



Universidade do Minho
Escola de Engenharia
Mestrado Integrado em Engenharia Informática

Unidade Curricular de Interoperabilidade Semântica

Ano Letivo de 2017/2018

Relatório do Trabalho Prático de Interoperabilidade Semântica

Bruno Pereira (a75135) e Maria Ana de Brito (a73580)

Junho, 2018

IS

Índice

1. Introdução	1
2. APIs	2
2.1. Primeira API	2
2.2. Segunda API	2
2.3. Terceira API	3
3. Arquitetura do Sistema	4
4. Diferença nas soluções implementadas	5
4.1. Base de dados	5
4.2. Ficheiros temporários	6
5. Web Service	7
6. Desempenho e possíveis melhorias	8
7. Interface	10
8. Conclusões	12

Índice de Figuras

Figura 1 - Arquitetura do sistema	4
Figura 2 - Modelo lógico da base de dados	5
Figura 3 - Tabela com Informação de EID	11

Índice de Tabelas

Tabela 1 - Resultado dos testes relativos ao tempo de um pedido	8
-----------------------------------------------------------------	---

1. Introdução

O projeto prático da Unidade Curricular de Interoperabilidade Semântica tem como objetivo disponibilizar as publicações de um autor, dado o seu número de ORCID. Assim, através deste identificador único mostramos numa tabela as informações dos trabalhos em que o autor está referenciado. Pretende-se, então, mostrar os títulos, anos e locais de publicação, bem como outros número identificativos do tipo de publicação, tais como o EID (*Scopus* ID), o WOS (*Web Of Science* ID), o DBLP e o SJR. Além disso, em alguns casos são ainda apresentados a lista dos autores do trabalho, o número de citações total e nos últimos três anos.

ORCID é uma organização que permite que todos os que participem em investigações possam ser unicamente identificados e, assim, conectados através das suas afiliações e contribuições, abrangente a todas as áreas científicas. Os trabalhos podem ser publicados em diversas plataformas, como, por exemplo, a *Scopus*, que se trata de um repositório que armazena informação acerca de artigos científicos aí publicados.

O relatório prático está dividido em oito partes, sendo que esta, a Introdução, é a primeira. De seguida, apresentam-se as APIs que foram usadas para retirar os dados necessários. A terceira secção diz respeito à arquitetura do sistema, enquanto que na quarta diferenciam-se as soluções implementadas. A quinta secção, *Web Service*, explica a implementação do serviço *web* desenvolvido. Por fim, faz-se uma avaliação do desempenho do modelo e sugerem-se possíveis melhoras. Na última secção, tecem-se as conclusões acerca do projeto prático.

2. APIs

2.1. Primeira API

A primeira API a ser usada no desenvolvimento deste projeto corresponde ao sistema ORCID, que através de um identificador único (o número de ORCID) disponibiliza as publicações que lhe estão associadas. Para tal foi usado o seguinte *url* que nos fornece, em formato XML, os trabalhos do autor:

https://pub.orcid.org/v2.1/NÚMERO_DE_ORCID/works

O número de ORCID do autor segue a forma de XXXX-XXXX-XXXX-XXXX e se usarmos, por exemplo, o número 0000-0003-4121-6169, o *link* tomaria o seguinte aspeto:

<https://pub.orcid.org/v2.1/0000-0003-4121-6169/works>

Desta forma, recebemos, então, um ficheiro XML com toda a informação acerca do autor e das suas publicações. Como apenas pretendemos retirar os dados acerca do título, ano, local de publicação, EID, WOS e DBLP (caso estes três últimos existam), foi necessário proceder à filtração da informação e extrair os campos relativos às *tags common-title*, *common:external-id-value*, *work:type* e *common:year*.

2.2. Segunda API

A segunda API diz respeito ao repositório *Scopus*, por isso só as publicações que tenham sido importadas daqui possuem um número de EID. O *url* que permite o acesso a esta API é o seguinte:

https://api.elsevier.com/content/abstract/citations?scopus_id=NÚMERO_DE_EID&apiKey=xxxxxxxxxxx&httpAccept=application%2Fxml

Como podemos verificar, é necessário a existência de uma *apiKey*. Esta pode ser gerada em dev.elsevier.com ou podemos usar a chave-exemplo fornecida nesse *website*. No entanto, após várias tentativas, o grupo só conseguiu aceder à API uma única vez, em que guardou o ficheiro XML localmente. Assim, só temos acesso apenas aos autores e ao número de citações correspondentes à publicação com EID igual a 38349047757. Este ficheiro é usado como exemplo, embora o acesso à API não esteja disponível.

Tal como na API anterior, tivemos que proceder à filtração da informação pretendida, que estava contida nas *tags index-name*, *rangeColumnTotal* e *grandTotal*.

2.3. Terceira API

A terceira API corresponde às publicações feitas em revistas ou séries, ou seja, possuem um número ISSN, indexadas no sistema SCIMAGO. Pretendemos retirar desta API o número de SJR. Desta forma, o *url* de acesso é o seguinte:

<https://api.elsevier.com/content/serial/title/issn/1947-315X?apiKey=xxxxxxxxxx>

À semelhança do caso anterior, não conseguimos aceder a esta API. Infelizmente nem conseguimos aceder uma única vez, logo não possuímos informação nenhuma acerca da estrutura dos dados em XML que nos seriam fornecidos. Desta forma, não foi possível obter um ficheiro exemplo, logo quando se apresenta as publicações na tabela, os dados relativos à coluna SJR encontram-se sempre vazios.

3. Arquitetura do Sistema

O sistema desenvolvido neste projeto é constituído por quatro componentes. A primeira, o cliente, desenvolvido em HTML (e suportado por JavaScript) efetua pedidos HTML a um *Web Service*, que lhe deverá responder com um ficheiro JSON. O *Web Service* para suportar estes pedidos, acede a 3 APIs externas onde irá receber os dados em formato XML. Estes dados serão processados pelo *Web Service*, armazenados numa base de dados (MySQL) para depois serem enviados para o cliente.

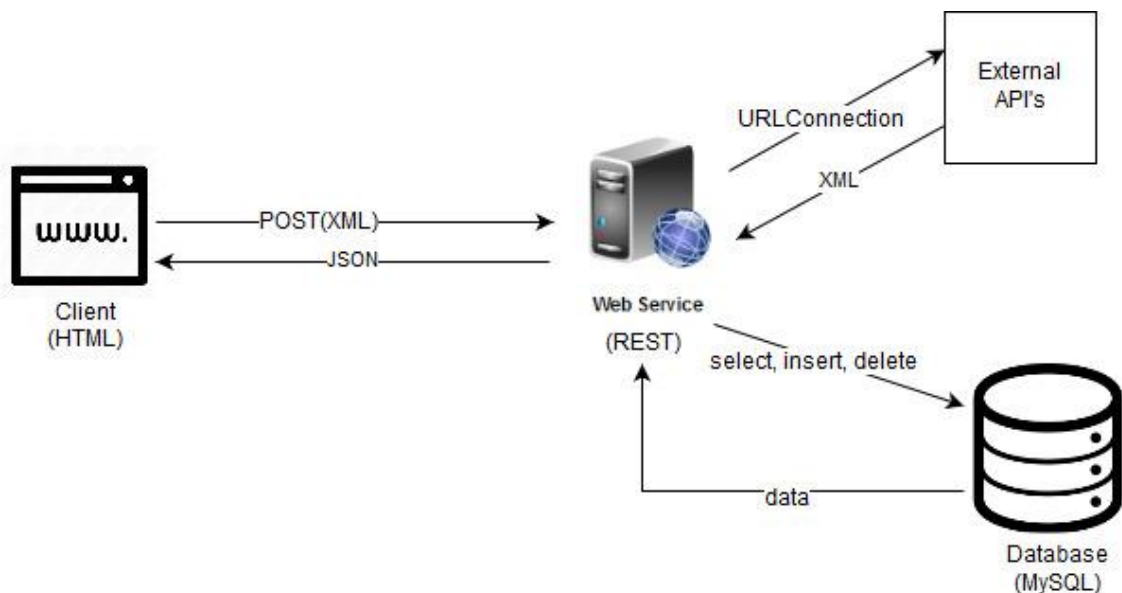


Figura 1 - Arquitetura do sistema

Esta arquitetura, ao longo do desenvolvimento do projeto, sofreu alterações. Das 3 APIs externas, só é possível aceder a uma delas (devido a problemas de permissões), enquanto que as outras duas só podem ser acedidas localmente, através de um ficheiro guardado previamente. No entanto, não conseguimos obter o ficheiro correspondente à 3ª API (capaz de retirar o *Impact Factor*) e na segunda API, só temos dados referentes ao *Scopus Id* 38349047757, como já foi referido.

A segunda alteração consistirá no desenvolvimento de entrega de informação ao cliente sem os dados serem armazenados na base de dados, no entanto, esta situação será analisada numa próxima secção.

4. Diferença nas soluções implementadas

Para analisar a complexidade do sistema, decidimos implementar duas soluções para a resolução do problema, sendo que ambas iriam apenas variar na forma como os dados retirados das APIs eram armazenados. Ou são armazenados numa base de dados, ou não são armazenados em lado algum.

4.1. Base de dados

De modo a conseguirmos armazenar todos os dados na base de dados, necessitamos de considerar três entidades: *autor*, *work* e *scopus*. Estas entidades serão traduzidas em tabelas na base de dados, em que os atributos destas tabelas serão os dados retirados das APIs. A tabela *Author_has_Work* representa a relação N-N entre a tabela *Author* e a tabela *Work*, significando que um autor pode ter vários trabalhos e um trabalho pode ter sido feito por vários autores. De seguida, apresenta-se o modelo lógico da base de dados.

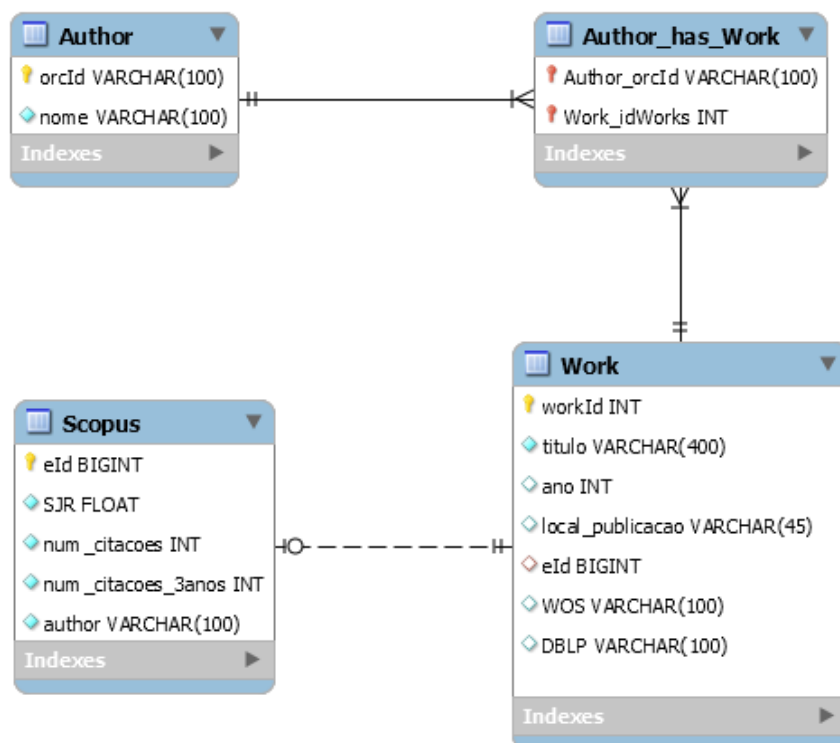


Figura 2 - Modelo lógico da base de dados

Considerando que os dados estão em APIs externas, possuir uma base de dados representa um problema grande em termos de manter os dados atualizados. Não nos podemos cingir a armazenar os dados uma única vez e, se os dados relativos a um identificador ORC, aceder sempre à base de dados, em vez das APIs. Se, por exemplo, um autor tiver removido um trabalho, esse trabalho não pode aparecer novamente na base de dados.

Tendo em conta este problema, a solução criada pode ser descrita pelos seguintes passos:

1. Obter da BD uma lista dos trabalhos relativos a um identificador ORC;
2. Se a lista for vazia, pode inserir os dados sem qualquer preocupação, estando o processo concluído;
3. Caso já existam trabalhos, obter todos os dados dos trabalhos através da 1ªAPI e guardá-los num ficheiro XML;
4. Para cada trabalho retirado do XML, verificar se ele está na base de dados, guardando os identificadores dos trabalhos que se encontram na base de dados e não no XML;
5. Caso existam identificadores de trabalhos a remover, removem-se esses trabalhos da base de dados.

4.2. Ficheiros temporários

A segunda solução consistiu em armazenar os dados retirados das APIs em ficheiros temporários, com os quais os dados iriam ser retirados e transformados em objetos JAVA. Um ponto forte desta solução é que, quando surge um novo pedido, a memória gasta nos objetos será tratada pelo *garbage collector* do JAVA, enquanto que os ficheiros temporários utilizados anteriormente irão ser reescritos.

5. *Web Service*

O *Web Service* é responsável por fornecer todos os dados ao cliente. O cliente, de modo a pedir os dados, realizará um pedido HTTP POST, em que o conteúdo da mensagem terá o formato XML, com os valores dos identificadores ORC sobre os quais deseja ver os *Works*. Estes pedidos serão do tipo POST em vez de GET, pois o utilizador pode inserir uma lista enorme de identificadores ORC, ultrapassando a capacidade de caracteres do pedido GET. Além disso, como enviamos uma lista, os identificadores ORC devem ir organizados sobre uma estrutura e nunca é boa prática enviar um pedido GET com um formato do ficheiro no URL. De modo a testar as duas soluções implementadas, o *Web Service* suporta dois tipos de pedidos.

No primeiro (de URL: <http://25.46.13.163:8080/IS/pack/webservice/orcids>) os dados obtidos pelas APIs externas não serão armazenados na base de dados, estando unicamente guardados em ficheiros temporários.

O segundo (de URL: <http://25.46.13.163:8080/IS/pack/webservice/orcidsdb>) terá o cuidado de armazenar todos os dados que obtém das APIs na base de dados.

Estes dados, independentemente da situação a testar, terão de ser convertidos para um ficheiro em formato JSON, a ser enviado para o cliente, que o irá processar e apresentar os dados em formato de tabela.

6. Desempenho e possíveis melhorias

Apesar de já existir uma noção acerca de qual das duas soluções apresentadas previamente, decidimos efetuar ainda um teste, de modo a analisar quanto tempo demora um pedido a ser processado. Os seguintes valores representam o tempo desde que recebe o pedido até ser enviado em milissegundos.

Tabela 1 - Resultado dos testes relativos ao tempo de um pedido

Solução	Nº ORCID usados	Teste 1	Teste 2	Teste 3
Sem BD	1	7699	6235	6847
Com BD	1	12655	11687	11706
Sem BD	2	7593	6921	6790

Analisando os tempos apresentados, há uma clara superioridade em não utilizar uma base de dados, visto que os tempos, tanto utilizando um único ORC ou mais, são quase metade relativamente aos tempos, se for utilizada a base de dados.

Apesar dos tempos já serem incrivelmente superiores com um único ORC, não é essa a razão pela qual só aparecem os tempos para um ORC, mas sim pelo facto da página HTML, do lado do cliente, não estar responsiva, situação que também acontece com um único ORC, daí o facto de nem ser necessário apresentar os valores com mais ORCs, pois já é possível que a solução que contém a base de dados é a pior das duas.

Assim, considerando que sempre que o cliente efetua um pedido deve receber os dados atualizados, a presença de uma base de dados do lado do servidor tornará o sistema obrigatoriamente mais ineficiente, pois os dados já provêm de uma base de dados do lado das APIs externas. A existência de uma base de dados torna-se inútil se considerarmos que o sistema estará sempre *online*, pois estaríamos a guardar os dados de uma base de dados noutra base de dados, sem necessidade aparente. No entanto, caso não seja sempre possível aceder às APIs externas, tem significado existir uma base de dados do lado do servidor onde já serão armazenados os dados.

Uma melhoria que poderia ser implementada na base de dados é, mediante um intervalo estabelecido, o próprio servidor, sem qualquer ação do cliente, fazia um *update* dos dados existentes na base de dados, verificando se estavam atuais (ou seja, se faltavam trabalhos ou se havia trabalhos que deveriam ser removidos). Esta verificação poderia ser feita

em horas não críticas, de modo a não sobrecarregar o sistema quando há uma maior afluência do sistema por parte dos clientes. Assim, neste caso tínhamos que a obtenção dos dados seria feita mais rapidamente, pois só seria necessário efetuar uns *selects*, em vez de andar a verificar a consistência dos dados cada vez que ocorre um pedido.


Esta solução tem um senão, que é possível o sistema devolver dados não atuais, no entanto, teria de ser um estudo a efetuar-se, de modo a verificar se seria benéfico manter os dados não atuais durante determinado período, ganhando, no entanto, no facto da base de dados poder funcionar em modo *offline* e sem serem necessárias verificações por cada pedido efetuado por um cliente.

7. Interface

O grupo pretendia ter uma *interface* intuitiva e simples, onde o utilizador pudesse facilmente navegar e interagir. Recorreu-se, assim, às ferramentas *HTML*, *CSS* e *Bootstrap* para criar a seguinte página:


Search an ORCID!

HomeWhat is ORCID?Why use ORCID?Search



What is ORCID?

ORCID is a non-profit organization supported by a global community of organizational members, including research organizations, publishers, funders, professional associations, and other stakeholders in the research ecosystem. We aim to help create a world in which all who participate in research, scholarship and innovation are uniquely identified and connected to their contributions and affiliations, across disciplines, borders, and time.



Why use ORCID?

ORCID provides a persistent digital identifier that distinguishes you from every other researcher and, through integration in key research workflows such as manuscript and grant submission, supports automated linkages between you and your professional activities ensuring that your work is recognized.

Curious about publications of one of our members? Search it out!

Search Publications

Title	Type of Publication	DBLP	WOS	Year	SJR	EID
First Title - An example	journal article	12345678	87654321	2015	--	--
Second Title - Another example	book chapter	11112222	22221111	2017	--	1234-1234
Third Title - Final example	book	575757575	757575757	2016	--	5790-0975

Interoperabilidade Semântica - Bruno Pereira (a75135) e Maria Ana de Brito (a73580)

Desta forma, o utilizador pode-se informar acerca da organização ORCID e procurar as publicações de um autor, inserindo o seu número de ORCID. Como se pode verificar é-lhe devolvida uma tabela que contém várias informações. Se seleccionarmos a hiperligação presente nos números de EID é apresentada uma tabela que contém os restantes dados, como se pode observar:

Search an ORCID!

Curious about publication

1111-2222-3333-4444

Search Publications

Number of Citations

100

Number of Citations in the last 3 years

45

Authors

Author1, Author2, Author3

Title	Type of Publication	DBLP	WOS	Year	SJR	EID
First Title - An example	journal article	12345678	87654321	2015	--	--
Second Title - Another example	book chapter	11112222	22221111	2017	--	1234-1234
Third Title - Final example	book	575757575	757575757	2016	--	5790-0975

Interoperabilidade Semântica - Bruno Pereira (a75135) e Maria Ana de Brito (a73580)

Figura 3 - Tabela com Informação de EID

8. Conclusões

O projeto prático proposto foi desenvolvido com sucesso. Conseguimos cumprir o que tínhamos planeado. Deste modo, desenvolvemos duas soluções distintas e comparamos o seu desempenho. Assim, conseguimos distinguir qual a melhor opção e, ainda, propor melhorias à implementação.

O grupo aplicou os conceitos e conhecimentos adquiridos nas aulas da unidade curricular, particularmente o desenvolvimento de uma arquitetura robusta baseada em *web services*.

Em suma, com o desenvolvimento deste trabalho prático planeamos e implementamos um sistema que recorre ao acesso de APIs para fornecer uma lista de publicações associadas a um autor, dado o seu número identificador ORCID.