Web Site preparado para a Web Semântica: Semântica de Eventos

João Rodrigues, Diogo Laginha

jarod@student.dei.uc.pt, dmachado@student.dei.uc.pt

Resumo: *EventSemantic* é um site de eventos musicais a realizar em Portugal. À semelhança do que a LastFM [1] oferece, esta iniciativa pretende apresentar e transformar a informação de eventos de música nacionais, em dados de um nível superior. Técnicas de recolha de informação e de Web Semântica foram usadas, de modo a concluir este trabalho com sucesso.

Palavras-chave: Eventos, Música, Semântica, Portugal

1. Introdução

Este projecto realizado no âmbito da Web Semântica, nasceu de uma lacuna ignorada pelas actuais páginas web de eventos nacionais. Estes, para além de possuírem conteúdos limitados, apresentam um modelo de navegação unidimensional pouco eficaz. Ademais, os métodos de pesquisa são deveras elementares e crus, uma vez que a informação disponível contem pouco significado e valor.

EventSemantic tem então como objectivo melhorar os serviços actualmente disponíveis. Assim sendo, foi desenvolvido uma aplicação web de pesquisa e navegação de eventos nacionais de música, recorrendo a técnicas de Web Semântica, onde será igualmente incorporado um sistema de sugestões de eventos.

De modo a atingir os nossos objectivos, foram definidas várias metas ao estilo de uma metodologia de desenvolvimento iterativa: criação e população de ontologia, design de páginas web *xhtml+rdfa*, implementação dos sistemas de pesquisa, navegação e sugestão semântica.

2. Descrição da arquitectura e design do sistema

No nosso sistema, independentemente da página em que o utilizador se encontra, estará sempre visível uma barra superior representando o menu principal e uma caixa de texto associada a um botão onde se pode inserir uma pesquisa semântica.

Figura 1 – Aparência da barra superior do sistema

Como podemos ver na figura 1, a opção *Home* do menu é a que vem seleccionada de raiz. Portanto, sempre que um utilizador acede ao *url* da página, aparece um ecrã de boas-vindas com informações úteis e conselhos sobre a utilização desta. Por outro lado, quando a opção *Contact* é seleccionada, o sistema não faz mais do que mostrar uma página com informações sobre como contactar os colaboradores e programadores da página.

Quando a opção *Events* é escolhida, o utilizador é reencaminhado para a página de *browsing*, onde pode filtrar os eventos consoante determinadas características. Do lado direito estão disponíveis dois menus onde os eventos podem ser filtrados, respectivamente, por categoria e por zona geográfica ao passo que mais a cima, debaixo do menu principal, aparece um sistema de *tabs* que permite a filtragem por data.

Figura 2 – Aparência dos menus de *browsing* e listagem de eventos por *scroll*

Assim, por cada vez que o utilizador escolher uma determinada característica de um dos menus, a página é automaticamente actualizada mostrando apenas os eventos que partilham as três características que estão actualmente seleccionadas em cada um dos menus de filtragem. Como podemos ver na figura 2, as características que estão seleccionadas de raiz são *todas as datas*, *todas as categorias* e *todas as zonas geográficas*, pelo que todos os eventos da base de dados vão estar listados de inicio.

Tanto para mostrar os eventos resultantes do *browsing* como os resultantes da pesquisa, optou-se por um sistema de *scroll* dentro de um bloco. Esta abordagem faz

com que o utilizador possa fazer *scroll* sobre os eventos sem nunca perder de vista tanto os menus de filtragem como a barra do menu principal com caixa de pesquisa.

Como em alguns casos vão ser listados muitos eventos, optou-se por se mostrar apenas as informações mais importantes de cada um na listagem, como o nome, artistas, localização e data, fazendo assim com que a informação ficasse mais legível para o utilizador. Se este, posteriormente, desejar visualizar informação mais detalhada sobre um determinado evento, poderá sempre clicar no seu nome e será redireccionado para a sua página.

Na página de cada evento, para além de ser disponibilizada informação mais detalhada sobre o mesmo, são também disponibilizados *links* para eventos sugeridos pelo sistema na parte direita do ecrã. Além disso, tanto na página de cada evento como na página de listagem de eventos, estão disponíveis *links* para a página de cada artista, onde se encontra informação mais detalhada sobre este, como por exemplo em que eventos participa, que géneros musicais o caracterizam e que álbuns já lançou.

3. Ontologia usada

A nossa aplicação tem por base uma única ontologia (MusicEvents) que está definida no ficheiro Projecto_WS.owl na pasta Protege e que tem a seguinte estrutura de classes:

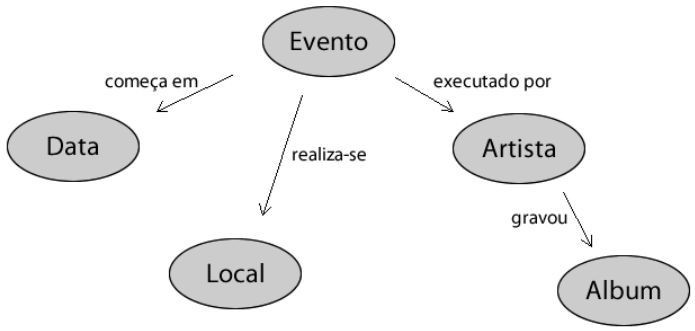

Figura 3 – Estrutura de classes da ontologia utilizada

Quanto às propriedades específicas de cada classe, estas estão definidas na seguinte tabela:

Tabela 1 – Propriedades de cada classe da ontologia utilizada

Classe	Propriedades
Evento	nome, descrição
Data	nome, nome do dia, número do dia, hora, minutos, segundos, nome do mês, número do mês, número da semana, ano, *timestamp*
Local	Nome, morada, código postal, localidade, país, latitude, longitude
Artista	nome, género, sumário, descrição
Album	nome, editora

4. Estrutura e organização do código

Na pasta *Scripts* poderão ser encontrados todos os *scripts python* que foram implementados de modo a extrair o máximo de informação possível sobre eventos da LastFM [1]. Este foi o site escolhido devido ao facto de ser completo e de possuir uma boa *API* que, apesar de ter alguns métodos interessantes, não nos permitiu retirar toda a informação que desejaríamos do site, daí termos também recorrido a técnicas de *screen scrapping* para recolha de mais informação.

Na pasta *event_semantics* encontram-se todos os ficheiros correspondentes à aplicação web. Esta aplicação foi implementada em *Django* [2], uma *framework* para aplicações web desenvolvida em *python,* e todo o código segue a organização e estrutura convencionais desta.

Portanto, sempre que há um pedido *http* ao servidor, o ficheiro urls.py vai analisar o url e, caso este seja válido, vai accionar a respectiva função de processamento. Resumindo, para cada url válido existe sempre uma função de processamento específica (*view*), que estará implementada no ficheiro views.py.

De seguida, a *view* que foi accionada vai ser executada, devolvendo no final um *template* (um ficheiro *html* com código *Django)* com a informação que foi computada associada a este (ex: o resultado de uma *query*). Este *template* é automaticamente executado sendo produzido no final um ficheiro *xhtml+rdfa* que vai ser devolvido ao utilizador.

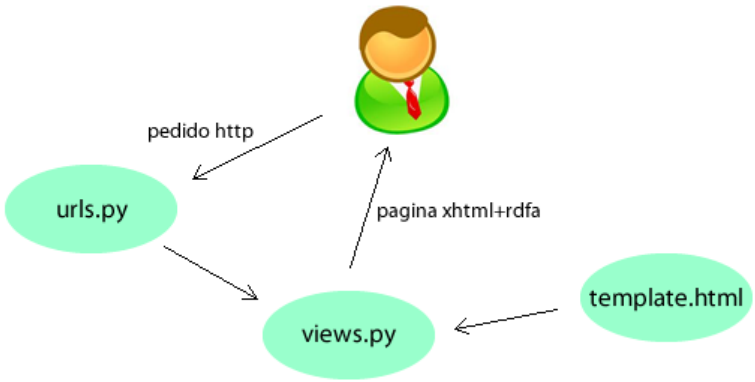

Figura 4 – Estrutura do código da aplicação web

Como todos *templates* vão repetir partes do *html* de outros (ex: menus), optou-se por implementar um ficheiro base.html com todo o *html* que é comum a todos os *templates*. Deste modo cada *template* só vai ter *html* específico e importa da base todo o *html* comum fazendo assim com que o código seja mais limpo.

Este ficheiro base.html, bem como os restantes *templates* e todos os ficheiros de processamento da aplicação web (urls, views, etc) encontram-se dentro da pasta *events* ao passo que a pasta *media* contem todos os ficheiros *javascript*, *css* e imagens.

5. Descrição dos módulos mais importantes do sistema

5.1. create_rdf.py

Este módulo, apesar de não fazer parte da aplicação web, é bastante importante na medida em que quando é executado carrega a base de dados com triplos. A sua tarefa é processar toda a informação obtida sobre os vários eventos, que se encontra espalhada por ficheiros que foram construídos previamente através de *APIs* e de técnicas de *screen scrapping*. Essa informação encontra-se em formato *xml* ou *json*, pelo que este módulo vai funcionar como uma espécie de *parser* que simplesmente a interpreta e cria triplos estabelecendo relações entre entidades.

Existem também alguns tipos de informação que não estão disponíveis nos ficheiros mas que este módulo os consegue obter através de processamento, como são o caso do número da semana e do *timestamp* do evento, informações que podem ser obtidas através do dia e da hora a que este ocorre. Destaca-se também neste módulo o

uso de expressões regulares de modo a extrair e filtrar um texto descritivo e um sumário que nos é fornecido sobre cada evento, eliminando assim o "lixo" que o *xml* fornecido contém.

5.2. get_rdf.py

Neste módulo estão definidas as classes *python* que representam cada classe da ontologia: Evento, Artista, Local, Data e Album. Todas estas classes possuem métodos que retornam não só as suas propriedades mas também outras classes relacionadas com as mesmas. Além disso, todas herdam de uma classe pai (RDFObject) que possui métodos generalizados que são comuns a todas elas, como por exemplo o *multiple_query*:

```
def multiple_query(self,prop,obj):
    return map(lambda a: obj(a),
               graph.query(""" SELECT ?art WHERE { ?ev me:%s ?art . } """
               % prop, initBindings={'ev': self.uri}))
```

Figura 5 – Função *multiple_query*

Esta função, recebendo como argumento uma propriedade/relação e um objecto, vai seleccionar todos os valores que essa propriedade/relação tem (da entidade em questão, identificada pelo *URI*) e criar um determinado objecto a partir de cada um deles, devolvendo-os no final em forma de lista. Assim, se por exemplo quisermos ter acesso aos objectos que representam todos os artistas de um determinado evento (para podermos saber os seus nomes ou os seus géneros) bastar-nos-á chamar esta função na classe Evento, passando-lhe como argumento a relação que os eventos têm com os artistas e a classe Artista, para que sejam criados objectos deste tipo a partir dos *URIs* resultantes da pesquisa:

```
class Event(RDFObject):
    def get_artists(self):
        return self.multiple_query("performed_by",Artist)
```

Figura 6 – Função *get_artists*

Existem também outras funções semelhantes como o *single_query* (para *queries* onde só interessa o saber o primeiro resultado) e o *inverse_multiple_query* (quando interessa saber o sujeito do triplo em vez da propriedade). Cada objecto possui também alguns atributos como o seu *URI*, um elemento identificativo e um peso, que será explicado mais a frente quando se falar na pesquisa semântica e nas sugestões.

5.3. views.py

Este é sem dúvida o módulo mais importante do sistema, na medida que contém todas as funções que devem ser processadas consoante o *URL* recebido. As funções mais importantes são descritas seguidamente:

5.3.1. get_objects

Ao longo de todo este módulo pode-se reparar que, apesar de haver muitas *queries SPARQL*, as palavras "SELECT" e "WHERE" só aparecem uma vez no código. Isto deve-se ao facto desta função processar as *queries* automaticamente por nós e devolver-nos uma lista com os objectos desejados, à semelhança do que acontece nas funções descritas acima na secção 4.2. Para isso, basta passar-lhe por argumento o tipo de instância que queremos seleccionar e um dicionário com as características que queremos que possua. Por exemplo, se quisermos seleccionar todos os eventos que vão acontecer no norte (latitude maior que 41) em Março de 2011, a chamada da função vai ser:

```
get_objects('Event',{
'Date' : [('Year','2011') , ('MonthNumber','3')],
'Place' : [('Lat','?l','FILTER (?l > 41)')]
} )
```

Figura 7 – Chamada da função *get_objects*

Como o primeiro argumento é a *string* "Event", a função sabe que terá de converter todos os *URIs* retornados pela *query* em objectos do tipo Evento, e retornar uma lista com os mesmos.

5.3.2. event_date, event_genre e event_zone

Estas três funções são auxiliares na medida em que constroem o dicionário que serve como *input* à função *get_objects* por nós. No caso da data, basta passar-lhe uma *string* que indique a altura em que desejamos que o evento ocorra (passado, este mês, próximo mês, esta semana ou este ano) e um objecto que indique qual o dia e a hora que deve servir como referência, que a função constrói e devolve o dicionário respectivo. Quanto ao género e à zona, no primeiro só precisamos de indicar o estilo de música que queremos que o evento tenha e no segundo só precisamos de lhe passar uma destas palavras chave: norte, centro ou sul.

5.3.3. act_date, act_genre, act_zone e browsing

O que as três primeiras funções fazem é actualizar respectivamente a data, o género e a zona, sempre que o utilizador altera os seus valores num dos menus de

filtragem do *browsing* do sistema. No final, cada uma delas vai chamar a função *browsing*, que vai passar os valores actualizados a cada uma das funções descritas na secção 4.3.2. Como cada uma delas devolve um dicionário, esta função vai fundi-los num só, que vai servir como *input* à função *get_objects*. Esta função vai devolver todos os eventos que possuem as características descritas no dicionário em simultâneo e vai passar essa informação a um *template*, juntamente com o valor actual de cada categoria. O *template* é de seguida processado e devolvido ao utilizador de maneira a que este possa visualizar os resultados da filtragem que acabou de fazer.

5.3.4. event_detail

Esta função vai ser chamada sempre que o utilizador fizer um pedido à página de um evento. Para mostrar informações sobre um determinado evento, basta passar o seu objecto ao *template* e as propriedades são mostradas ao utilizador. No entanto, também são sugeridos alguns eventos relacionados, o que faz com que esta função não seja trivial.

Para calcular os eventos mais relacionados, a função vai primeiro buscar à base de dados uma lista com todos os eventos que possuem artistas em comum com o evento corrente, dando a cada um desses eventos um peso de 8 unidades.

O mesmo vai ser aplicado para eventos que têm o mesmo género (peso de 6 unidades), que ocorrem na mesma zona geográfica (peso de 4 unidades) e que acontecem na mesma semana que o evento em questão (peso de 3 unidades).

De seguida, as quatro listas de eventos anteriormente seleccionadas vão se unir iterativamente duas a duas até ficar apenas uma lista com todos os eventos (função *reduce*). Para a união ser feita vai se verificar para cada evento de uma determinada lista se este já se encontra na outra lista ou não. Em caso afirmativo, apenas se soma o seu peso ao peso do evento igual que está na outra lista, caso contrário o evento é adicionado à mesma. Isto vai servir para que eventos que estejam presentes em mais do que uma lista, tenham o seu peso actualizado. Por exemplo, um evento que ocorra na mesma semana e na mesma zona geográfica que o evento seleccionado vai ter o peso de 7 (4+3).

No final, depois de todos os eventos estarem na mesma lista e com os seus pesos actualizados, a mesma vai ser ordenada por ordem decrescente de peso, ficando os eventos com mais peso (aqueles que partilham mais características) nas primeiras posições, sendo sugeridos ao utilizador aqueles que se encontram nas primeiras 10.

5.3.5. event_search

Quando um utilizador faz uma pesquisa no sistema, esta função é accionada, recebendo por parâmetro uma lista com as palavras chave que foram inseridas na caixa de texto da pesquisa.

Existem 8 palavras chave que o nosso sistema detecta automaticamente: artista, evento, dia, mês, ano, hora, local e estilo. Esta função começa por percorrer a lista de palavras e, para cada palavra, vai verificar se tem ou não uma destas palavras

reservadas na posição imediatamente antes ou depois. Em caso afirmativo, o sistema vai associar essas duas palavras, caso contrário estaremos na presença de uma palavra ambígua.

As palavras são sempre agrupadas em forma de pilha. Por exemplo a pesquisa "artista Shakira Lisboa" vai produzir os mesmo resultados que a pesquisa "Shakira artista Lisboa", pois o sistema vai sempre associar a palavra artista com Shakira e assim sabe que o utilizador está à procura de eventos em que participem a artista Shakira. Neste caso, como Lisboa não tem associada nenhuma palavra reservada, o sistema vai interpretá-la como sendo uma palavra ambígua que tanto pode ser uma cidade como um artista ou um estilo musical.

Resumindo, o primeiro passo é separar as palavras recebidas por argumento em palavras não ambíguas e ambíguas. Para as primeiras, vão ser executadas *queries* específicas enquanto que para as segundas vão mais gerais.

Pegando no exemplo anterior, o sistema iria executar duas *queries*: na primeira iria seleccionar todos os eventos em que participasse a artista Shakira enquanto que na segunda iria seleccionar todos os eventos que tivessem a palavra Lisboa em qualquer um destes campos: nome, dia, mês, ano, hora, nome dos artistas, género dos artistas e localidade.

De seguida, à semelhança do que acontece na função descrita na secção 4.3.4, vai ser atribuído um peso a cada um dos eventos resultantes de uma determinada *query*. O peso é atribuído de forma decrescente, sendo que as primeiras palavras a serem introduzidas irão ter um peso maior que as últimas. Voltando ao mesmo exemplo, os eventos resultantes da pesquisa "Shakira artista" iriam ter um peso superior aos eventos resultantes da pesquisa ambígua "Lisboa".

No final as várias pesquisas vão ser unidas aplicando um algoritmo semelhante ao descrito na secção 4.3.4 e são devolvidos todos os eventos ordenados por ordem decrescente de peso associados a um *template*.

6. Conclusão

Neste projecto foi desenvolvido uma aplicação web (usando a *framework Django* [2]) de pesquisa e navegação de eventos nacionais de música, recorrendo a técnicas de Web Semântica, sendo que foram constituídas metas ao longo do semestre (que foram cumpridas) de modo a atingir os objectivos propostos.

A equipa de desenvolvimento começou por recolher vários dados de eventos da LastFM [1] recorrendo à sua *API* e a técnicas de *screen scrapping*. Seguidamente foi criada uma base de dados de triplos, usando como base uma ontologia que foi desenvolvida e está descrita no capítulo 3.

No final foi construída uma aplicação web por cima da base de dados. Esta aplicação tem um *front-end* em que as páginas estão anotadas com RDFa para interacção com os clientes e com aplicações que o interpretem.

Entre as diferentes funcionalidades da aplicação destacam-se a pesquisa semântica por palavras chave, browsing de eventos e sugestão de páginas relacionadas semanticamente.

7. Referências

[1] LastFM: LastFM Web site. http://www.last.fm/home

[2] Django: Python Web Framework homepage. http://www.djangoproject.com