Estruturas de Betão;
- EN 1993 Eurocódigo 3: Projeto de Estruturas de Aço;
- EN 1994 Eurocódigo 4: Projeto de Estruturas Mistas Aço-Betão;
- EN 1995 Eurocódigo 5: Projeto de Estruturas de Madeira;
- EN 1996 Eurocódigo 6: Projeto de Estruturas de Alvenaria;
- EN 1997 Eurocódigo 7: Projeto Geotécnico;
- EN 1998 Eurocódigo 8: Disposições para Projeto de Estruturas Resistentes aos Sismos;
- EN 1999 Eurocódigo 9: Projeto de Estruturas de Alumínio.

Dentro de cada uma destas normas está ainda prevista a adequação e contextualização a cada país da comunidade.

Embora ainda não tivesse sido referido anteriormente neste documento, todo o tipo de cálculo estrutural presente neste trabalho tem por base a norma vigente em território português o Eurocódigo 3 [30], já que apenas são abordadas estruturas metálicas. Este documento estabelece regras que devem ser seguidas pelos profissionais de modo a serem cumpridos vários requisitos como por exemplo a estabilidade estrutural e o bom funcionamento em serviço. O âmbito do trabalho está compreendido na primeira parte do EC3 (EN 1993-1-1: Regras gerais e regras para edifícios), na qual estão definidas as disposições previstas para o dimensionamento de estruturas metálicas convencionais.

4.2. ANÁLISE E DIMENSIONAMENTO DE ESTRUTURAS METÁLICAS

4.2.1. O MATERIAL AÇO

No contexto deste capítulo é indispensável fazer uma caracterização dos materiais que estão a ser utilizados nas análises. No *web service* apenas ainda está contemplado a utilização do material aço, pois o pretendido é o cálculo de diversas características de perfis metálicos. Porém, como demonstrado no capítulo anterior, a programação orientada a objetos permite-nos uma fácil expensão do universo, e será com facilidade que a integração de outros materiais, como por exemplo o betão, será conseguida no contexto do WS. Tal como no código do WS, onde um objeto da classe Material tem vários atributos como a sua resistência, também no mundo real é necessário definir essas propriedades.

Desde a sua extração da natureza o minério de ferro sofre um processo de tratamento que permite a redução do teor de carbono da sua composição. Para obter o material aço é necessário baixar o teor de carbono para menos de 2%, pois valores acima deste limite resultam no material de ferro fundido. Assim o aço trata-se um composto de ferro e carbono, juntamente com outras substâncias que lhe conferem diversas propriedades. Os perfis comerciais são perfis que surgem normalmente do processo de laminagem a quente cujo aço contêm uma percentagem de carbono de cerca de 0.2%. Interessa-nos conhecer o valor da sua resistência, o que pode ser encontrado na norma EN10025-2 (2004), onde f_y e f_u representam os valores característicos das tensões de cedência e de rotura respetivamente:

A Tabela 2 define os valores característicos R_k dos valores de f_y e f_u presentes no EC3.

Tabela 2 - Resistências para cada classe de aço estrutural segundo EN10025-2

Norma e Classe do Aço	Espessura nominal *t* do componente da secção [mm]			
	$t \leq 40$		$40 \leq t \leq 80$	
	f_y [N/mm^2]	f_u [N/mm^2]	f_y [N/mm^2]	f_u [N/mm^2]
S 235	235	360	215	360
S 275	275	430	255	410
S 355	355	510	355	470
S 450	440	550	410	500

Contudo a norma prevê ainda a aplicação de coeficientes de segurança γ_M para redução destes valores, o que os torna consequentemente em valores de cálculo R_d, e portanto elegíveis a serem aplicados em análise estrutural, como indica a expressão (1).

$$R_d = \frac{R_k}{\gamma_M} \tag{1}$$

O Anexo Nacional do EC3 estabelece o valor do coeficiente de segurança γ_{M0} igual à unidade. Este coeficiente deve ser aplicado em secções transversais de edifícios correntes.

4.2.2. MÉTODOS DE ANÁLISE ESTRUTURAL

Na engenharia de estruturas encontramos vários métodos de análise que se classificam do seguinte modo:

- Análises lineares, em que é considerado o comportamento elástico linear dos materiais, e onde o equilíbrio da estrutura é imposto na configuração indeformada (análise de primeira ordem);
- Analises não lineares, nas quais é considerada a não linearidade física dos materiais, e também a não linearidade geométrica, isto é, a consideração do equilíbrio na configuração deformada da estrutura (análise de segunda ordem);
- Pode também ocorrer que apenas seja considerada uma das componentes de não linearidade citadas.

Os métodos de análise linear são os mais utilizados para analisar estruturas metálicas. Neste tipo de análise aceita-se a reversibilidade das deformações e a sobreposição dos efeitos das diversas ações. A sua aplicação não necessita do dimensionamento exato da estrutura. Uma análise estrutural linear permite verificar com rigor suficiente, por exemplo, as especificidades requeridas numa verificação do estado limite último, e também na grande maioria dos casos o estado limite de serviço.

Quando é feita uma análise estrutural não linear não deve ser contemplado o princípio da sobreposição dos efeitos, e a resposta estrutural depende da história das cargas aplicadas na estrutura, já que não existe uma proporcionalidade entre a ação e as respostas. Geralmente, uma análise desta tipologia implica um processo iterativo de sucessivas análises lineares até um ponto de convergência, e serve normalmente para tirar partido das propriedades plásticas dos materiais. A não linearidade tem início em fases avançadas de carga ao ser alcançado o estado limite elástico do material em algumas fibras, ou então ao serem desencadeados fenómenos de encurvadura local nas chapas mais esbeltas das secções transversais. Neste caso é importante conhecer com detalhe todas as dimensões da estrutura ao pormenor.

4.2.3. BASES PARA A CLASSIFICAÇÃO DE SECÇÕES METÁLICAS

É essencial uma análise rigorosa dos esforços de cálculo de uma estrutura, assim como a determinação da capacidade resistente dos seus elementos. Sendo as estruturas metálicas o alvo de estudo deste trabalho, a análise global de esforços e deslocamentos numa estrutura está diretamente dependente das suas características de deformabilidade e rigidez, assim como da estabilidade global e local dos seus elementos como o comportamento das secções transversais, das ligações, das imperfeições e da deformabilidade dos apoios. Assim, podem fazer-se dois tipos de análises distintas: uma análise global elástica, ou uma análise global plástica.

Uma análise global elástica baseia-se na hipótese de a relação tensão-deformação do material ser linear para qualquer tipo de tensão atuante. Esta é a análise que deve ser aplicada a estruturas isostáticas, já que estas estruturas atingem o colapso assim que em algum ponto é atingido o seu limite resistente.

Já as análises globais plásticas trazem algumas vantagens relativamente às anteriores. As propriedades do material aço conferem-lhe uma especial aptidão, à adoção de métodos de dimensionamento plástico, isto devido à existência de um patamar de cedência, e extensões garantidas superiores a 15%, $\varepsilon_u \geq 15\varepsilon_y$ e $f_u/f_y \geq 1.10$, tal como estipulado na cláusula 3.2.2 da parte 1-1 do EC3. A isto juntando um comportamento razoavelmente aproximado por uma lei constitutiva elasto-plástica, torna-se possível

fazer-se uma avaliação da resistência das secções assumindo-se a sua plastificação total, mas também avaliar a capacidade resistente global de uma estrutura tirando partido da sua hiperestaticidade e da formação de sucessivas rótulas plásticas até à formação de um mecanismo e consequente colapso (Fig. 4.1). Este aproveitamento da capacidade plástica dos materiais resulta geralmente em estruturas mais económicas [31].

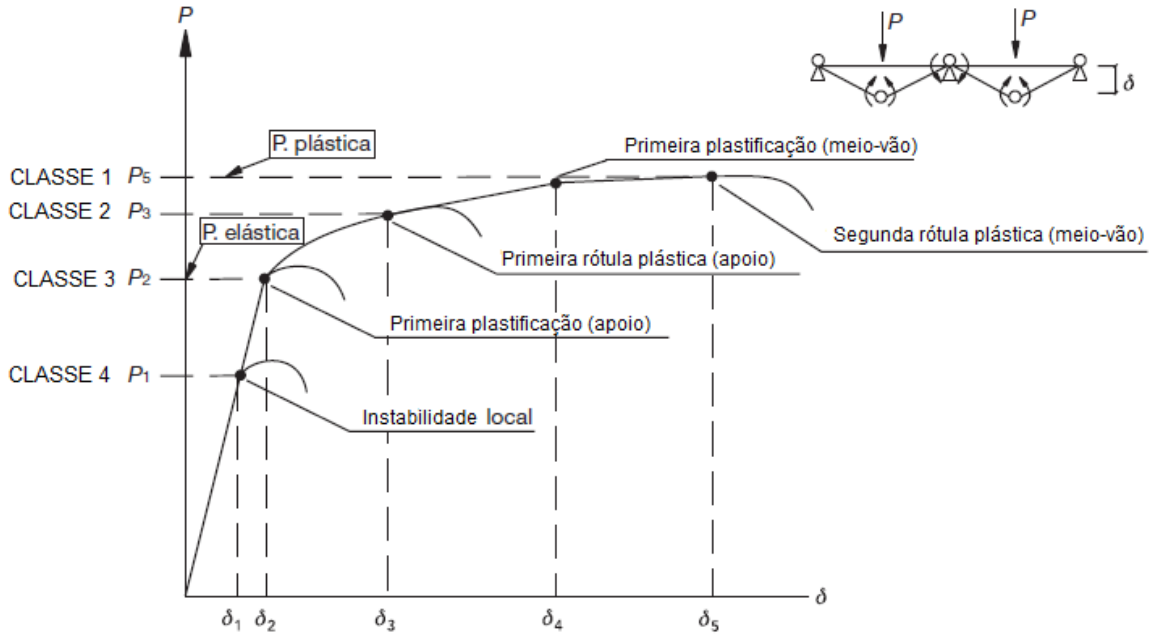


Fig. 4.1 – Aproveitamento da hiperestaticidade das estruturas através de uma análise plástica [32] (adaptado)

De modo a evitar eventuais roturas frágeis geradas devido ao aproveitamento das propriedades não lineares do aço nas zonas de formação de rótulas plásticas, a utilização da análise global plástica está sujeita a alguns condicionamentos. Em oposição à análise global elástica, a análise global plástica apenas pode ser usada como ferramenta quando a estrutura exibir uma capacidade de rotação suficiente nos pontos onde ocorram as rótulas plásticas, de forma a permitir que ocorram as necessárias redistribuições de momentos fletores. Além disso, esta análise apenas é permitida quando garantida a estabilidade dos elementos nos pontos onde se formam rótulas plásticas. Nesses pontos a secção transversal do elemento deverá possuir uma capacidade de rotação superior ou igual à necessária nesse local. Num elemento estrutural de secção constante a capacidade de rotação de uma rótula plástica pode ser considerada suficiente se forem satisfeitos os seguintes requisitos:

- O elemento em questão deverá ter secções transversais de classe 1 onde se formarem as rótulas plásticas;
- Na secção onde se formar a rótula plástica, se estiver aplicado um esforço transverso superior a 10% da sua resistência ao esforço transverso, deverão ser aplicados reforços na alma a uma distância não superior a h/2 da rótula plástica, sendo h a altura da secção transversal.

O EC3 prevê igualmente vários tipos de análises globais plásticas, corresponde a diferentes níveis de sofisticação na incorporação dos efeitos da não linearidade do material:

- Análise elasto-plástica (anteriormente referida) com a plastificação de secçõese/ou de ligações, definindo rótulas plásticas;
- Análise plástica não-linear considerando a plastificação parcial de elementos em zonas plásticas;
- Análise rígido-plástica desprezando o comportamento elástico entre as rótulas.

Em qualquer dos casos referidos, o EC3 permite a utilização da relação tensão-extensão bilinear da Fig. 4.2. [31]

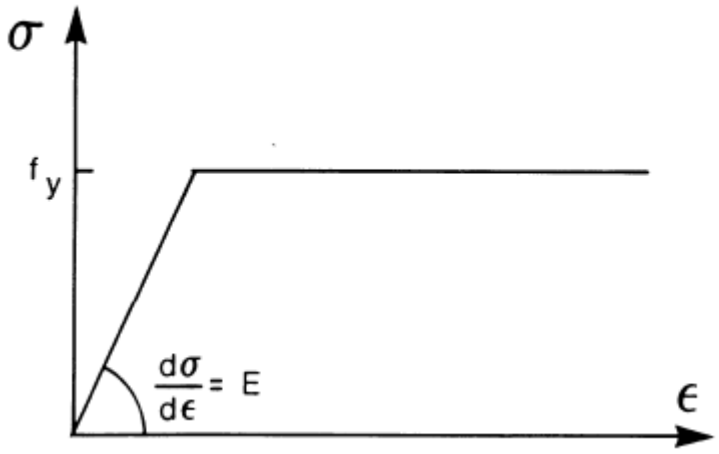


Fig. 4.2 - Relação tensão-extensão bilinear do aço

Como exposto anteriormente, a esbelteza dos componentes das secções metálicas pode condicionar a qualquer análise estrutural. Se por um lado, para o mesmo tipo de material, ao termos secções muito pouco esbeltas podemos tirar partido das propriedades de plasticidade da secção, por outro lado ao termos placas demasiado esbeltas (relação entre o seu comprimento e a sua largura), o regulamento pode colocar entraves quando em algum instante essa placa alvo de instabilidade atinge o seu limite elástico. O EC3 prevê então a divisão em quatro classes as secções transversais dos elementos, em que os critérios de separação das classes são baseados na resistência das secções (limite elástico ou plástico), e na sua capacidade de deformação à flexão. Este conceito de classe de secção transversal permite desde logo integrar a verificação da encurvadura local nas condições de flexão ou compressão de secções.

Os fatores que influenciam a classe de uma secção são os seguintes:

- O limite elástico do aço da secção;
- A geometria da secção, e em particular a esbelteza das suas placas parcial ou totalmente comprimidas;
- As condições de apoio das placas constituintes da secção;
- No caso de secções assimétricas, o sentido da flexão (positivo ou negativo);
- A posição das fibras neutras tanto elástica como plástica, que são influenciadas pela combinação do esforço axial com o momento fletor;

- A direção do eixo do momento fletor em casos de flexão desviada, que irá determinar a orientação da fibra neutra, e portanto a geometria e extensão das zonas comprimidas.

Torna-se então importante conhecer em detalhe as propriedades de cada uma das classes, para se perceber as vantagens e desvantagens do uso de secções de uma determinada classe, tal como é demonstrado na Fig. 4.3:

- As secções de classe 1 (plásticas) são aquelas em que se podem formar rótulas plásticas, com a capacidade de rotação necessária para uma análise plástica, sem a redução da sua resistência;
- Já às secções classificadas como de classe 2 (compactas), embora possam atingir a sua resistência plástica, por terem elementos demasiado esbeltos não podem sofrer o mesmo nível de deformação rotacional que as secções de classe 1, o que impossibilita a formação de rótulas plásticas nestas secções;
- Nas secções de classe 3 (semicompactas), as tensões devem ser calculadas com base numa distribuição elástica, e a fibra extrema comprimida deve apresentar um nível de tensão não superior à tensão de cedência do aço. Nestas secções o fenómeno da encurvadura local impede que o momento resistente plástico seja atingido;
- Por fim, as secções de classe 4 (esbeltas) são aquelas em que o fenómeno de encurvadura local ocorre antes de ser atingida a tensão de cedência do aço, em uma ou mais partes da secção transversal. Nestas secções, deve proceder-se à redução da secção transversal para fins de cálculo.

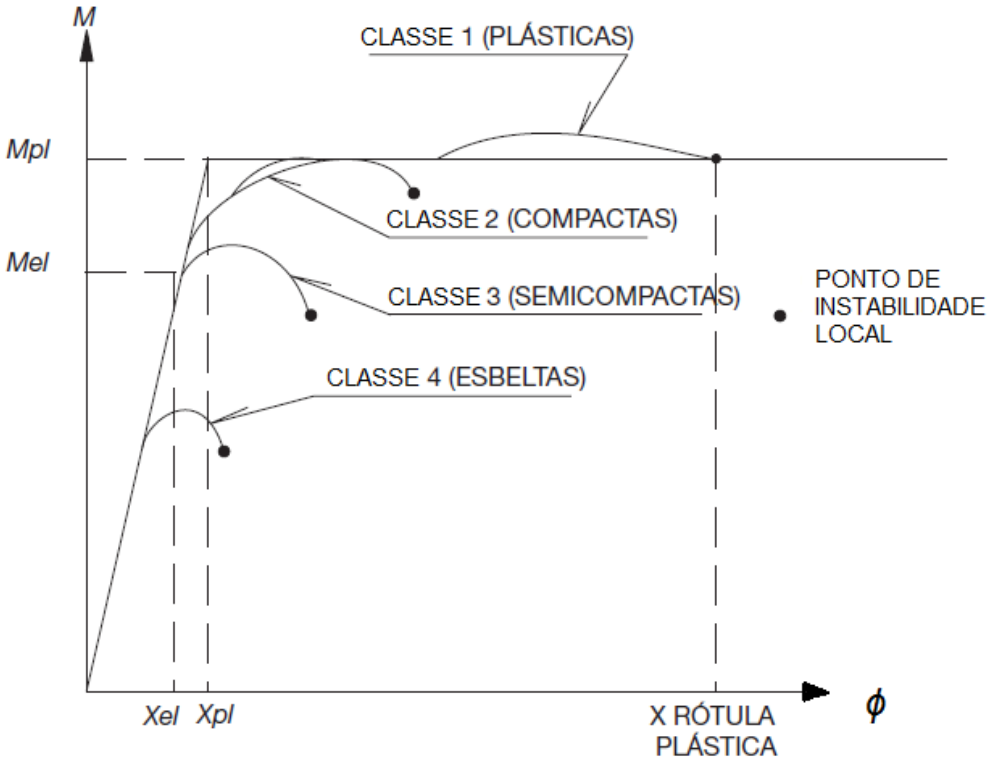


Fig. 4.3 – Gráfico demonstrativo da diferença na capacidade resistente e rotacional, de secções de diferentes classes [32] (adaptado)

Compreenda-se que numa determinada secção poderão existir diferentes placas cuja classificação é diferente entre si, dependendo da sua esbelteza e da extensão da zona comprimida. Por exemplo em determinado perfil em I, pode ter a sua alma com a classificação de classe 4, enquanto os seus banzos permanecem de classe 1. Este é aliás um exemplo muito recorrente em perfis comerciais da gama europeia IPE, nos casos de flexão em torno do eixo dos *yy*. Nestes casos a classe final da secção é definida como a máxima classe de todos os seus elementos, ou seja a classe menos favorável. No Anexo A pode visualizar-se o quadro do EC3 para a classificação de componentes internos (almas), e também o quadro para a classificação de elementos exteriores (banzos).

4.3. VERIFICAÇÕES DE SEGURANÇA DO EC3

Como podemos constatar até este ponto, a classificação das secções terá um papel decisivo na avaliação de parâmetros resistentes da secção. Este será sempre o primeiro passo a tomar, sempre que a verificação de segurança de uma determinada secção metálica (ou mista) seja requerida. Neste capítulo será tido em conta uma secção tipo como a representada na Fig. 4.4, para uma melhor interpretação das expressões.

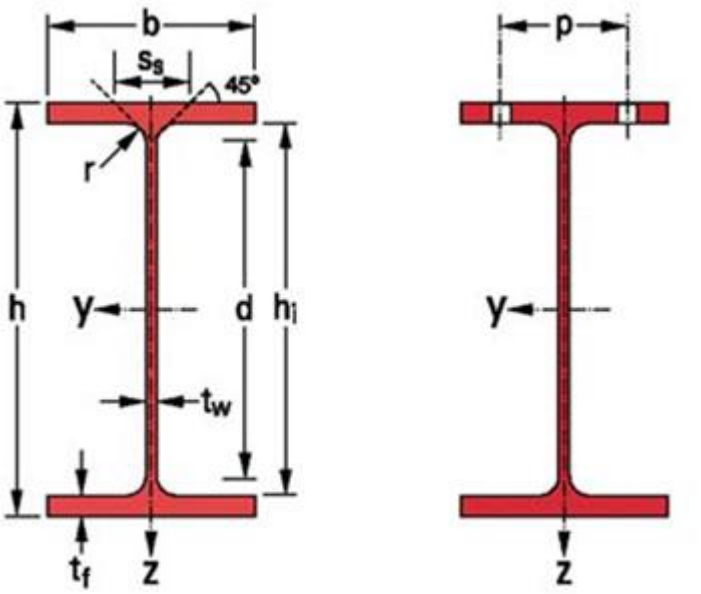


Fig. 4.4 - Secção Tipo – IPE

Para uma análise em regime elástico a teoria da resistência dos materiais prevê uma abordagem para a avaliação do esgotamento da resistência mecânica no ponto mais desfavorável através da expressão (2), para um estado de tensão plana:

$$\left(\frac{\sigma_{x,Ed}}{f_y/\gamma_{M0}}\right)^2 + \left(\frac{\sigma_{z,Ed}}{f_y/\gamma_{M0}}\right)^2 - \left(\frac{\sigma_{x,Ed}}{f_y/\gamma_{M0}}\right)^2 . \left(\frac{\sigma_{z,Ed}}{f_y/\gamma_{M0}}\right)^2 + 3\left(\frac{\tau_{Ed}}{f_y/\gamma_{M0}}\right)^2 \leq 1 \qquad (2)$$

Na expressão (2) os valores de $\sigma_{x,Ed}$ e de $\sigma_{z,Ed}$ correspondem aos valores da tensão normal nos eixos especificados e no ponto considerado, enquanto o valor de τ_{Ed} corresponde ao valor da tensão tangencial no ponto considerado.

Este critério não considera a plastificação na distribuição de tensões na secção transversal, logo trata-se de um critério conservador para muitas secções. A resistência das secções transversais de acordo com critérios plásticos deveria verificar-se através da obtenção de uma distribuição plástica de tensões que equilibre os esforços de solicitação sem que se supere o limite elástico. A dita distribuição deve ser compatível com a classificação da secção transversal que se analisa.

De modo expedito, aproximado e conservador pode aplicar-se a todas as secções das classes 1, 2 ou 3 a Expressão (3), que não é mais que a soma linear da contribuição do esforço axial e dos momentos fletores numa secção.

$$\frac{N_{Ed}}{N_{Rd}} + \frac{M_{y,Ed}}{M_{y,Rd}} + \frac{M_{z,Ed}}{M_{z,Rd}} \leq 1 \tag{3}$$

Enquanto os numeradores da expressão representam os esforços de cálculo atuantes na secção em questão, os denominadores correspondem aos valores resistentes de cálculo de cada uma das diferentes ações singulares, sem mais alguma ação concomitante. Como é de prever, estes valores resistentes dependem da sua classificação, que por sua vez, e como referido anteriormente, dependem dos esforços atuantes que influenciam as zonas comprimidas da secção. Por exemplo, ao analisar-se uma secção submetida a um esforço axial de compressão pode eventualmente concluir-se que nesse caso a sua classificação seria de classe 3, o que não acontecerá necessariamente no caso da flexão, podendo a secção estar classificada como de classe 1 ou 2.

Aquando desta análise, pode ocorrer que a classificação da secção, tanto sujeita a esforço axial de compressão, como submetida a momento fletor em qualquer dos eixos principais, seja de classe 4. Nesta situação a expressão anterior deixará de ser válida, já que a secção terá de ser reduzida devido ao efeito da instabilidade local, e consequentemente irá alterar os valores de cálculo resistentes.

4.3.1. ESFORÇO AXIAL DE TRAÇÃO

Quando uma determinada secção transversal é submetida a um esforço axial de tração, esta não irá apresentar qualquer problema de instabilidade local, dado que este fenómeno só tem lugar em regime de compressão. Concluindo, não faz sentido neste caso a classificação da secção transversal. O seu esforço axial resistente corresponde a um esforço resistente plástico, como se verifica na Expressão (4), de acordo com a cláusula 6.2.3(2) da parte 1-1 do EC3.

$$N_{pl,Rd} = \frac{Af_y}{\gamma_{M0}} \tag{4}$$

Onde *A* refere-se à área bruta da secção. O valor do esforço axial atuante não deve ultrapassar o valor do esforço resistente plástico, como indica a Expressão (5) da cláusula 6.2.3(1) da parte 1-1 do EC3.

$$\frac{N_{Ed}}{N_{pl,Rd}} \leq 1.0 \tag{5}$$

4.3.2. ESFORÇO AXIAL DE COMPRESSÃO

Já solicitações axiais de compressão em secções de perfis metálicos podem conduzir à necessidade de redução da área. Assim, é necessário a classificação da secção. A Expressão (6) permite obter o esforço resistente de compressão para secções de classe 1, 2 ou 3, segundo a cláusula 6.2.4(2) da parte 1-1 do EC3.

$$N_{c,Rd} = \frac{Af_y}{\gamma_{M0}} \tag{6}$$

Quando a secção á classificada como de classe 4, esta tem de ser calculada de acordo com a sua área efetiva A_{eff}, através da Expressão (7), da mesma cláusula. O cálculo da área efetiva está definido na parte 1-5 do EC3, onde é abordado em maior profundidade o tratamento do fenómeno de encurvadura local.

$$N_{c,Rd} = \frac{A_{eff}f_y}{\gamma_{M0}} \tag{7}$$

De modo semelhante à tração, pode ser verificada a segurança de uma secção metálica sujeita a um esforço de compressão de acordo com a Expressão (8), que corresponde à cláusula 6.2.4(2) da parte 1-1 do EC3.

$$\frac{N_{Ed}}{N_{c,Rd}} \leq 1.0 \tag{8}$$

4.3.3. MOMENTO FLETOR

Antes de se definir o momento fletor resistente de uma determinada secção, importa conhecer as diferenças existentes entre o cálculo seccional elástico e o plástico. Segundo a resistência dos materiais o valor do momento resistente é igual ao produto entre o módulo de flexão *W*, e tensão de cedência f_y da secção em análise, como se verifica na Expressão (9).

$$M_{Rd} = Wf_y \tag{9}$$

Porém este módulo de flexão pode dividir-se em dois tipos distintos: o módulo elástico W_{el}, e o plástico W_{pl}, com duas metodologias de cálculo igualmente distintas. Para calcular o módulo de flexão elástico, W_{el}, em torno de um determinado eixo, é necessária a inércia em torno desse mesmo eixo e dividir pela distância do centróide da secção à fibra mais afastada. Para se definir o módulo de flexão

plástico, W_{pl}, o processo é ligeiramente diferente. Neste caso o módulo plástico de flexão segundo o mesmo eixo corresponde ao momento estático em relação ao eixo que divide a secção em duas de igual área. Pode eventualmente ocorrer que a secção tenha uma esbelteza não compatível com as classificações 1, 2 ou 3, sendo que nesta situação deve-se calcular o módulo de flexão W_{eff}, com a secção reduzida das zonas comprimidas devido à flexão.

Pode então definir-se o momento resistente de cálculo para secções de classe 1 ou 2 segundo a expressão (10), de acordo com a cláusula 6.2.5(2) da parte 1-1 do EC3.

$$M_{c,Rd} = \frac{W_{pl} f_y}{\gamma_{M0}} \tag{10}$$

De modo semelhante, para secções de classe 3, o momento resistente é dado pela Expressão (11), da mesma cláusula do EC3.

$$M_{c,Rd} = \frac{W_{el} f_y}{\gamma_{M0}} \tag{11}$$

Para o caso das secções de classe 4, é utilizada a Expressão (12), ainda proveniente da mesma cláusula do EC3.

$$M_{c,Rd} = \frac{W_{eff} f_y}{\gamma_{M0}} \tag{12}$$

De modo análogo aos esforços axiais de tração e compressão, o EC3 define que deve ser satisfeita a condição presente na Expressão (13), definida na cláusula 6.2.5(1) da parte 1-1 do EC3.

$$\frac{M_{Ed}}{M_{c,Rd}} \leq 1.0 \tag{13}$$

4.3.4. ESFORÇO TRANSVERSO

A verificação da resistência ao esforço transverso deverá ser feita com recurso à comparação do esforço transverso atuante com o esforço transverso plástico resistente que é dado pela Expressão (14), presente na cláusula 6.2.6(2) da parte 1-1 do EC3.

$$V_{pl,Rd} = \frac{A_v \left(f_y / \sqrt{3} \right)}{\gamma_{M0}} \tag{14}$$

Onde A_v representa a área de corte que para os perfis comerciais da gama HE e IPE, que no eixo dos *yy* é dada pela Expressão (15), e no eixo dos *zz* é dada pela Expressão (16), ambas presentes na cláusula 6.2.6(3) da parte 1-1 do EC3.

$$A_v = 2bt_f + (t_w + 2r)tf \leq 1.2h_w t_w \tag{15}$$

$$A_v = A - \sum h_w t_w \tag{16}$$

Em que b e t_f correspondem à largura e espessura do banzo respetivamente, r corresponde ao raio das zonas curvas da ligação da alma aos banzos, e A corresponde a área bruta da secção. Em perfis soldados em I, o parâmetro A_v é dado pela Expressão (17) no eixo dos *yy*, e pela Expressão (18) no eixo dos *zz*, tal como está previsto na mesma cláusula.

$$A_v = 1.2h_w t_w \tag{17}$$

$$A_v = A - \sum h_w t_w \tag{18}$$

A cláusula 6.2.6(1) da parte 1-1 do EC3 indica que o valor de esforço transverso atuante na secção, deve satisfazer a condição presente na Expressão (19).

$$\frac{V_{Ed}}{V_{c,Rd}} \leq 1.0 \tag{19}$$

4.3.5. INTERAÇÃO DE ESFORÇOS

Os esforços anteriormente definidos quando aplicados às secções de um modo simultâneo provocam efeitos ainda mais desfavoráveis às ditas secções. Já foi referido neste documento que a posição do eixo neutro pode influenciar a classificação da secção, já que a zona comprimida da secção depende dessa posição. Em situações de flexão composta verifica-se portanto que a classificação é em todos os aspetos mais complexa que em situações de flexão simples ou esforço axial. Neste subcapítulo apenas será referida a abordagem regulamentar presente no EC3. No próximo subcapítulo far-se-á uma explicação sobre o método expedito implementado no *web service* para fazer a classificação das secções, e o cálculo dos momentos resistentes que darão origem à curva de interação de esforço axial/momento fletor.

Para secções transversais de perfis em I das classes 1 e 2, para uma situação de flexão composta em torno do eixo dos *yy*, o EC3 estabelece o limite para o momento atuante, representado na Expressão (20), que provém da cláusula 6.2.9.1(5).

$$M_{N,y,Rd} = M_{c,y,Rd}\frac{1-n}{1-0.5a} \leq M_{c,y,Rd} \tag{20}$$

Onde os valores dos coeficientes *n* e *a* são fornecidos pelas Expressões (21) e (22) respetivamente, obtidas da mesma cláusula do EC3.

$$n = \frac{N_{Ed}}{N_{pl,Rd}} \tag{21}$$

$$a = \frac{A - 2bt_f}{A} \leq 0.5 \tag{22}$$

Já para flexão em torno do eixo perpendicular ao anterior, o eixo dos *zz*, o cálculo do momento resistente é feito de modo similar. Para $n \leq a$ utiliza-se Expressão (23) para encontrar o momento, enquanto nos restantes casos é utilizada a Expressão (24). Ambas as Expressões estão indicadas na cláusula 6.2.9.1(4). Os valores de *n* e *a* podem ser novamente obtidos através das Expressões (21) e (22).

$$M_{N,z,Rd} \leq M_{c,z,Rd} \tag{23}$$

$$M_{N,z,Rd} = M_{c,z,Rd}\left[1 - \left(\frac{n-a}{1-a}\right)^2\right] \tag{24}$$

Em ambos os casos de flexão, em torno do eixo dos *yy* ou em torno do eixo dos *zz*, a parte 1-1 do EC3 refere que, se os esforços respeitarem a Expressão (25) da cláusula 6.2.9.1(2) da norma, será garantida a segurança da secção.

$$M_{Ed} \leq M_{N,Rd} \tag{25}$$

As Expressões anteriores tratam-se de simplificações estabelecidas pela norma, que correspondem a resultados aproximados. No próximo subcapítulo veremos que estas Expressões podem ser por vezes conservativas, outras vezes pelo contrário estão associados a níveis de tensões que ultrapassam os valores estipulados pelo cálculo exato. Neste trabalho foi desenvolvida uma metodologia que permite saber com maior exatidão os valores dos momentos resistentes das secções em ambos os eixos, que será abordada no seguinte subcapítulo dedicado às curvas de interação de esforço axial/momento fletor.

Faltam definir os momentos resistentes em que as secções estão classificadas como de classes 3 ou classe 4. Em secções de classe 3, terá de se verificar a condição presente na Expressão (26), presente na cláusula 6.2.9.2(1).

$$\sigma_{x,Ed} \leq \frac{f_y}{\gamma_{M0}} \tag{26}$$

Em que $\sigma_{x,Ed}$, corresponde ao valor de cálculo da tensão longitudinal local atuante devida ao momento fletor e ao esforço normal. Na prática para verificações de segurança é utilizada a Expressão (3) que corresponde a uma simplificação.

Em secções de classe 4 terá igualmente de ser satisfeita a condição presente na expressão (26), presente na cláusula 6.2.9.3(1). Contudo a expressão (3) deixa de ter validade, isto porque para alem da redução da área da secção, esta mesma redução pode provocar uma variação do centro de gravidade da secção que pode gerar um momento adicional, visto que o esforço axial deixa de estar aplicado no centro de gravidade da secção. Para a segurança ser verificada em secções de classe 4 é então necessário que seja cumprido o critério da Expressão (27), da cláusula 6.2.9.3(2).

$$\frac{N_{Ed}}{A_{eff} f_y/\gamma_{M0}} + \frac{M_{y,Ed} + N_{Ed} e_{Ny}}{W_{eff,y} f_y/\gamma_{M0}} + \frac{M_{z,Ed} + N_{Ed} e_{Nz}}{W_{eff,z} f_y/\gamma_{M0}} \leq 1 \tag{27}$$

Nesta fórmula, os valores dos módulos de flexão efetivos $W_{eff,y}$ e $W_{eff,z}$, terão se ser calculados no respetivo eixo, numa situação de flexão simples. De modo similar o valor da área efetiva A_{eff}, é obtido numa situação de compressão simples. Este cálculo estabelece também o uso das excentricidades e_{Ny} e e_{Nz}, que correspondem à diferença entre as posições do eixo neutro elástico, antes e depois da redução da área efetiva em compressão simples A_{eff}.

4.4. CURVAS DE INTERAÇÃO

No cálculo de secções em betão armado é comum o dimensionamento com recurso a curvas de interação de esforço axial com momento fletor [33]. O mesmo não sucede no dimensionamento de peças metálicas, apesar de estas na maioria das vezes serem compostas por perfis comerciais, perfeitamente tabelados e normalizados, mas para os quais não se encontra uma solução para o dimensionamento em flexão composta que não sejam as fórmulas expostas no subcapítulo anterior. Este facto deve-se sobretudo à complexidade do algoritmo que possibilita a obtenção destas curvas de interação. Neste subcapítulo far-se-á uma abordagem ao desenvolvimento deste tipo de curvas, que representam uma inovação no campo das estruturas metálicas.

Independentemente da classificação podem fazer-se duas análises distintas à secção em situação de flexão composta: uma análise elástica, em que não é ultrapassado em nenhum ponto da secção a sua tensão de cedência, ou em alternativa e tirando partindo da teoria da plasticidade, uma análise em que todos os pontos da secção se apresentam com uma tensão igual à tensão de cedência.

O problema pode então colocar-se do seguinte modo: para um dado esforço axial atuante determinar qual o momento resistente elástico e plástico, sendo que o plástico será sempre superior ao elástico. Note-se que terá sempre de ser cumprida a premissa que o dado esforço axial atuante nunca seja superior ao esforço axial resistente anteriormente definido (tanto em compressão como em tração).

4.4.1. CURVA DE INTERAÇÃO ELÁSTICA

A curva de interação elástica num determinado eixo é obtida com base na teoria da resistência dos materiais, através da conhecida Expressão (28).

$$\frac{f_y}{\gamma_{M0}} \geq \frac{N_{Ed}}{A} + \frac{M_{Ed}}{W_{el}} \tag{28}$$

Para um determinado esforço axial atuante, existe um momento resistente elástico limite, em que num determinado ponto da secção é atingida a tensão de cedência. A área da secção pode ter de ser alterada para uma área efetiva se a secção for de classe 4 em compressão pura, e de modo semelhante para o módulo de flexão, onde também poderá existir a possibilidade da sua redução para um módulo efetivo, se a secção estiver de classificado como de classe 4 em flexão simples em torno do eixo em questão. Estes aspetos são estabelecidos na Expressão (29).

$$\frac{f_y}{\gamma_{M0}} \geq \frac{N_{Ed}}{A_{eff}} + \frac{M_{Ed} + N_{Ed} e_N}{W_{eff}} \tag{29}$$

Em que os valores de A_{eff}, W_{eff} e e_N são os mesmo que os estabelecidos na Expressão (27). O limite das regiões definidas nas expressões anteriores forma a denominada curva de interação elástica representada na Fig. 4.5.

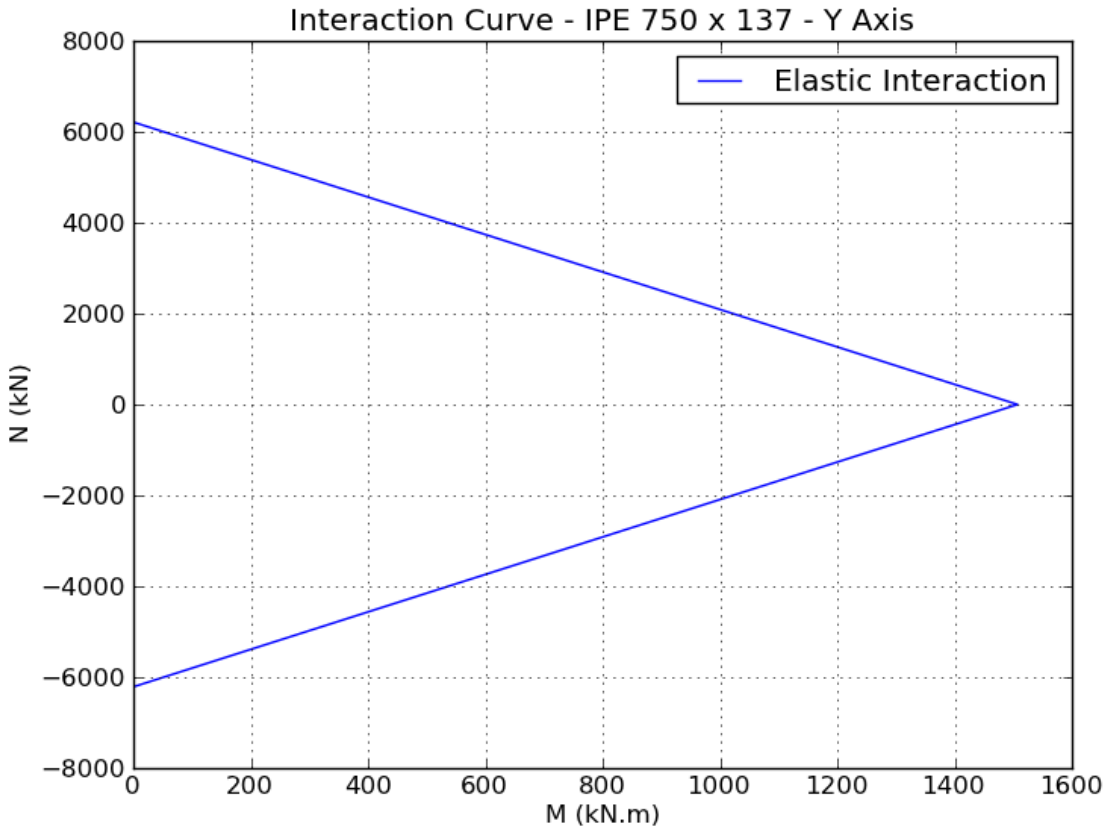


Fig. 4.5 - Curva de interação elástica para o perfil IPE 750 x 137, para flexão em torno do eixo dos *yy*, obtida através do *Matplotlib*

4.4.2. CURVA DE INTERAÇÃO PLÁSTICA

O processo de obtenção da curva de interação plástica é contudo um pouco mais elaborado. Neste algoritmo, o primeiro passo a tomar é o cálculo do eixo neutro plástico, que se obtém através do esforço axial atuante, que é absorvido por uma parte da secção transversal, deixando a parte restante para resistir ao momento fletor, como de resto está representado na Fig. 4.6.

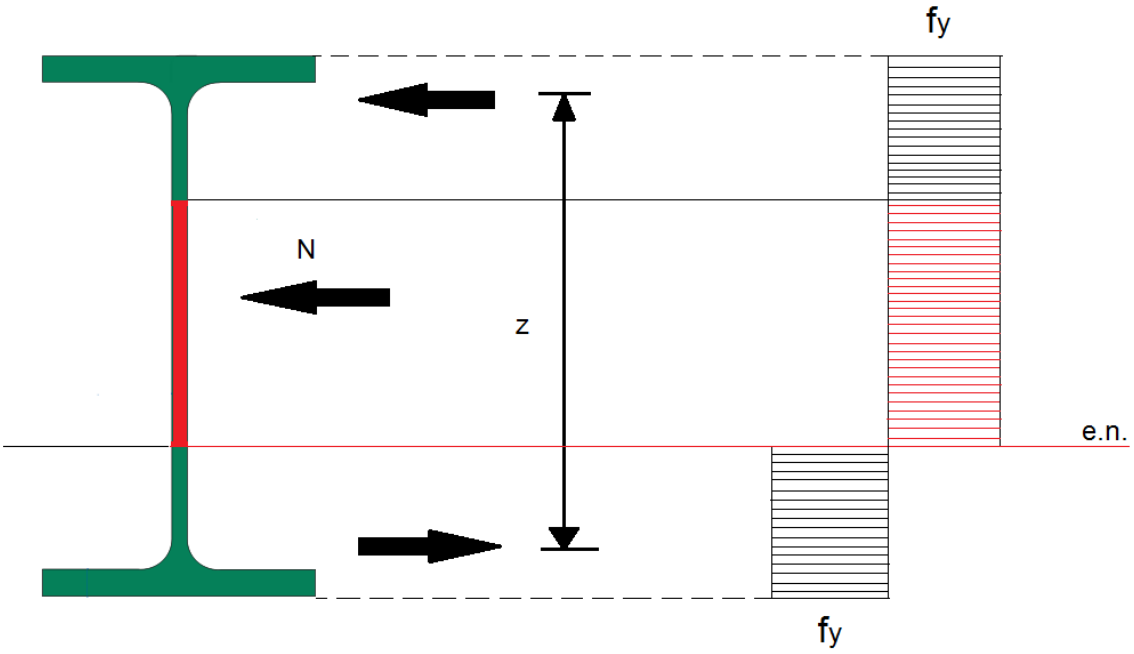


Fig. 4.6 - Representação de um perfil IPE sujeito a um esforço axial, e o seu momento resistente

Dependendo da natureza do esforço axial (tração ou compressão) e do sentido do momento fletor, a posição do eixo neutro será um dos bordos da zona alocada a esse esforço axial (a vermelho na Fig. 4.6). O momento fletor resistente é obtido através da multiplicação das forças geradas pelas zonas adjacentes à zona anteriormente referida, pelo seu respetivo braço (*z* na Fig. 4.6). É neste ponto, que o cálculo de uma secção comercial IPE ou HE, mais se afasta do cálculo de uma secção duplamente simétrica soldada, isto devido aos bordos arredondados presentes nos perfis comerciais, que são de difícil integração no código.

Nos casos em que o eixo neutro plástico se sobreponha nas zonas onde se situam as ligações da alma aos banzos, pode classificar-se a alma e os banzos em situação de compressão simples. Como se pode imaginar, o cálculo do momento resistente nestes casos torna-se muito elaborado, pois é necessário saber com exatidão a área da secção disponível para o momento fletor, e também a posição do seu centro de gravidade para assim se obter o braço (*z*) correspondente. Para os pontos da curva de interação correspondentes a esta situação, neste trabalho optou-se por não se fazer a avaliação do momento resistente nestes pontos, usando-se em alternativa uma aproximação linear bastante próxima da realidade. Um exemplo desta curva está representado na Fig. 4.7

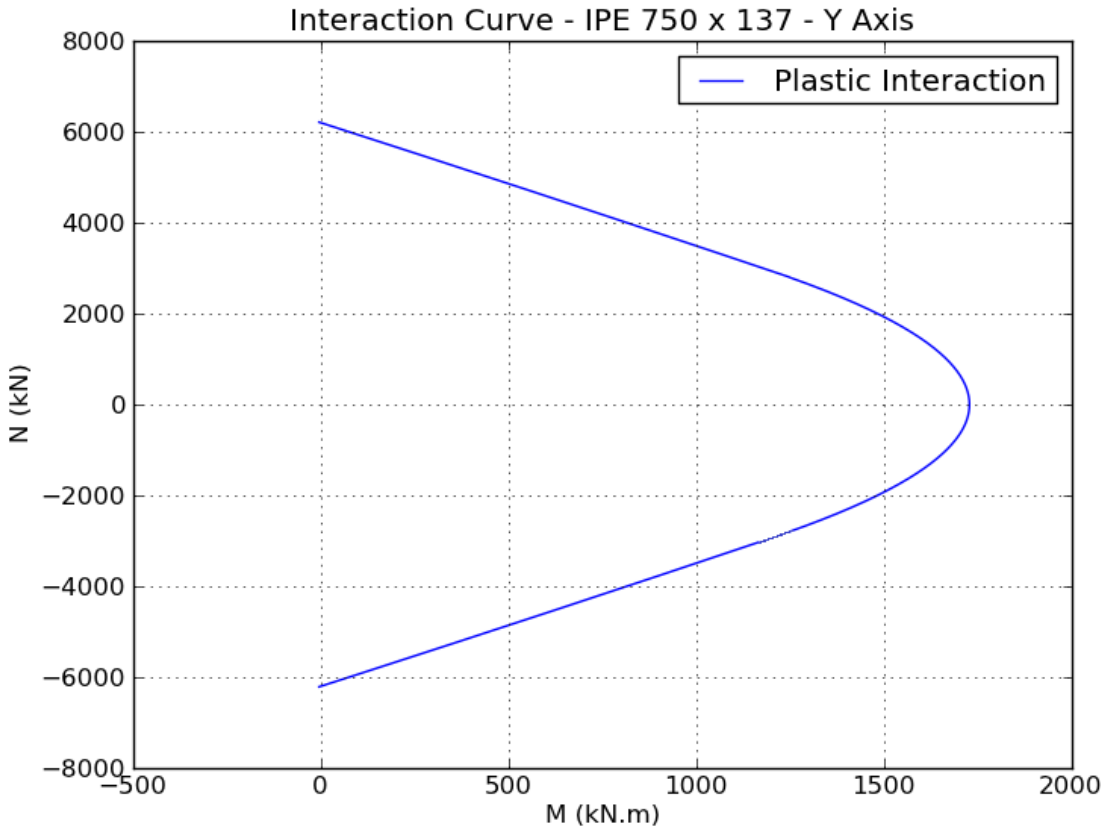


Fig. 4.7 - Curva de interação plástica para o perfil IPE 750 x 137, para flexão em torno do eixo dos *yy*, obtida através do *Matplotlib*

4.4.3. CURVA COM BASE NOS CRITÉRIOS DE CLASSIFICAÇÃO DO EC3

Ao contrário da primeira impressão que possa ser passada, estas curvas de interação não se tratam de funções contínuas definidas matematicamente, mas de um conjunto de pontos definidos num *array Numpy*, que ao ser interpretado pelo *Matplotlib* gera gráficos de bastante qualidade. Para a obtenção destes pontos foi utilizada a seguinte metodologia:

1. A uma dada secção transversal podem ser aplicados, uma gama infinita de esforços axiais compreendida entre a sua resistência à tração $N_{t,Rd}$ e a sua resistência à compressão $N_{c,Rd}$, isto claro, se não existirem para já momentos aplicados. O que o algoritmo do *web service* faz não é mais que dividir este intervalo num determinado número de pontos, em que cada um destes pontos corresponde a um esforço axial a ser aplicado a secção. A curva terá mais qualidade quantos mais pontos forem considerados, ou seja menor for o intervalo entre cada ponto. Através da sucessiva análise de curvas com a consideração de diferentes números de pontos, verificou-se que secções onde foram considerados 1000 pontos já produzem resultados de qualidade;
2. Para cada um destes valores de esforço axial, serão aplicados as metodologias referidas nos dois anteriores subcapítulos para o cálculo dos momentos resistentes elástico e plástico;

3. Paralelamente ao ponto anterior é necessário também conhecer a posição das fibras neutras elástica e plástica;
4. Com os dados do ponto transato, podemos fazer uma classificação da secção para esforço axial em questão, e construir os *arrays Numpy* necessários com os valores dos momentos resistentes;
5. No final do varrimento de todos os pontos de esforço axial, é necessário para os perfis comerciais com bordos arredondados fazer a aproximação linear;
6. Por fim basta a representação dos pontos através do *Matplotlib*, tal como exemplificado na Fig. 4.8.

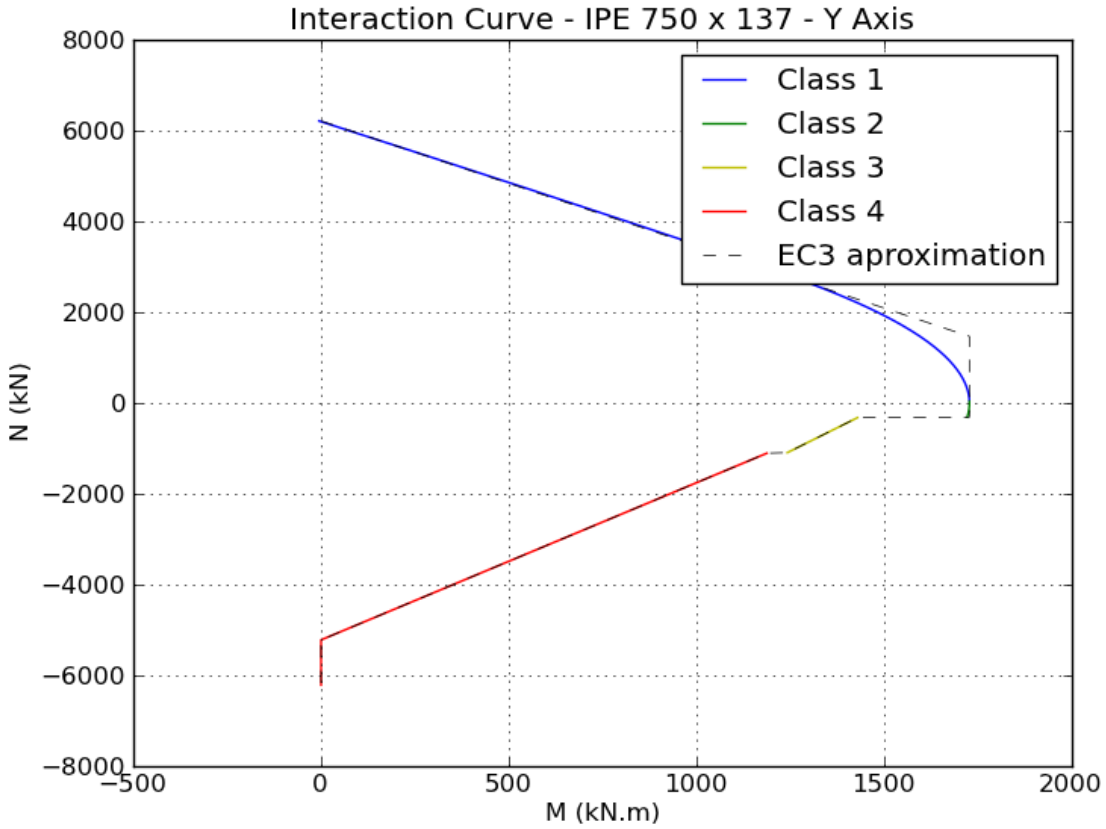


Fig. 4.8 - Representação da curva de interação, de acordo com as normas de classificação do EC3

Através da representação de curvas de interação como na Fig. 4.8 é possível uma verificação de segurança de acordo com o EC3, de uma forma expedita e sem recurso às expressões referidas no subcapítulo 4.3. Com se verifica, pode visualizar-se as diferentes classes, através de diferentes cores e tipos de linha. Através destas curvas pode considerar-se que uma secção está em segurança, se para um par esforço axial/momento fletor, se o seu ponto correspondente estiver representado no interior da curva.

Além da verificação de segurança, também o dimensionamento de secções de perfis metálicos comerciais é possível. À imagem dos ábacos presentes no trabalho de Barros e Figueiras [33], é também possível a geração de curvas de interação de uma determinada gama de perfis metálicos para

assim se proceder ao dimensionamento da secção, através da escolha da respetiva curva, como está representado na Fig. 4.9 - Curvas de interação de alguns perfis da gama IPE Nesta figura representou-se as curvas de interação correspondentes aos perfis da gama europeia IPE: IPE100, IPE160, IPE200, IPE220, IPE240, IPE270, IPE300, IPE330, IPE360, IPE400, IPE450, IPE500, IPE550 e IPE600, estando igualmente as curvas na Fig. 4.9 por esta ordem, do interior para o exterior da figura. De modo similar à verificação de segurança, a representação de um par de esforços na figura através de um ponto, pode levar à escolha da curva correspondente ao perfil que seja mais económico, e que verifique simultaneamente a segurança da secção.

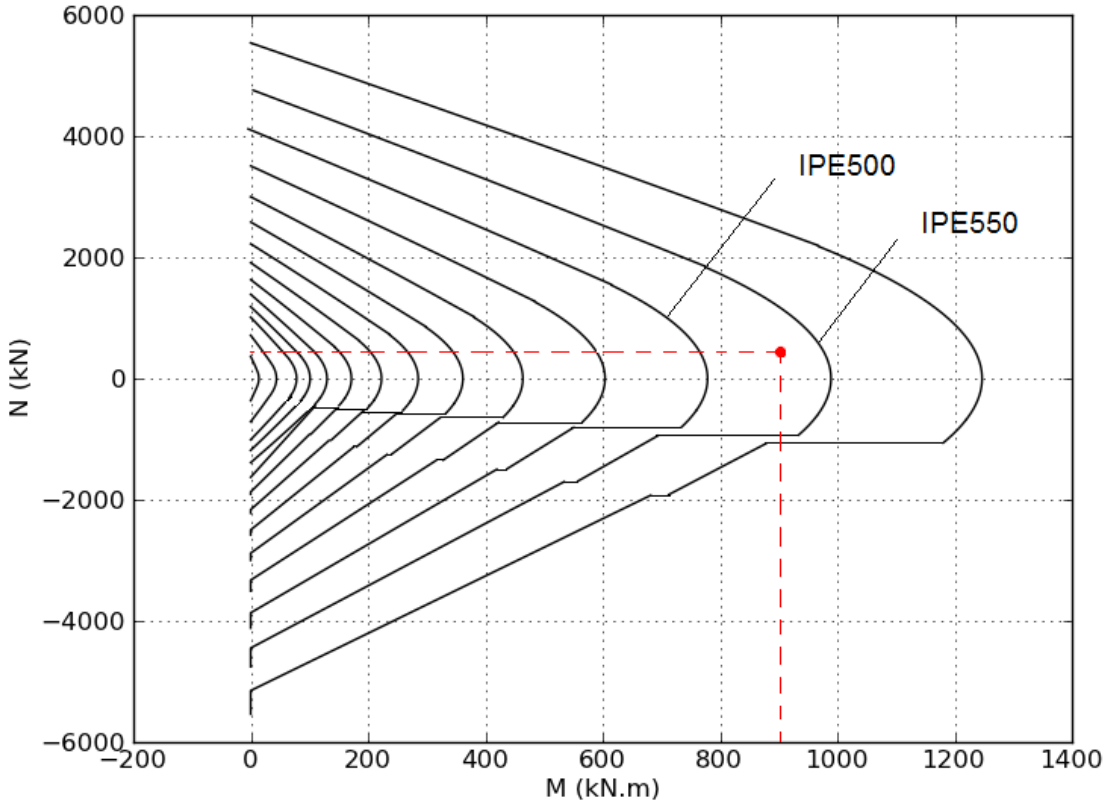


Fig. 4.9 - Curvas de interação de alguns perfis da gama IPE

4.4.4. CONFRONTAÇÃO COM AS CURVAS OBTIDAS DAS FÓRMULAS DO EC3

Aquando da aplicação das fórmulas do EC3 que, devido à complexidade da interação plástica, fazem uma aproximação à realidade, foi verificado com recurso aos gráficos do *Matplotlib* que esta aproximação nem sempre é conservativa. Como se pode verificar na Fig. 4.10, que representa um caso de um perfil em particular, com recurso a esta tecnologia pode concluir-se que existem diferenças algo significativas em zonas da curva de interação onde os momentos estão próximos do máximo e existe pouco esforço axial. Esta diferença é visível em todos os perfis das gamas comerciais abordadas neste trabalho e também nos perfis soldados, sendo que esta diferença é significativa em flexão composta em torno do eixo dos *yy*. Esta é mais uma prova que o cálculo destas secções com recurso a gráficos gerados por estas ferramentas pode ser uma vantagem no dimensionamento estrutural.

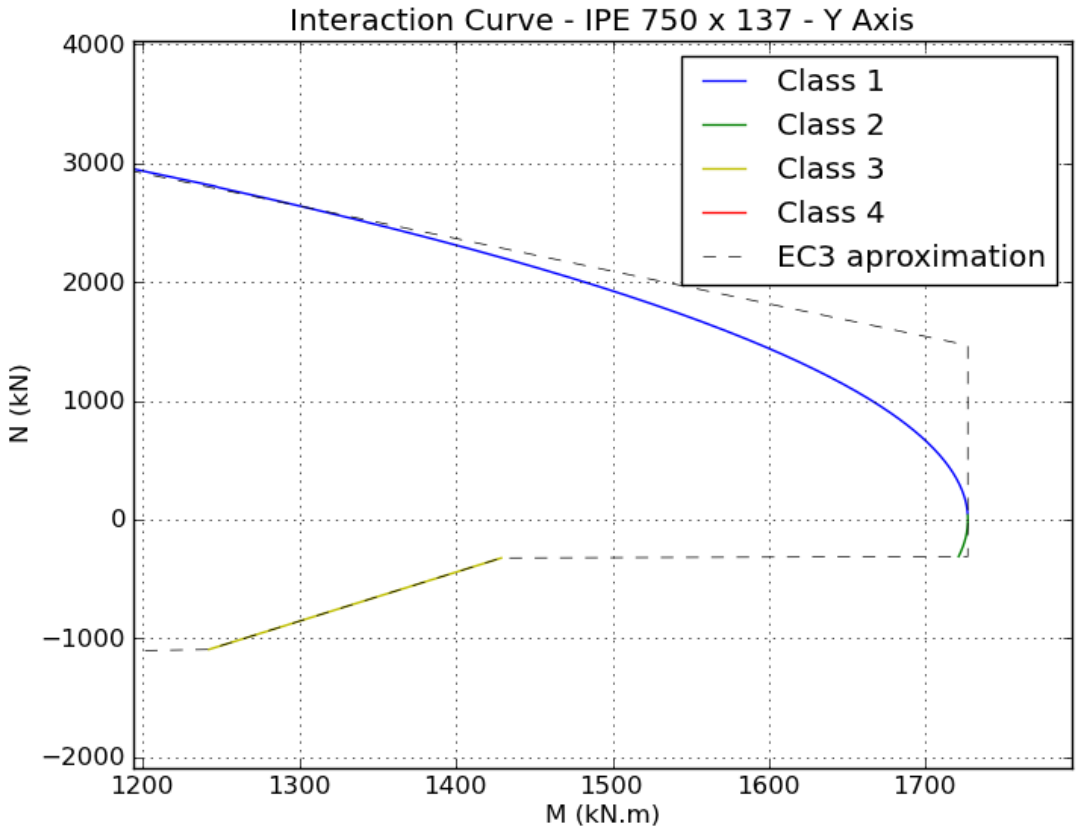


Fig. 4.10 - Detalhe da curva de interação referente à zona onde o EC3 não é conservativo na sua simplificação

4.5. CONSIDERAÇÕES FINAIS

Neste capítulo foi abordada a metodologia presente no EC3 para as verificações de segurança do EC3. Foram ainda apresentadas alternativas a esta norma para a verificação de segurança através do recurso a representações gráficas, possíveis através da ferramenta do *Python* o *Matplotlib*. No próximo capítulo será descrito em maior detalhe como foi feita a implementação do *web service*, e como pode ser utilizado.

5
IMPLEMENTAÇÃO DO WEB SERVICE

5.1. INTRODUÇÃO

Neste capítulo será exposto todo o trabalho prático produzido. Será explicada em detalhe a forma como foram implementadas as tecnologias referidas nos capítulos anteriores e todo o processamento dos dados pela API.

5.2. ABORDAGEM LÓGICA

Durante o desenvolvimento da API inúmeras questões foram-se levantando. Primeiro de tudo foi, como já foi referido, que linguagem de programação utilizar, mais tarde que a que *framework* recorrer, e finalmente como seriam efetuados os pedidos HTTP. Estes foram os três grandes campos de trabalho da API, onde apenas o primeiro se relaciona com a engenharia de estruturas pois é nele onde está contido o *business logic* disponibilizado pela API. Este subcapítulo ramifica-se então em três abordagens distintas que se completam:

- O *model* da arquitetura MVC, onde estão contidos todos os cálculos efetuados ao nível da engenharia de estruturas;
- A estrutura de dados dentro da *framework Flask*, onde será explicada como foi feita a articulação dos *controllers*, com o *view* e com o *model*;
- A forma como a informação é pedida, e como é disponibilizada, sendo que será dado mais enfoque à parte do *view* do modelo MVC, e na metodologia REST utilizada.

5.2.1. *BUSINESS LOGIC*

A grande base deste trabalho que refere-se ao cálculo de estruturas metálicas e está contida apenas num *package* denominado *steelsections*. Este *package* está isolado do resto da API, sendo apenas uma fonte de informação. Esta independência tem a ver com o facto de se ter procurado que o código pudesse ser utilizado em outro contexto que não o do *web service*, e até mesmo num princípio de expansibilidade, isto porque se esta parte do código estivesse de algum modo ligada por exemplo à parte dos pedidos HTTP, ou à parte da visualização dos resultados, seria de mais difícil manuseamento quando eventualmente se quisesse adicionar funções. Dentro deste espectro de funcionalidades que podem ser adicionadas à ferramenta, estão por exemplo o uso dos objetos para o cálculo de bases de ligações, vigas mistas, entre outros.

Neste trabalho foram estudados, como já referido, dois tipos de perfis metálicos: os perfis comerciais laminados a quente e os perfis soldados. Dentro destas duas gamas, existem funções especificamente criadas para cada uma e outras partilhadas por ambas. As funções específicas de cada gama são métodos da classe corresponde à sua gama, enquanto as funções partilhadas estão contidas em módulos específicos que são importados por essas classes quando necessário. Importa todavia dissecar o conteúdo de cada uma das classes de secções referentes a cada um dos tipos, especificando o seu funcionamento, e as suas funcionalidades. Para uma mais fácil compreensão desta parte do documento, recomenda-se a consulta em paralelo do Anexo B.

No caso dos perfis comerciais, dada à sua estandardização, as suas características podem ser armazenadas numa base dados que não tem de ser repetidamente reformulada, já que essas características são imutáveis e estão perfeitamente tabeladas. Como referido, a plataforma que foi escolhida para armazenar esta informação foram as bases de dados *SQLite* que são de fácil implementação em código *Python*, através de uma configuração de conexão a um ficheiro existente paralelamente ao código e presente no *package steelsections.* Porém, para os perfis soldados, o algoritmo tem de gerar todas essas características de raiz, dado que existem infinitas possibilidades de configurações geométricas.

A informação disponibilizada pela API pode dividir-se em dois ramos, em que o segundo depende do primeiro:

- A informação básica relativa aos perfis metálicos (compreenda-se área, inércias, módulos de flexão, raios de giração), que é devolvida pela base de dados no caso dos perfis comerciais, e que no caso dos perfis soldados tem de ser calculada;
- As curvas de interação de esforço axial e momento fletor e as verificações de segurança descritas no anterior capítulo, sendo que as verificações têm a sua base nas curvas de interação. Neste caso, em ambas as gamas de perfis as curvas e as verificações são calculadas de raiz.

A melhor forma de se entender este processo de geração de informação é começar pela geração dos objetos. Não existem apenas classes de secções no algoritmo. Existem outros elementos onde a aplicação da OOP fez todo o sentido, e que alimentam os objetos da classe secção. A classe transversal aos dois tipos de secção é a classe *"Material"*, já que todas as secções são compostas por um material. Cada objeto desta classe necessita dos valores de f_y, e da massa volúmica para serem calculadas as características das secções.

Além disso, como os perfis soldados podem ser divididos em partes mais pequenas, optou-se por dividir o perfil em banzos e alma, sendo estes elementos placas independentes entre si. Essas placas são também objetos de uma classe denominada *"Plate"*. Já estes objetos têm como parâmetros de entrada os valores das suas dimensões (a largura e a altura), e também um objeto da anterior classe material (o *Python* permite a inserção de objetos "dentro" de outros).

Também uma classe denominada *"Actions"* foi criada para ser integrada nas verificações de segurança. Os parâmetros de entrada são o esforço axial e os momentos e esforços transversos nos dois eixos principais.

Como já se referiu anteriormente, há diferenças significativas entre as duas classes de perfis metálicos, que se denominam por *"Commercial"*, para os perfis laminados a quente, e *"I2sym"*, pois tratam-se de perfis soldados em I duplamente simétricos. Essas dissemelhanças são logo evidentes à partida aquando da criação dos objetos, pois a classe *Commercial* apenas tem como parâmetros de entrada uma *string* com a especificação da secção para a busca na base de dados e um objeto da classe

Material, enquanto a classe *I2sym*, tem como parâmetros de entrada dois objetos da classe *Plate* (referentes aos banzos e à alma), e de igual modo um objeto da classe *Material*. Cada vez que feito um pedido à API é gerado um destes objetos conforme o solicitado, independentemente da informação requerida, e é gerada toda a informação relativa a cada secção. Como se pode verificar no esquema apresentado na Fig. 5.1, existem funções específicas para cada classe, e outras partilhadas entre ambas.

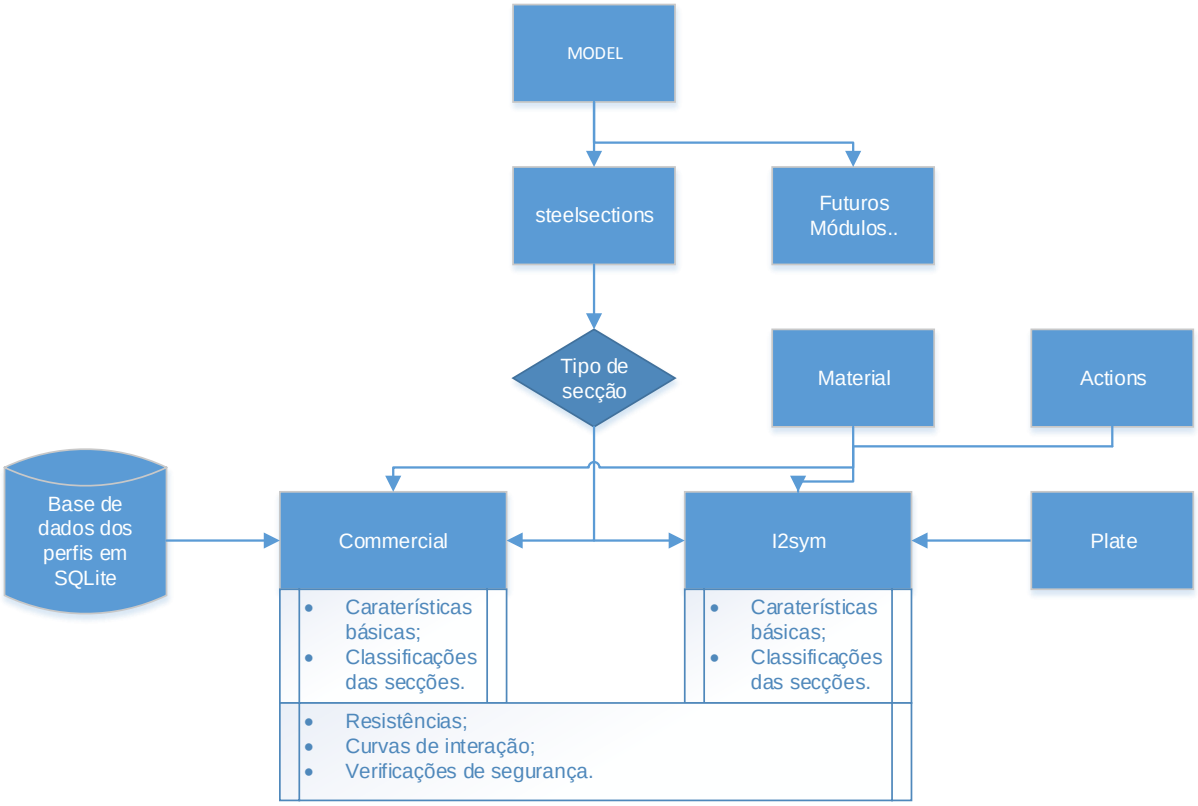


Fig. 5.1 - Esquematização interna do *model* do modelo MVC

5.2.2. MVC

Antes de ser especificada a forma dos URLs dos pedidos HTTP, far-se-á uma descrição sobre a arquitetura interna da API, e do fluxo de informação, dando especial atenção ao modelo MVC. Este modelo, como já referido no Capítulo 3 baseia-se em três principais agentes o *model* o *view* e o *controller*. Enquanto no anterior ponto se abordou o conteúdo do *model*, neste será dada atenção aos *controllers* e à sua articulação com o *model* e o *view*.

Quando é feito um pedido HTTP ao servidor, especificamente ao domínio *openg.fe.up.pt/api*, este é recebido na porta 80 do servidor, que contém um *software* chamado *Apache*, que se encarrega de ler esse pedido e de o reencaminhar, recorrendo ao módulo WSGI do *Apache*, para um módulo *Python* denominado *main.py*, no qual se gera a criação de um objeto da classe da *web framework Flask*. É neste módulo que se inicia todo o processo de execução do *web service*.

A Fig. 5.2 mostra uma aplicação minimalista na qual é exemplificada a utilização da *web framework Flask*, onde se pode verificar a iniciação da aplicação. Podemos verificar que neste módulo base da API que faz a receção dos pedidos HTTP, é criado um objeto *app* da classe *Flask*, sendo que é este objeto responsável pela leitura dos pedidos. Essa leitura é feita através do método desse objeto chamado *route*, que tem como parâmetro de entrada uma *string* com a especificação da parte do URL

existente depois do domínio. A resposta ao pedido é dada por uma função seguinte neste caso *hello()* [28].

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Fig. 5.2 - Exemplo de uma aplicação minimalista através da *framework Flask* [28]

Especificamente no caso do *web service* produzido neste trabalho há inúmeras vantagens em serem colocadas variáveis no próprio URL. Na Fig. 5.3 pode identificar-se a forma preconizada pelo *Flask* para inserir estas variáveis no código. Estas variáveis servirão para o utilizador especificar a informação pretendida, como será abordado no subcapítulo seguinte. Por agora interessa especificar como foi feita a divisão entre os pedidos de perfis comerciais e de perfis soldados. Isso foi feito através de *Blueprints* (em português "diagrama"), que como o nome indica serve de ajuda ao mapeamento em grandes APIs, pois possibilita a memorização de partes do URL e assim separar o código em função do pedido do utilizador [28]. No Anexo B.1 podemos ver esta metodologia concretizada, ao serem registados dois prefixos de URL, um para cada tipo de secção: *"sections/commercial"* e *"sections/builtup"*.

```
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post %d' % post_id
```

Fig. 5.3 - Uso de variáveis nos URLs. [28]

O módulo *main.py* encarrega-se portanto de receber o pedido, e de encaminha-lo para o respetivo *Blueprint* dependendo tipo de secção metálica. Dentro de cada *Blueprint,* ainda no módulo *main.py*, é executado o respetivo *controller* que contém duas funções: a primeira encarrega-se da criação de objetos das respetivas classes (*Commercial, I2sym*, *Material*, *Plate, Actions*), e a segunda trata de preencher o *template* produzido no módulo *view.py* e de converter essa informação para o formato JSON ou XML. Depois da execução destas duas funções, é enviada a resposta ao cliente. Todo este processo está exemplificado na Fig. 5.4.

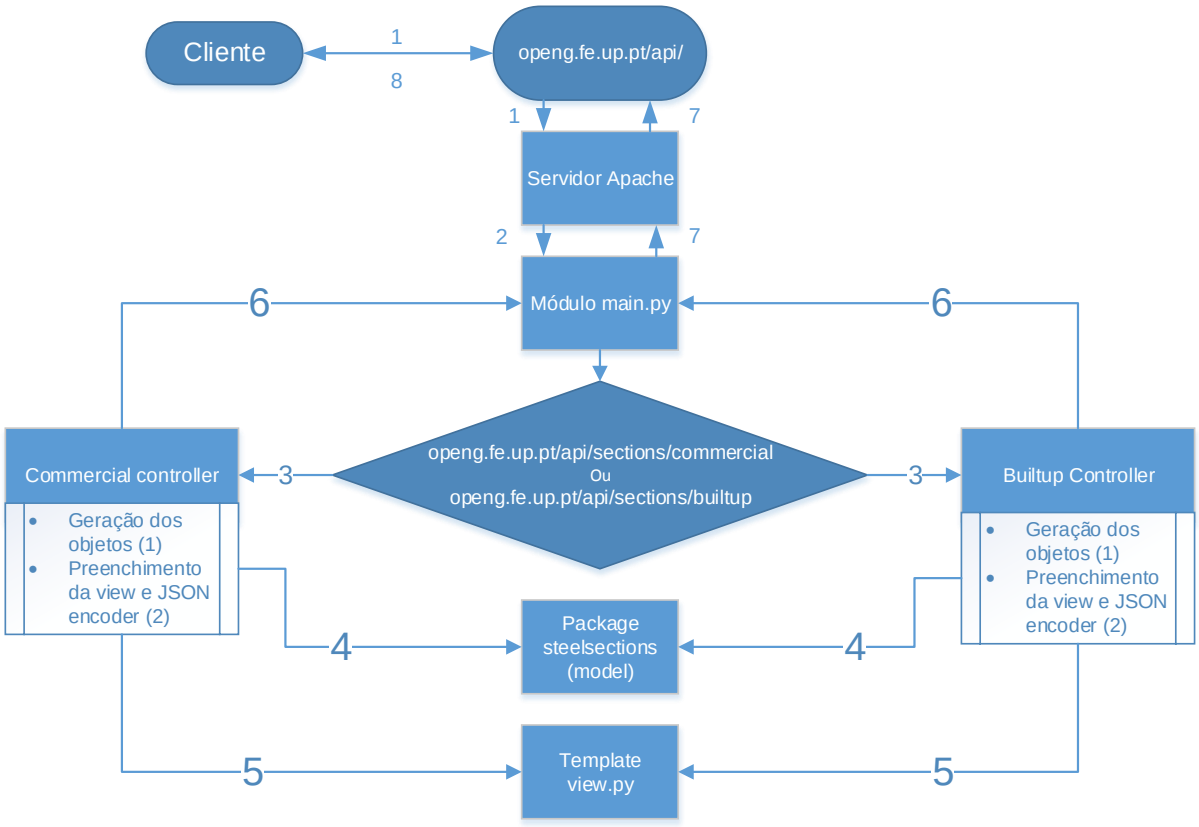


Fig. 5.4 - Articulação entre os vários componentes do modelo MVC

Na Fig. 5.4 pode-se identificar os vários processos referidos:

1. Pedido HTTP do cliente ao *web serivice*, que é recebido na porta 80 do servidor através do *Apache*;
2. Reencaminhamento do pedido para o módulo *Python main.py*, que identifica o pedido através do objeto da classe *Flask*;
3. Reencaminhamento para o respetivo *controller* consoante o tipo de secção especificado;
4. Geração dos objetos das respetivas classes;
5. Preenchimento do *template* com a informação requerida pelo cliente;
6. Devolução da informação ao módulo *main.py*;
7. Entrega da resposta ao servidor *Apache;*
8. Resposta do servidor HTTP para o cliente com a informação solicitada.

5.2.3. Estrutura do URL

Conforme foi referido ambos os *controllers* executam duas funções sendo que é a segunda relativa à visualização da informação por parte do cliente que será abordada neste subcapítulo. Esta visualização depende diretamente do pedido efetuado pelo cliente, podendo ser mais ou menos específica, o que terá de ser tido em conta na estruturação do URL por parte do cliente. Estando a maior parte deste pedido já especificada no subcapítulo anterior, através da divisão nos dois grandes grupos de pedidos,

existem diferenças e semelhanças entre ambos os casos. Neste capítulo será demostrada a aplicabilidade da metodologia REST.

No *whitepaper* da *APIgee* [23], são referidas boas práticas para a aplicação do REST. Uma das primeiras características das APIs que seguem esta arquitetura, apontada neste documento, é a advertência *"keep simple things simple"*. É referido que o URL deve ser tão intuitivo que quase nem deva ser necessária a sua documentação, e que este consiga "comunicar" com o cliente. Além disso o grau de especificidade deve ir aumentando assim que se avança ao longo do URL. Uma outra frase paradigmática é a seguinte: *"Nouns are good, verbs are bad."*. Isto significa que apenas nomes de coleções devem ser contidos no URL, enquanto as ações executadas nessas coleções devem estar associadas ao método do pedido HTTP.

No caso deste *web service*, apenas o pedido *GET* é utilizado na API, pois esta permite apenas a consulta de informação (como está explicado no Capítulo 3). Existindo uma grande coleção denominada *sections*, essa coleção vai sendo ramificada ao longo do URL, como já tinha sido dado a entender. A Fig. 5.5 mostra as possibilidades dessa ramificação.

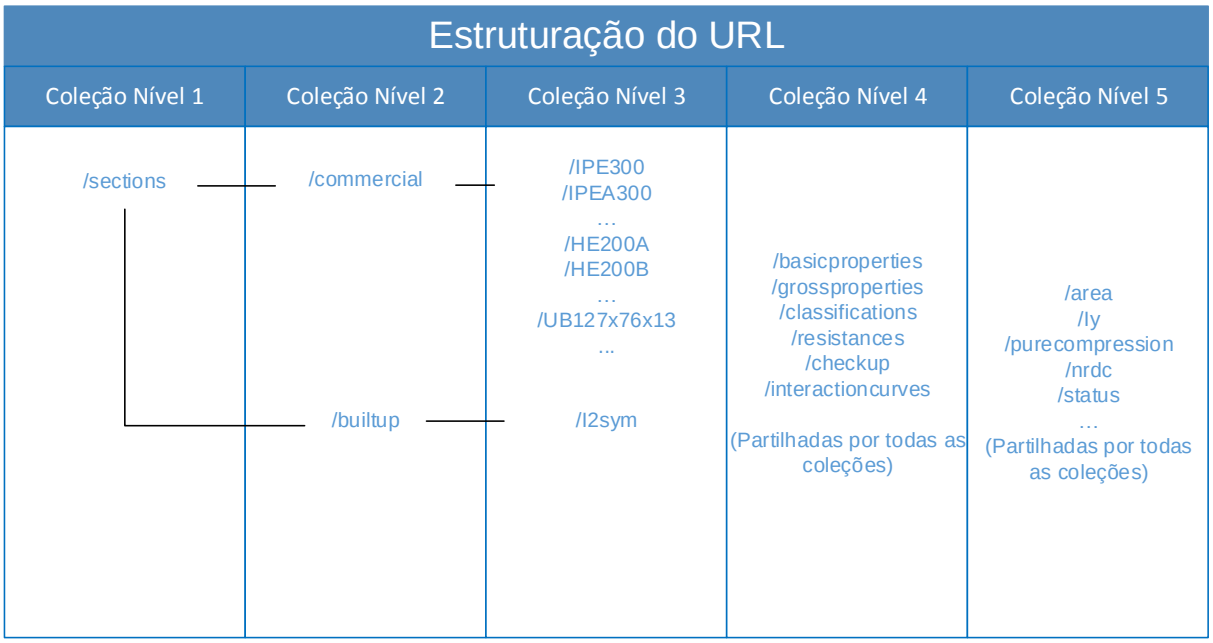


Fig. 5.5 - Estruturação do URL

Além destes conselhos, é também mencionado no referido documento que os nomes destas coleções devem vir no plural e serem o mais concretos possível. Existem ainda algumas considerações a tomar relativamente à forma como os parâmetros mais complexos vão organizados no URL. Para estes casos o autor refere que a melhor forma de abordar a complexidade nestes casos, é colocar os parâmetros na *query string*, como é por exemplo o caso dos valores de f_y, e dos parâmetros relativos às dimensões dos perfis soldados.

5.2.4. ARQUITETURA FINAL DO MODELO DE EXPANSÃO

Como tem ficado patente ao longo deste capítulo, a correta articulação entre todos estes componentes é a chave mestra para o bom funcionamento do *web service*. De igual modo, é possível esquematizar a articulação entre os agentes externos do WS, e os agentes internos, e deste modo perceber melhor o

alcance alvejado para este WS, isto porque os agentes externos assumem um papel fundamental tanto no desenvolvimento como no sucesso da API. Este papel está em consonância com o conceito já atrás referido de *open source*, onde o código é disponibilizado para ser criticado e desenvolvido, e assim obter resultados mais abrangentes e mais próximos da perfeição.

Esta identificação e separação do ambiente interno e do externo permitem uma melhor perceção do papel de cada entidade no processo construtivo. Os clientes internos podem aceder às funções diretamente ao código *Python*, e assim melhora-lo e expandi-lo, tanto na parte do *business model*, como na manutenção da correta interface com o exterior. Já os clientes externos têm outro tipo de preocupações. Enquanto clientes externos, estes podem aceder ao repositório do código *online*, e assim propor alterações e melhoramentos através, por exemplo, da tecnologia *git*. Neste momento o código já se encontra num repositório *online*, o *BitBucket*, porém ainda invisível ao público dada a fase experimental em que se encontra. Deste modo estes clientes externos assumiam o papel de clientes internos. Porém, como já demonstrado no Capítulo 3, o sucesso do WS está na quantidade de aplicações *web* criadas com base nele próprio, e é aqui onde os clientes externos têm um papel fundamental no desenvolvimento e utilização de aplicações que ajudem e obriguem o WS a ganhar cada vez mais robustez. A Fig. 5.6 representa todo o processo pretendido para o *web service*.

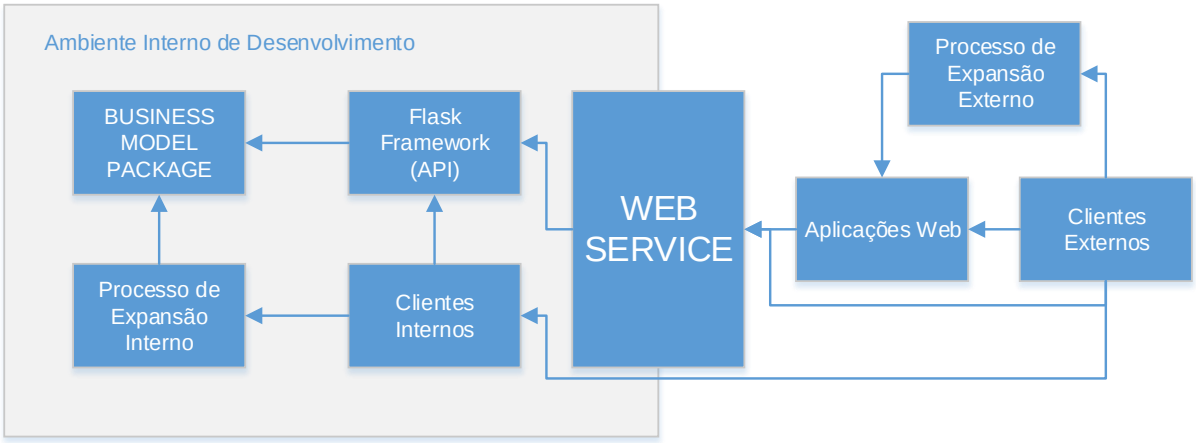


Fig. 5.6 - Articulação entre os vários agentes de processo de desenvolvimento

Esta passagem do paradigma de utilizador para *developer*, é sem duvida o maior desafio de todos, pois como já foi mencionado anteriormente, ainda existe uma inércia e por vezes relutância, nos agentes da indústria em usarem estas tecnologias, e ainda mais contribuírem para a sua expansão.

5.3. DOCUMENTAÇÃO

Já várias vezes foi referido ao longo deste documento a importância da documentação, para uma correta utilização das APIs. Como tal, durante este trabalho, foi desenvolvido um documento que explicitasse com precisão todos os comandos necessários ao uso do WS, o qual se encontra disponível através do endereço: http://openg.fe.up.pt/api/docs.

5.4. EXEMPLOS DE APLICAÇÃO

Este documento estaria incompleto sem a devida ilustração de exemplos de utilização e aplicação deste *web service*, o que se pode encontrar na Fig. 5.7, na Fig. 5.8 e na Fig. 5.9. Nestes pedidos percebe-se que a fácil manipulação do URL conduz à obtenção da informação em formato JSON.

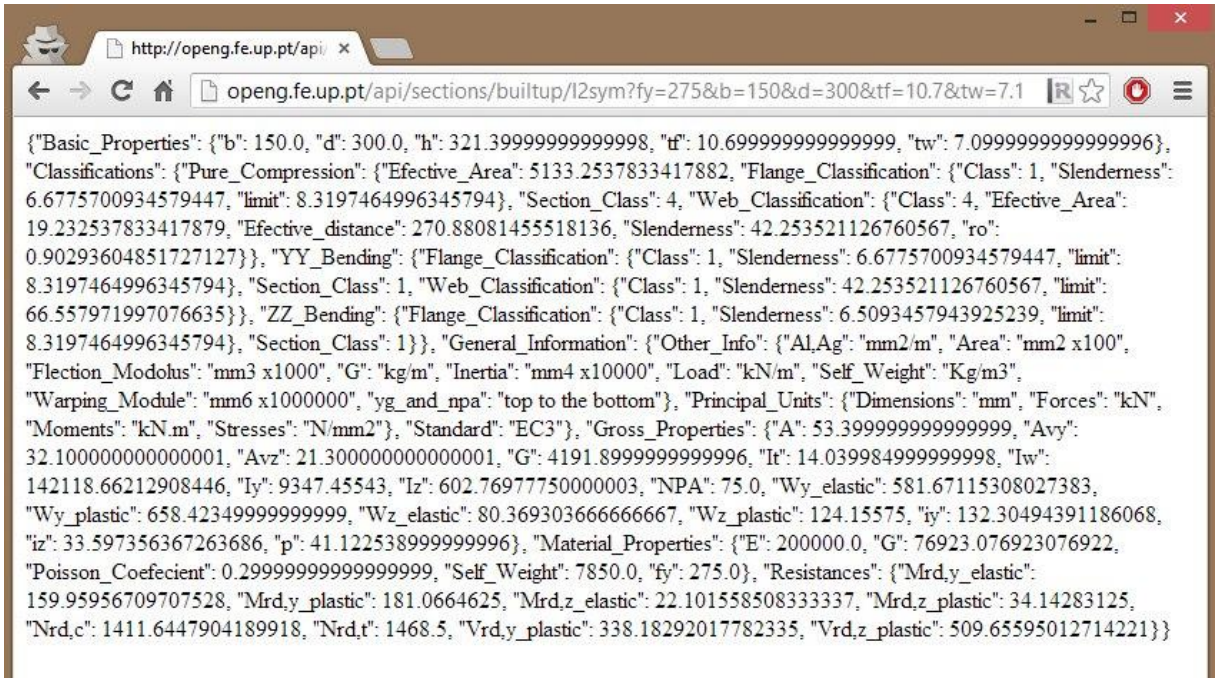


Fig. 5.7 - Exemplo de um pedido de um perfil *built up*

Fig. 5.8 - Exemplo de um pedido de um perfil IPE300

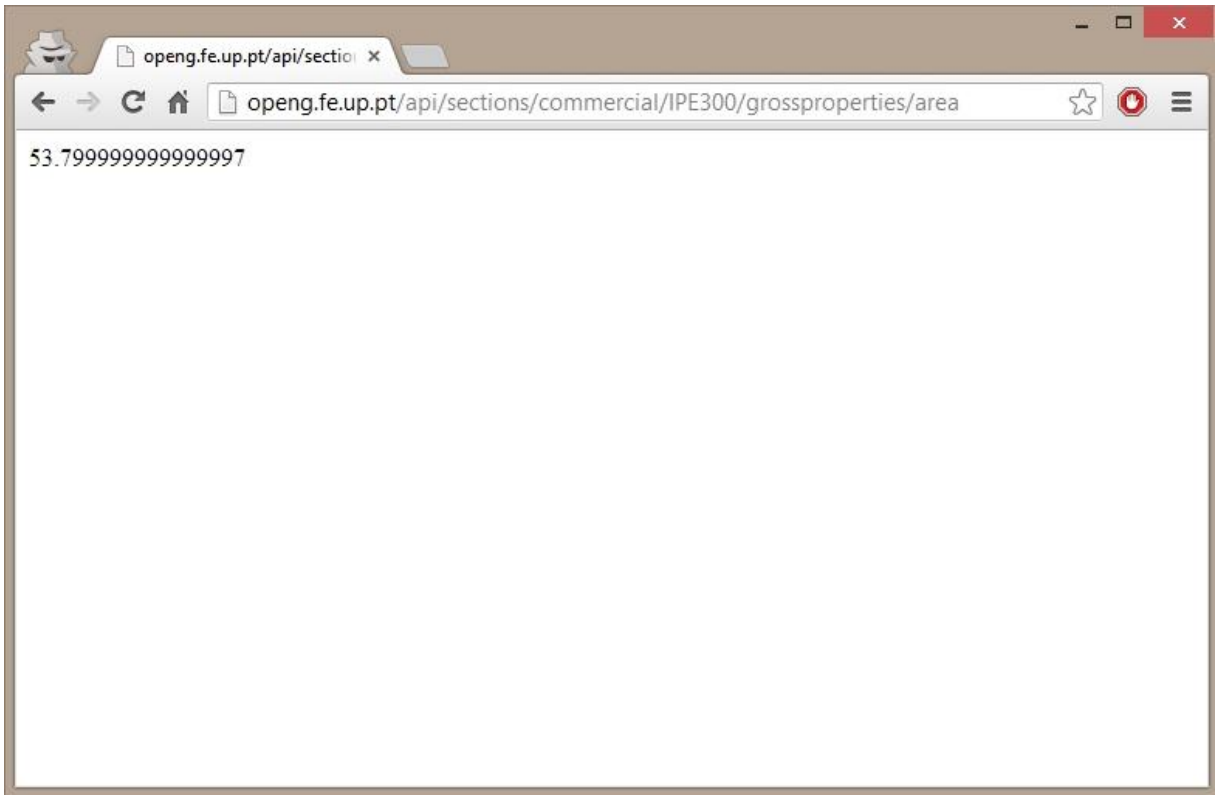
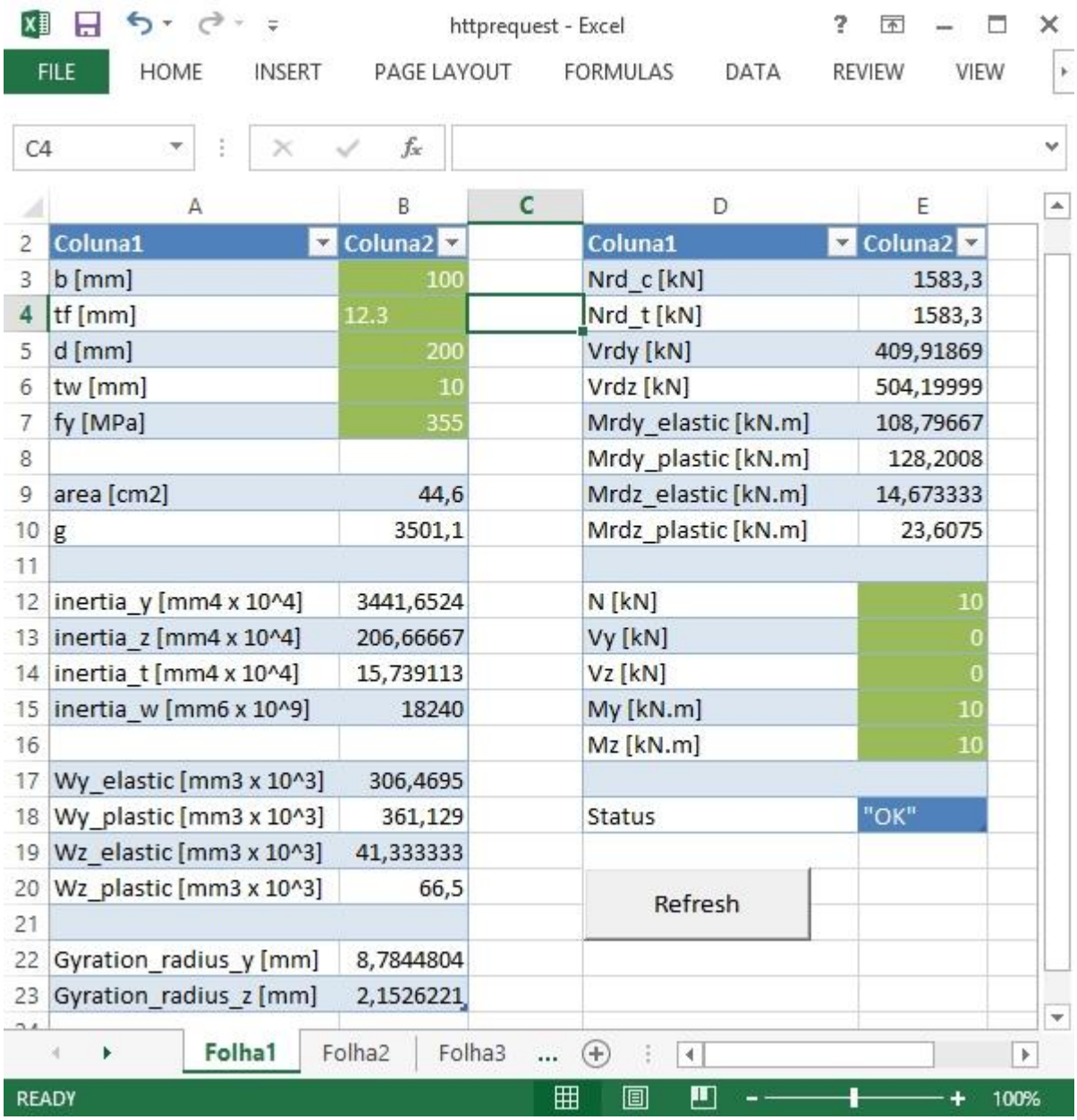


Fig. 5.9 - Exemplo de um pedido da área de um perfil IPE300

Também é percetível que a leitura destes dados pelo utilizador é bastante difícil quando feita do modo exemplificado nas figuras. Como existe a noção que no mundo prático da engenharia estrutural, a ferramenta *Microsoft Excel* é um denominador comum a qualquer gabinete de projeto de estruturas, foi criada uma folha de cálculo que permite efetuar pedidos HTTP ao servidor (Fig. 5.10). Na folha foi criada uma macro em *Visual Basic* que se encarrega destes pedidos, e onde são preenchidos as células da folha com os dados disponibilizados pelo *web service*. Esta foi uma forma encontrada para aproximar e sensibilizar os profissionais da área ao uso deste tipo de tecnologias, e demonstrar que é mais fácil o uso de funções remotas já validadas, do que toda a programação de raiz, muitas vezes sujeita a erros, e mais morosa em termos de desenvolvimento.



	A	B	C	D	E
2	Coluna1	Coluna2		Coluna1	Coluna2
3	b [mm]	100		Nrd_c [kN]	1583,3
4	tf [mm]	12.3		Nrd_t [kN]	1583,3
5	d [mm]	200		Vrdy [kN]	409,91869
6	tw [mm]	10		Vrdz [kN]	504,19999
7	fy [MPa]	355		Mrdy_elastic [kN.m]	108,79667
8				Mrdy_plastic [kN.m]	128,2008
9	area [cm2]	44,6		Mrdz_elastic [kN.m]	14,673333
10	g	3501,1		Mrdz_plastic [kN.m]	23,6075
11					
12	inertia_y [mm4 x 10^4]	3441,6524		N [kN]	10
13	inertia_z [mm4 x 10^4]	206,66667		Vy [kN]	0
14	inertia_t [mm4 x 10^4]	15,739113		Vz [kN]	0
15	inertia_w [mm6 x 10^9]	18240		My [kN.m]	10
16				Mz [kN.m]	10
17	Wy_elastic [mm3 x 10^3]	306,4695			
18	Wy_plastic [mm3 x 10^3]	361,129		Status	"OK"
19	Wz_elastic [mm3 x 10^3]	41,333333			
20	Wz_plastic [mm3 x 10^3]	66,5		Refresh	
21					
22	Gyration_radius_y [mm]	8,7844804			
23	Gyration_radius_z [mm]	2,1526221			

Fig. 5.10 - Exemplo de aplicação do *web service* em folha de cálculo Excel

Na Fig. 5.11 apresenta-se o código em *Visual Basic* que permite efetuar o pedido HTTP através da folha de cálculo.

```
Dim url As String
url = "http://openg.fe.up.pt/api/sections/builtup/I2sym/grossproperties/area

Dim http As New WinHttpRequest
http.Open "GET", url, False
http.Send

Dim area As String
area = http.ResponseText

Range("area").Value = area
```

Fig. 5.11 - Código *Visual Basic* que possibilita o pedido HTTP

5.5. CONSIDERAÇÕES FINAIS

Ao longo deste capítulo houve a preocupação em expor ao leitor de forma detalhada, o modo como foram implementadas as tecnologias presentes neste trabalho. No próximo e final capítulo serão dadas todas as conclusões referentes a este trabalho, e também possíveis futuros desenvolvimentos.

6
CONCLUSÕES E FUTUROS DESENVOLVIMENTOS

O grande objetivo proposto para esta dissertação, que passava pela criação de um *web service* que disponibilizasse informações relativas a características de perfis metálicos, foi atingido na sua plenitude. Contudo, além disso, inúmeras conclusões podem ser extraídas do desenvolvimento do *web service*, já que este careceu de um processo de aprendizagem bastante exaustivo.

Em primeiro lugar, há que assinalar que foi demonstrado de uma forma clara ao longo deste trabalho, que o mundo da engenharia civil deve cada vez mais olhar para as TI como um aliado, e não com o estigma de que estes dois campos não se articulam. Ficou patente neste trabalho que isso não é mais que uma inverdade, demonstrando-se que nos dias que correm, com a quantidade de informação gratuita existente na *web*, e com a migração do paradigma de aprendizagem para cursos igualmente gratuitos *online* onde existe um universo infinito de áreas de conhecimento, só por desleixo o mundo da engenharia civil pode ficar alheio a estas mudanças na sociedade. Ficou demonstrado que a integração do conhecimento do engenheiro civil em aplicações que lhe poupem tempo e dinheiro, não é, afinal, tão utópica.

Este capítulo estaria incompleto sem uma alusão à linguagem de programação utilizada neste *web service*, a linguagem *Python*. Ficou evidenciado que as vantagens do *Python*, incluindo como é óbvio neste pacote as bibliotecas *Numpy*, o *Matplotlib* e a também a *Flask web framework*, que foram determinantes no sucesso do desenvolvimento do *web service*. É importante que no futuro se continue a valorizar esta linguagem que se molda com grande versatilidade à engenharia como ficou provado. Com o *Python* foi possível executar cálculo de engenharia estrutural de alguma complexidade, e ao mesmo tempo sem abandonar esta linguagem, criar a ponte para a disponibilização ao público através da internet. É importante que se continuem a produzir cada vez mais *packages* como o que foi criado, que diversifiquem as áreas de conhecimento, e também aumentar o já criado.

Conseguiu-se também tirar algumas conclusões acerca da utilização das curvas de interação no apoio ao projeto de estruturas metálicas. Como ficou demonstrado no Capítulo 4, existem grandes vantagens na geração destas curvas para o apoio no dimensionamento e verificação de segurança das secções, já que possibilitam a visualização da classe da secção, e consequente o tipo de análise a ser efetuado, para um dado par de esforços de esforço axial conjugado com momento fletor. Tal como ficou patente no final desse capítulo, nem sempre o regulamento é conservador nas suas análises, ficando aqui a sugestão para um futuro trabalho, o estudo desta imprecisão.

Para finalizar, há que apontar o caminho para onde deverá caminhar esta articulação entre a engenharia estrutural e os *web services*. Em primeiro lugar há que continuar com a aposta no desenvolvimento de APIs cada vez mais robustas, com mais funcionalidades, e mais facilmente

utilizáveis de modo a chegarem a cada vez mais publico para. Paralelamente ao desenvolvimento das APIs, existe um vasto leque de possibilidades para a criação de aplicações suportadas por este WS, ou por outros que possam ser produzidos. Com o desenvolvimento das ferramentas ao nível do *browser* como o HTML5 e o WebGL torna-se aliciante o desenvolvimento de aplicações *web* cada vez mais gráficas com uma grande incidência no 3D, e assim conseguir ambientes cada vez mais imersivos para os utilizadores, podendo assim estas tecnologias competirem com os programas comerciais existentes no mercado.

BIBLIOGRAFIA

[1] Garcia, J., *Desenvolvimento de uma plataforma web para aplicações de cálculo estrutural*. 2012.

[2] Eischen, K., *Information Technology: History, Practice and Implications for Development*, 2000.

[3] Stuttard, D. and M. Pinto, *The web application hacker's handbook: discovering and exploiting security flaws*. 2007: Wiley. com.

[4] Casteleyn, S., *Engineering Web Applications*. 2009: Springer.

[5] Kappel, G., et al., *Web engineering: The discipline of systematic development of web applications*. 2006: John Wiley & Sons.

[6] *MOOC List - Programming*. 2013 [visualizado em Junho de 2013]; Disponível em: http://www.mooc-list.com/tags/programming.

[7] Majowiecki, M., *Architecture & Structures: Ethics in free-form design*. Proceedings of the International Association of Shell and Spatial Structures (IASS), 2007.

[8] Narayan, L., *Computer aided design and manufacturing*. 2008: PHI Learning Pvt. Ltd.

[9] *IFC - Where it all started - The End of Babel*, 1994.

[10] *Industry Foundation Classes (IFC) data model*. [visualizado em Junho de 2013]; Disponível em: http://www.buildingsmart.org/standards/ifc.

[11] Law, K.H., *Web services in CIVIL and Structural Engineering Simulations*. 2011.

[12] Chen, H.-M. and Y.-C. Lin, *Web-FEM: An internet-based finite-element analysis framework with 3D graphics and parallel computing environment*. Advances in Engineering Software, 2008. **39**(1): p. 55-68.

[13] *ifcwebserver*. [visualizado em Junho de 2013]; Disponível em: https://code.google.com/p/ifcwebserver/.

[14] Armstrong, D.J., *The quarks of object-oriented development*. Commun. ACM, 2006. **49**(2): p. 123-128.

[15] Nirosh, L.W.C. *Introduction to Object Oriented Programming Concepts (OOP) and More*. [visualizado em Junho de 2013]; Disponível em: http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep#Architecture.

[16] Scott, M.L., *Programming Language Pragmatics, Third Edition*. 2009: Morgan Kaufmann Publishers Inc. 942.

[17] Stroustrup, B., *The 5 Programming Languages You Need to Know*, B. Think, Editor 2011.

[18] *Quotes about Python*. [visualizado em Junho de 2013]; Disponível em: http://www.python.org/about/quotes/.

[19] *The Python Tutorial*. [visualizado em Junho de 2013]; Disponível em: http://docs.python.org/2/tutorial/.

[20] Hunter, J.D., *Matplotlib: A 2D graphics environment.* Computing In Science \& Engineering, 2007. **9**: p. 90--95.

[21] Borges, L.E., *Python para desenvolvedores.* Rio de Janeiro, 2010.

[22] 3scale, N.S.L., *What is an API? Your guide to the Internet Business (R)evolution*, 2011.

[23] Mulloy, B., *Web API Design - Crafting Interfaces that Developers Love*.

[24] json.org. *Introducing to JSON*. [visualizado em Junho de 2013]; Disponível em: http://www.json.org/.

[25] Bray, T.P., Jean; Sperberg-McQueen, C. M.; Maler,Eve; Yergeau,François. *Extensible Markup Language*. 2008 [visualizado em Junho de 2013]; Disponível em: http://www.w3.org/TR/REC-xml/.

[26] Autodesk. *Exploiting Autodesk Robot Structural Analysis Professional API for Structural Optimization*. [visualizado em Junho de 2013]; Disponível em: http://bimandbeam.typepad.com/AUTODESKROBOT_structuralanalysisprofessional_WHITEPAPER_final.pdf.

[27] Fielding, R., et al., *Hypertext transfer protocol–HTTP/1.1*, 1999, RFC 2616, June.

[28] Ronacher, A. *Flask, web development one drop at a time*. 2010 [visualizado em Junho de 2013]; Disponível em: http://flask.pocoo.org/.

[29] Reenskaug, T.M.H., *The original MVC reports.* 1979.

[30] CEN, *Eurocódigo 3 - Projecto de estruturas de aço*. 2010: IPQ.

[31] da Silva, L.S. and H. Gervásio, *Manual de dimensionamento de estruturas metálicas: métodos avançados :eurocódigo 3 : projecto de estructuras de aço*. 2007: CMM-Associaçao Portuguesa de Construçao Metálica e Mista.

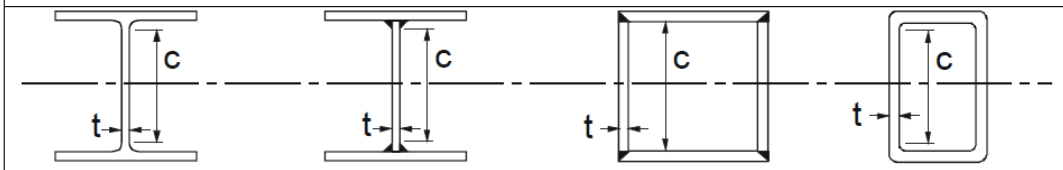
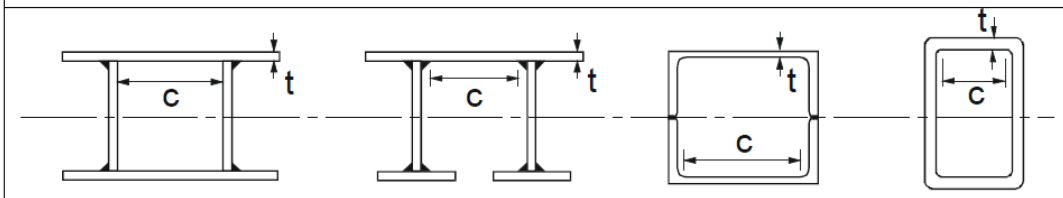
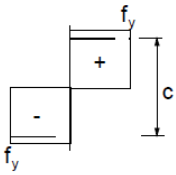
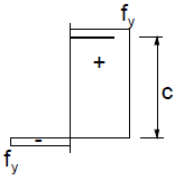
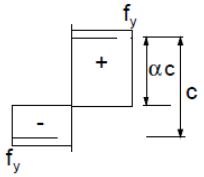
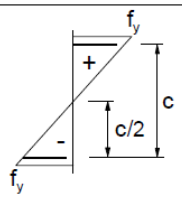
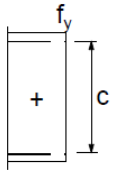
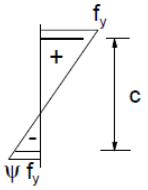
[32] *Instrucción de Acero Estructural*, 2006, Ministerio de Fomento.

[33] Barros, H. and J. Figueiras, *Tabelas e Ábacos de Dimensionamento de Secções de Betão Solicitadas à Flexão ea Esforços Axiais Segundo o Eurocódigo 2.* Porto, Portugal, 2010.

ANEXOS

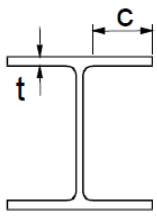
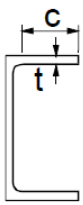
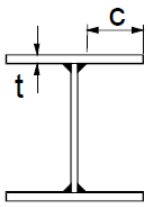
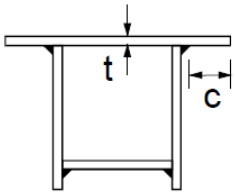
ANEXO A

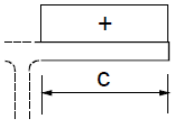
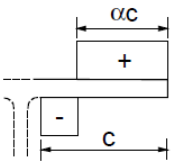
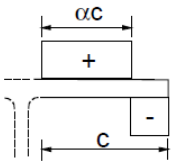
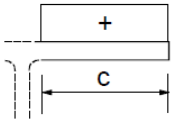
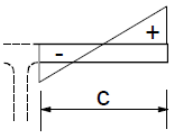
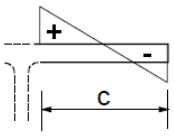
REGRAS DE CLASSIFICAÇÃO DE SECÇÕES

Componentes internos comprimidos			
			Eixo de flexão
			Eixo de flexão
Classe	Componente solicitado à flexão	Componente solicitado à compressão	Componente solicitado à flexão e à compressão
Distribuição das tensões nos componentes (compressão positiva)			
1	$c/t \le 72\varepsilon$	$c/t \le 33\varepsilon$	quando $\alpha > 0{,}5$: $c/t \le \frac{396\,\varepsilon}{13\,\alpha - 1}$ quando $\alpha \le 0{,}5$: $c/t \le \frac{36\,\varepsilon}{\alpha}$
2	$c/t \le 83\varepsilon$	$c/t \le 38\varepsilon$	quando $\alpha > 0{,}5$: $c/t \le \frac{456\,\varepsilon}{13\,\alpha - 1}$ quando $\alpha \le 0{,}5$: $c/t \le \frac{41{,}5\,\varepsilon}{\alpha}$
Distribuição das tensões nos componentes (compressão positiva)			
3	$c/t \le 124\varepsilon$	$c/t \le 42\varepsilon$	quando $\psi > -1$: $c/t \le \frac{42\varepsilon}{0{,}67 + 0{,}33\psi}$ quando $\psi \le -1$[*]: $c/t \le 62\varepsilon(1-\psi)\sqrt{(-\psi)}$

$\varepsilon = \sqrt{235/f_y}$	f_y	235	275	355	420	460
	ε	1,00	0,92	0,81	0,75	0,71

Anexo A.1 – Quadro do EC3 para classificação de elementos internos de uma secção

Banzos em consola			
			
Secções laminadas		Secções soldadas	

Classe	Componente solicitado à compressão	Componente solicitado à flexão e à compressão	
		Extremidade comprimida	Extremidade traccionada
Distribuição das tensões nos componentes (compressão positiva)			
1	$c/t \le 9\varepsilon$	$c/t \le \frac{9\varepsilon}{\alpha}$	$c/t \le \frac{9\varepsilon}{\alpha\sqrt{\alpha}}$
2	$c/t \le 10\varepsilon$	$c/t \le \frac{10\varepsilon}{\alpha}$	$c/t \le \frac{10\varepsilon}{\alpha\sqrt{\alpha}}$
Distribuição das tensões nos componentes (compressão positiva)			
3	$c/t \le 14\varepsilon$	$c/t \le 21\varepsilon\sqrt{k_\sigma}$ Para k_σ ver a EN 1993-1-5	

$\varepsilon = \sqrt{235/f_y}$	f_y	235	275	355	420	460
	ε	1,00	0,92	0,81	0,75	0,71

Anexo A.2 – Quadro do EC3 para classificação de elementos externos de uma secção

ANEXO B

CÓDIGO *PYTHON*

```
from __future__ import division
from flask import Flask, Blueprint
from controllers import commercial_controller, I2sym_controller

app = Flask(__name__)

sections_handler = Blueprint("sections_handler",__name__)
commercial_handler = Blueprint("commercial_handler",__name__)
builtup_handler = Blueprint("builtup_handler",__name__)

connections_handler = Blueprint("connections_handler",__name__)

@app.route("/")
def main():
    return "Welcome to this awsome API!"

@builtup_handler.route("/<typ>.<ext>", defaults = {"prop":None, "func":None})
@builtup_handler.route("/<typ>", defaults = {"prop":None, "func":None,
"ext":"json"})
@builtup_handler.route("/<typ>/<prop>.<ext>", defaults = {"func":None})
@builtup_handler.route("/<typ>/<prop>", defaults = {"func":None, "ext":"json"})
@builtup_handler.route("/<typ>/<prop>/<func>.<ext>")
@builtup_handler.route("/<typ>/<prop>/<func>", defaults = {"ext":"json"})
def builtup(typ, prop, func, ext):
    if typ == "I2sym":
        sec, actions = I2sym_controller.create_obj()
        return I2sym_controller.rendering(sec, actions, prop, func, ext)

@commercial_handler.route("/<typ>.<ext>", defaults = {"prop":None, "func":None})
@commercial_handler.route("/<typ>", defaults = {"prop":None, "func":None,
"ext":"json"})
@commercial_handler.route("/<typ>/<prop>.<ext>", defaults = {"func":None})
@commercial_handler.route("/<typ>/<prop>", defaults = {"func":None, "ext":"json"})
@commercial_handler.route("/<typ>/<prop>/<func>.<ext>")
@commercial_handler.route("/<typ>/<prop>/<func>", defaults = {"ext":"json"})
def commercial(typ, prop, func, ext):
    sec, actions = commercial_controller.create_obj(typ)
    return commercial_controller.rendering(sec, actions, prop, func, ext)

app.register_blueprint(sections_handler, url_prefix='/sections')
app.register_blueprint(commercial_handler, url_prefix='/sections/commercial')
app.register_blueprint(builtup_handler, url_prefix='/sections/builtup')

app.register_blueprint(connections_handler, url_prefix='/connections')

if __name__ == "__main__":
    app.run(debug=False)
```

Anexo B.1 – Módulo *main.py*

```
import json
import dicttoxml
from flask import request
from controllers import steel_controller
from models.steelsections.types import commercial
from models.steelsections.actions import Actions
from views import view, view_filter

def create_obj(sec):

    act = [0, 0, 0, 0, 0]

    if request.args.get('n'):
        act[0]=float(request.args.get('n'))*1000
    if request.args.get('vy'):
        act[1]=float(request.args.get('vy'))*1000
    if request.args.get('vz'):
        act[2]=float(request.args.get('vz'))*1000
    if request.args.get('my'):
        act[3]=float(request.args.get('my'))*1000000
    if request.args.get('mz'):
        act[4]=float(request.args.get('mz'))*1000000

    sec = sec.upper()
    steel = steel_controller.create_steel()
    sec = commercial.Commercial(sec, steel)
    actions = Actions(act)
    return sec, actions

def rendering(sec, actions, prop, func, ext):
    if prop != None:
        prop = prop.lower()
    if func != None:
        func = func.lower()
    ext = ext.lower()
    if prop=="interactioncurves":
        if func=="ybending":
            points = sec.get_n_my_points(1000)
            return json.dumps(points.tolist())#, json.dumps({"axial": actions.n}),
json.dumps({"y moment": actions.my})
        elif func=="zbending":
            points = sec.get_n_mz_points(1000)
            return json.dumps(points.tolist())#, json.dumps({"axial": actions.n}),
json.dumps({"z moment": actions.mz})
        else:
            points_y = sec.get_n_my_points(1000)
            points_z = sec.get_n_mz_points(1000)
            return json.dumps(points_y.tolist()), json.dumps(points_z.tolist())

    data = view.render(sec, actions)
    com_view = view_filter.filter(data, prop, func)
    if ext == "json":
        return json.dumps(com_view, sort_keys=True)
    else:
        return dicttoxml.dicttoxml(com_view)
```

Anexo B.2 – Módulo *comercial_controller.py*

```
import sqlite3
from database import path
from models.steelsections.types.__init__ import *
from models.steelsections import plates

class Commercial(object):
    def __init__(self, section, mat):
        connection = sqlite3.connect(path)
        cur = connection.cursor()
        cur.execute("SELECT section, G, h, b, tw, tf, r, A, hi, d, p_min, p_max,
Al, Ag, In_y, Wel_y, Wpl_y, iy, Av_z, In_z, Wel_z, Wpl_z, iz, ss, It, Iw FROM
section_tables WHERE url_name=?", [section])
        data = cur.fetchone()

        self.mat = mat
        self.name = data[0]
        self.G = data[1]
        self.load = self.G * 9.81 / 1000
        self.h = data[2]
        self.b = data[3]
        self.tw = data[4]
        self.tf = data[5]
        self.r = data[6]
        self.area = data[7]*100
        self.hi = data[8]
        self.d = data[9]
        self.p_min = data[10]
        self.p_max = data[11]
        self.Al = data[12]
        self.Ag = data[13]

        self.flange = plates.Plate('flange', self.b, self.tf, self.mat)
        self.web = plates.Plate('web', self.tw, self.hi, self.mat)

        self.inertia_y = data[14]*10000
        self.wy_el = data[15]*1000
        self.wy_pl = data[16]*1000
        self.iy = data[17]*10

        self.Av_z = data[18]*100
        self.Av_y = 2*self.b*self.tf

        self.inertia_z = data[19]*10000
        self.wz_el = data[20]*1000
        self.wz_pl = data[21]*1000
        self.iz = data[22]*10

        self.ss = data[23]
        self.It = data[24]*10000
        self.Iw = data[25]*1000000000
```

Anexo B.3 – Excerto do módulo *commercial.py*