CaVa: an example of the automatic Generation of Virtual Learning Spaces

Ricardo G. Martini[1], Cristiana Araújo[1], Pedro Rangel Henriques[1], and Maria João Varanda Pereira[2]

[1] Algoritmi Research Centre, Department of Informatics
University of Minho - Gualtar, Braga, Portugal
[2] Algoritmi Research Centre
Institute Polytechnic of Bragança, Portugal
`rgm@algoritmi.uminho.pt`,`decristianaaraujo@hotmail.com`,`prh@di.uminho.pt`, `mjoao@ipb.pt`

Abstract. In order to construct web Learning Spaces (LS), more than collect and digitalize information, a powerful data extraction and querying engine and a sophisticated web publishing mechanism are needed. In this paper, a system to automatically construct those learning spaces based on a digital repository is presented. The system takes XML files from repositories and populates an ontology (representing the knowledge base, the core of our system) to create the triples internal representation. A Domain Specific Language (CaVa^{DSL}) will be used to specify the learning spaces based on that ontology. The formal description, written in that DSL, will be processed by Cava^{gen} engine to generate the final LS.

Keywords: Virtual Learning Spaces, Automatic Generation, DSL, Ontology, XML, RDF

1 Introduction

'Memory Institutions' like museums, archives or libraries preserve nowadays their collections, or assets, as Digital Objects (databases or annotated documents). After digitalization and recording, the immediate goal is to explore those huge sources of relevant information that constitutes the humanity's cultural heritage; this requires at least accurate search engines and powerful Web publishing mechanisms. Virtual Museums – this is, museums that are not located in a building and has no physical objects to show – sprang out in this context. On the other way around, they display in their exhibition rooms objects collected from digital repositories. In that case, exhibition rooms (that in our work we call Learning Spaces – LS) are Web pages; the visitor accesses the objects navigating on a browser [1].

To create a virtual museum in the Web, it is necessary to query the repository's digital storage, and to process (transform and relate) the returned information before publishing it as Web pages.

The work reported starts with a discussion on how to implement generic and efficient tools able to extract automatically the necessary data (concepts and relations) from the repository. We then discuss how to build the virtual museum Web pages in a systematic way using a formal description of each room written in a Domain Specific Language, CaVa$^{\mathrm{DSL}}$, designed for that purpose – in that way the building platform can be easily adapted from one project to another. Cava$^{\mathrm{gen}}$ is the generator that consumes the formal LS descriptions and creates the queries to retrieve the information from the ontology data storage to display it in the final Web pages, the Virtual Learning Space.

In the project under discussion, we deal with annotated documents, and construct a text filter capable of automatically create triples[3] that will populate the museum's ontology.This text filter translates XML (eXtensible Markup Language) documents into RDF (Resource Description Framework) notation.

As a case study, to illustrate the implementation of this process and its successful application, we will use the assets of the Museum of the Person (MP) [2][3] [4].

In Section 2, we introduce the proposed CaVa architecture that is designed to accomplish our aim: create Virtual (or Web-based) Learning Spaces from a digital repository. After the general system overview, we go into details and, in Section 3, we discuss the design and the development of the text filter, named XML2RDF translator, whose function is to transform XML documents into RDF triples. The creation of Virtual Learning Spaces (VLS) and how we extract the information stored in the ontology to display on the Web (on the VLS) is presented in Section 4. Also in this section, the approach is illustrated presenting some exhibition rooms for the Museum of the Person, as a case study to test both translators built. Finally, Section 5 presents the conclusion and directions for future work.

2 Architecture of the System

The core, or heart, of this approach is an ontology that models the knowledge domain related to the museum to be built. The platform that will be introduced, CaVa [5] (Figure 1), splits the building process into a first module, the Ingestion Function (XML2RDF), to extract data from the sources and upload the ontology triples, and a second module, the Generator (CaVa$^{\mathrm{gen}}$), to automatically generate the query for each exhibition room based on a formal specification and a subset of the main ontology, and to organize the returned information to be exhibited in adequate Web pages.

As said above, our approach can be characterized by an architecture that comprises: the repository; the Ingestion Function (M1) responsible for reading the annotated documents, extracting and preparing the data, and store the information gathered; a Data Storage (DS) that contains the ontology instances;

[3] A triple is a structure that represents a link (a semantic connection) between two concepts through a relation in the form of subject-predicate-object interpretation (e.g. U2 is-a band, Mark is-brother-of John, etc.).

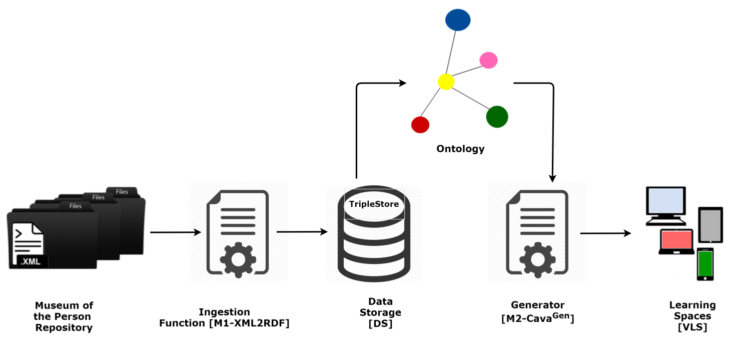
Ontology
Files
Files
Files
.XML
TripleStore
Museum of the Person Repository
Ingestion Function [M1-XML2RDF]
Data Storage [DS]
Generator [M2-Cava^{Gen}]
Learning Spaces [VLS]

Fig. 1. Proposed Architecture

an Ontology that describes the knowledge domain linking the concepts through a set of relations; the Generator (M2) to receive and interpret the requests for information, access the DS and return the answers that are combined to set up the final VLS [6] [7] (see Figure 1).

3 Data Extraction and Ontology Population: XML2RDF

The role of M1 - XML2RDF in Figure 1 is to read the annotated documents, extract and prepare the data, and store the collected information. Thus, to develop M1, it is necessary to observe the elements and structure that can appear in input documents, write a collection of production rules based on regular expressions, and use a text filter generator to derive the final program.

In this case, the input structure is a structured collection of XML and the output will be a sequence of triples (<subject, predicate and object>). In each triple (subject and object) the concepts correspond to some of the data items that are the value of the attributes of an XML element or even the element content. The relations (predicate) linking concepts can be inferred from the XML elements and their structure [8].

As discussed above, this process can be described using a set of production rules. Each production rule is a pair: on the left side, we specify the element we want to look for – regular expression (RE); The right side is a code that transforms the input data and writes the respective output.

To illustrate the implementation of this proposal, we will use the assets of the Museum of the Person (MP). The digital repository of the MP is composed of three types of documents (BI - basic information, Legend - of the respective photos, and Edited Interview).

From this repository, an ontology was built, using the CIDOC-CRM[4], FOAF[5] and DBpedia[6] standards, to store the information contained in the assets, in the

[4] In: `http://www.cidoc-crm.org/`
[5] In: `http://www.foaf-project.org/`
[6] In: `http://wiki.dbpedia.org/`

form of triples. For more information on this concrete ontology, see: `http://npmp.epl.di.uminho.pt/cidoc_foaf_db.html`.

Thus, to process the digital repository referred above, we construct a text filter to process the input data automatically, producing a triple store – XML2RDF [7][9][10]. This text filter was developed using the ANTLR (Another Tool for Language Recognition) Compiler Generator system. ANTLR generates a lexical parser that implements the desired text filter for data extraction, based on a set of regular expressions.

This text filter receives as input an XML document. After reviewing and processing it, the translator will issue a RDF description. The XML2RDF architecture is described in Figure 2. ANTLR, through the *XML2RDF.g4* grammar file, generates the compiled *XML2RDF.java* class, including the *Person.java* class, to create the desired XML2RDF processor [7][8].

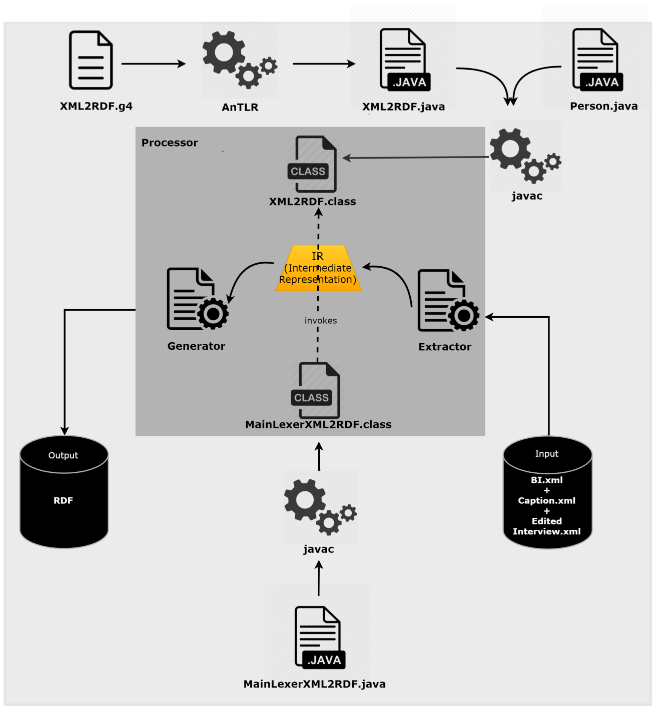

Fig. 2. Architecture of Ingestion Function [XML2RDF]

An example of an ANTLR mode is shown in Listing 1. This grammatical fragment renders an episode of General Character, when the person interviewed narrates an episode of this type.

In this case, the extractor when it finds the opening mark of the block, which corresponds to a General Character episode, activates the appropriate mode to

Listing 1. Lexer Grammar: Mode to cope with 'Episodes' in an interview

```
1  mode sMP ;
2  mode sEPISCab ;
3
4  GetEPISCarac:  [ ]*'caracter="'      -> mode(sEPISCarac) ;
5  GetEPISQuem:   [ ]*'quem="'          -> mode(sEPISQuem) ;
6  GetEPISTitulo: [ ]*'titulo="'        -> mode(sEPISTitulo) ;
7  GetEPISTermo:  [ ]*'termo="'         -> mode(sEPISTermo) ;
8  GetEPISTexto:  '>'                   -> mode(sEPISTexto) ;
9
10 mode sEPISCarac ;
11 GetCaracter: ~('"')+                 { episCarac = getText(); } ;
12 OutCaracter: '"'                     -> mode(sEPISCab) ;
... other modes (sEPISQuem, sEPISTitulo, sEPISTermo, sEPISTexto) are similar ...
```

process the contents of this block. When it finds the block closing mark, the processor exits the mode and returns to the initial mode.

The fourth initial auxiliary modes (lines 4-7) of Listing 1 contain specific rules for extracting information from label attributes. The fifth auxiliary mode (line 8) contains a specific rule for extracting the description of the General Character episode.

The generation of the RDF output file is performed by the grammatical excerpt shown in Listing 2. This grammatical fragment is composed of the rules executed at the end of the processing to print the RDF triples stored in the internal representation.

Listing 2. Lexer Grammar: Print Modes

```
if (general.size() > 0) {
   System.out.println("<rdf:Description rdf:about=\"&ecrm;General_Interviewed_"
                      +countinterview+"\">");
   System.out.println("<rdf:type rdf:resource=\"&ecrm;E55_Type\"/>");
   System.out.println("<P2_has_type rdf:resource=\"&ecrm;General\"/>");
   for (String item: general) {
       System.out.println("<P3_has_note rdf:datatype=\"&xsd;string\">"
                          +item+"</P3_has_note>");
   }
   System.out.println("</rdf:Description>");
}
```

In the next section, we detail the automatic generation of Virtual Learning Spaces (VLS) to display information extracted from the XML2RDF translator in a Web browser.

4 CaVa: automatic Generation of Virtual Learning Spaces

Next sections deal with the formal specification in CaVa^{DSL} and the main module of CaVa, called CaVa^{gen}, which is a set of processors aiming at generating the final Virtual LS [11].

4.1 CaVa$^{\mathrm{DSL}}$: specifying Virtual Learning Spaces

CaVa$^{\mathrm{DSL}}$ was designed having in mind its use by the curator of a museum, an archivist, or any other cultural institution responsible. The syntax of the language is simple but expressive, enabling the end-user to describe the exhibition rooms in an easy way. The CaVa$^{\mathrm{DSL}}$ structure is split into four major blocks which describe the main configuration, the header, the content, and the footer of the LS.

- The main configuration (*mainconfig*): specifies the LS title and main description (e.g. the text about the cultural institution). Moreover, it describes other components related to the entire LS;
- Header (*menu*): specifies the main menu of the LS. It comprises: the brand, background and foreground colors, behavior (if the menu should be fixed or it should follow the scrolling), and type of the menu items (dropdown or simple) with the label and the link;
- The content (*exhibitions*): the exhibitions' list. Each exhibition comprehends: a title, short description, and icon; additional info with title and a description; behavior (if the component of the list should stay opened (expanded) or closed (collapsed)); exhibition type (must be "permanent", "temporary", "future", or "special"); a query operator ("all": which searches for all occurrences of specified ontology concept and returns the set of result instances; "one": which searches for only one instance that matches the conditional parameter and the ontology concept. It returns the first result found.);
- The footer (*footer*): specifies an area at the bottom of the page that comprehends: images and date, company or developer name, behavior (like the header component), and style (if the footer is simple with the data mentioned or extended, having another options to specify (e.g. social networks link)).

Notice that CaVa$^{\mathrm{DSL}}$ can be extended to comprise more components, it is just needed to create new productions in the grammar (CaVa$^{\mathrm{grammar}}$) that rules the language. For the sake of space, CaVa$^{\mathrm{grammar}}$ is not exposed in this paper. To exemplify how to describe an element in CaVa$^{\mathrm{DSL}}$, Listing 3 presents a fragment of a specification to generate the main menu of the "Museum of the Person" Virtual Learning Space.

The description presented in Listing 3 specifies a main menu that has a title (brand), with foreground and background colors, a behavior (fixed at the top of the page or scrolling). Moreover, this menu has some options related to the number of items or submenus (in this case, 2): a dropdown submenu labeled "Exhibitions", which contains five sub-menus ("All", "Permanent", "Temporary", "Future", and "Special"); and a submenu (a simple one) labeled "About". For each of these menu items, a correspondent webpage is created.

To build the desired main menu, as well as the whole LS, a set of processors are necessary to analyze the CaVa$^{\mathrm{DSL}}$ specification and produce the Web page code.

Listing 3. Fragment of the VLS menu specification in CaVa^{DSL}

```
menu [
     brand: “Museum of the Person”,
     background color: crimson,
     foreground color: white,
     behavior: fixed,
     options [
             label: “Exhibitions”, dropdown [
                  dropdown label: “All”, url: “exhibitions”,
                  dropdown label: “Permanent”, url: “permanent_exhibit”,
                  dropdown label: “Temporary”, url: “temporary_exhibit”,
             ]
             label: “About”, url: “about”, extension: php,
     ]
]
```

4.2 CaVa^{gen}: generating Virtual Learning Spaces

CaVa^{gen} is a set of processors that given the right input, produce the output files (static and dynamic content) related to the final Virtual Learning Space in accordance with the CaVa^{DSL} specification. This paper describes two of those processors: CaVa Processor, the core of CaVa^{gen}; and CaVaSPARQLTriples Processor, which deals with the generation of dynamic content, i.e. the generation of SPARQL queries.

CaVa Processor transforms a CaVa^{DSL} specification into various Web program files written in different languages (HTML, PHP, JS, template engines, CSS, etc.) that, when placed all together, configure multiple Web pages, i.e., the final Virtual Learning Space. Figure 3 presents the schema of CaVa Processor.

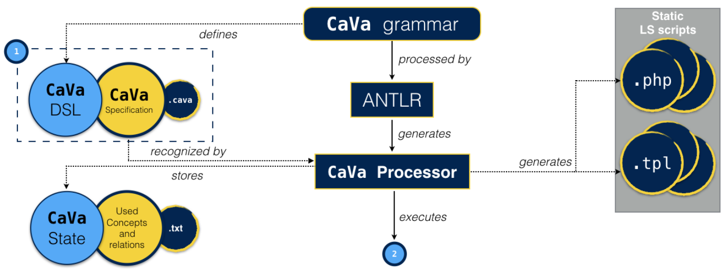

Fig. 3. CaVa Processor schema

The rectangle identified by number (1) is related to the specification file (extension “.cava”) describing a Virtual LS based on the rules of CaVa^{DSL}. It is the main input to the CaVa Processor. From the input specification, CaVa Processor generates the static content of the LS (basically the .php and .tpl (template) files). As can be seen in Figure 3, CaVa Processor is created by the Compiler Generator ANTLR taking as input the CFG CaVa^{DSL}. The *CaVa State* circle represents the state files necessary to store some configurations (in our case, used concepts and relations of the ontology) in a plain text file (.txt) to be used by CaVaSPARQLTriples Processor.

The implementation of CaVa Processor was based on ANTLR's Listeners [12]. Basically, this means that for each production of our grammar, exists a listener method that handles the recognized token and produces some output. CaVa Processor normally receives an input, recognizes it and generates PHP code (static content). So, for example, getting the Listing 3 as input, CaVa Processor, through a listener method called "enterHeader()", produces the code shown in Listing 4.

Listing 4. Generated PHP code for creating the menu according to CaVa^{DSL} specification

```
<?php
    $data = array(
        'brand'=>"Museum of the Person",
        'bgColor'=>"crimson",
        'fontColor'=>"white",
        'behaviour'=>"fixed",
        'options'=>array(
            array('label'=>"Exhibitions", 'dropdown'=>"true",
                  'dropdownListItems'=>array(
                     array('labelDropDown'=>"All",
                           'urlDropDown'=>"exhibitions"
                     ),
                     ... the 'Permanent' and 'Temporary' items are similar ...
                 ),
            ),
            array('label'=>"About", 'url'=>"about",
                  'dropdown'=>"false"
            ),
        ),
   );
   $tpl = new SMTemplate();
   $tpl->render('header', $data);
```

The code of Listing 4 is stored in a file (in this case, "header.php") and later, the content of the $data variable is passed to a template file (called "header.tpl"), which contains some placeholders to deal with $data content and render the final menu. Notice that the generation of the other files and content is similar to the process of the creation of the LS menu.

Besides the generator of static content (CaVa Processor), CaVa^{gen} contains other processor to generate dynamic content, i.e. generation of the SPARQL queries and the exhibition room files.

As already mentioned, CaVa^{DSL} allows, at the content (exhibitions) block, the specification of query operators. When a query operator is defined in an exhibition component, CaVa Processor stores the information about the statement and delegates the processing to CaVaSPARQLTriples Processor, that is the processor that knows how to handle with that information. In this stage, we are concerned with the generation of dynamic content, more precisely SPARQL queries.

To solve the automatic generation of SPARQL queries, we have used an approach that reuses well-stablished grammars (RDF and Turtle). Figure 4 presents the CaVaSPARQLTriples Processor schema that deals with this concern.

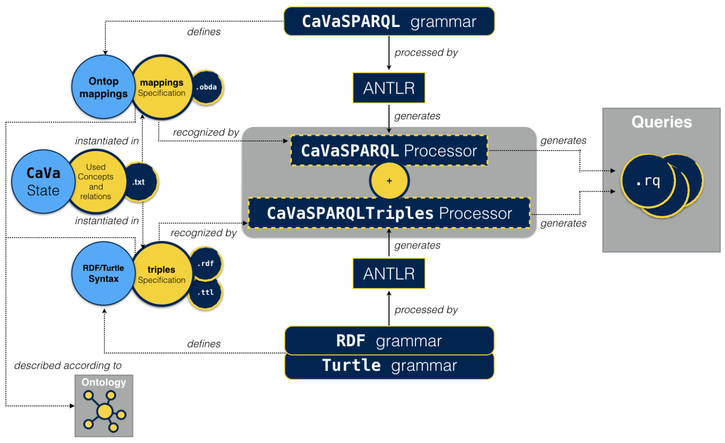

Fig. 4. CaVaSPARQLTriples Processor schema

Figure 4 shows the whole schema for the generation of SPARQL queries (".rq" files) based on used concepts and relations (specified in the "content" block of CaVa^{DSL}) of the ontology. As presented in Section 2, CaVa^{gen} receives as input the triples from the Data Storage and, based on the ontology, recognizes the concepts and relations related to the operator specified in the CaVa^{DSL} specification. Thus, CaVaSPARQLTriples Processor recognizes the triples and assembles the SPARQL query based on that set of triples that respect to a determined ontology.

Aiming at generating the dynamic content of the Virtual Learning Space, CaVaSPARQLTriples Processor recognizes the input, and based on ANTLR listeners, processes the code and produces an output that is a SPARQL query. So, from the CaVa^{DSL} specification file, recognized and processed by CaVa Processor, when a query operator is found in the exhibitions block, it means that CaVaSPARQLTriples Processor needs to access the CaVa State file (created by CaVa Processor) to get the information needed to assemble the SPARQL query.

Having the concept and relations recognized, CaVaSPARQLTriples Processor executes a sequence of actions in order to produce the right output (SPARQL query) to be used in the exhibition room file in the final Virtual LS. These actions are summarized as follows:

1. Expand each instance of the RDF file, selecting only those related to the concept specified in the CaVa^{DSL} specification;
2. Collect and store all those instances and information important to the final query (those expanded in step 1);
3. Transform each instance name in a new variable of the SPARQL WHERE clause (e.g. ?General_Interviewed_11, ?General, etc.). Also transforms each literal in a new variable of the SPARQL SELECT clause, naming each one ?p___0, ?p___1, and so on, depending on how many literals are found and important to the query;

4. Concatenate the two SPARQL clauses (SELECT + WHERE) and the prefixes needed (found in the RDF input file) in a string;
5. Create and write the SPARQL query file (.rq) containing the string's content of step 4.

Subsequently to the generation of the SPARQL query, it needs to be executed. To perform this action, we have created another processor, called Query Processor [11], which receives the ".rq" generated file as input, searches for the results in the triple store and returns the set of results found, storing it into a JavaScript Object Notation (JSON) file to be consumed by the exhibition room script file (in our case, a PHP file automatically generated by CaVa Processor) and passed to the web browser to render it based on a template engine file (.tpl).

This configures a complete exhibition room inside a Virtual Learning Space, where the user can navigate over the instances and the concepts of the ontology according to the curator exposure. To conclude, this approach was used to implement all the VLS of the Museum of Person. For the sake of space, to see some screenshots and more details about the final VLS, visit `http://www4.di.uminho.pt/~gepl/paper-HD`.

5 Conclusion

The description of Virtual Learning Spaces in a simple language, directed to a specific user, boosts the generation of virtual learning environments and empowers the responsible of the cultural institution to focus only on the content exposure (the structure of the exhibition rooms), arranging this content in a way that the user deems most appropriate. This eliminates any problem regarding the learning of various general-purpose programming languages by the person in charge of the cultural institution, that is, learning the CaVa^{DSL} language it is enough to specify and generate the desired Virtual Learning Space. Both the DSL and the software artifacts to deal with it were introduced and explained along the paper.

It is important to emphasize that we do not know about another similar system that deserves to be studied or compared.

Many fresh ideas arise while developing such a project. However, as future work the most important is to conduct experiments to assess the effectiveness and usability of our proposal, as well as, to apply the approach to different scenarios and case studies. For this reason, performance tests were not included in this study.

Acknowledgements: This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013. The work of Ricardo G. Martini is supported by CNPq, grant 201772/2014-0.

References

1. Schweibenz, W.: The development of virtual museums. In: Virtual Museums. Volume 57(3). ICOM (2004)

2. Almeida, J.J., Rocha, J.G., Henriques, P.R., Moreira, S., Simões, A.: Museu da Pessoa – arquitectura. In: Encontro Nacional da Associação de Bibliotecários, Arquivista e Documentalistas, ABAD'01, BAD (2001)
3. Simões, A., Almeida, J.J.: Histórias de Vida + Processamento Estrutural = Museu da Pessoa. In: XATA 2003 — XML: Aplicações e Tecnologias Associadas, Braga, Portugal, UM (2003) 16
4. Martini, R.G., Araújo, C., Almeida, J.J., Henriques, P.R. In: OntoMP, An Ontology to Build the Museum of the Person. Springer International Publishing, Cham (2016) 653–661
5. Martini, R.G., Librelotto, G.R., Henriques, P.R.: Formal description and automatic generation of learning spaces based on ontologies. Procedia Computer Science **96** (2016) 235 – 244 Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 20th International Conference KES-2016.
6. Araújo, C., Henriques, P.R., Martini, R.G., Almeida, J.J.: Architectural approaches to build the museum of the person. In: 2016 11th Iberian Conference on Information Systems and Technologies (CISTI). (2016) 1–6
7. Araújo, C.: Building the Museum of the Person Based on a combined CIDOC-CRM/ FOAF/ DBpedia Ontology. MSc Thesis, Universidade do Minho (2016)
8. Araújo, C., Henriques, P.R., Martini, R.G.: Automatizing ontology population to drive the navigation on virtual learning spaces. In: 2017 12th Iberian Conference on Information Systems and Technologies (CISTI). (2017) 1–6
9. Araújo, C., Martini, R., Henriques, P.R., Almeida, J.J.: Building the Museum of the Person from RDF Triples and SPARQL. Communications and Innovations Gazette (ComInG) **1** (2016) 1–14
10. Araújo, C., Martini, R.G., Henriques, P.R., Almeida, J.J. In: Annotated Documents and Expanded CIDOC-CRM Ontology in the Automatic Construction of a Virtual Museum. Springer International Publishing, Cham (2018) 91–110
11. Martini, R.G., Henriques, P.R.: Automatic generation of virtual learning spaces driven by cavadsl: An experience report. In: Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences. GPCE 2017, New York, NY, USA, ACM (2017) 233–245
12. Parr, T.: The Definitive ANTLR 4 Reference. 2nd edn. Pragmatic Bookshelf (2013)