UMinho | 2018

Ricardo Giuliani Martini

Formal Description and Automatic Generation of Learning Spaces based on Ontologies

Universidade do Minho
Escola de Engenharia

Ricardo Giuliani Martini

# Formal Description and Automatic Generation of Learning Spaces based on Ontologies

March 2018

**Universidade do Minho**
Escola de Engenharia

Ricardo Giuliani Martini

# Formal Description and Automatic Generation of Learning Spaces based on Ontologies

Doctoral Thesis
Doctoral Program on Informatics

Thesis supervised by
**Professor Dr. Pedro Manuel Rangel Santos Henriques**
**Professor Dr. Giovani Rubert Librelotto**

March 2018

DECLARAÇÃO

Nome: Ricardo Giuliani Martini

Endereço electrónico: giulianimartini@gmail.com Telefone: +351 966 120 909

Número do Bilhete de Identidade: 14823V3F1

Título da tese: Formal Description and Automatic Generation of Learning Spaces based on Ontologies

Orientador(es):

Professor Doutor Pedro Manuel Rangel Santos Henriques e Professor Doutor Giovani Rubert Librelotto

Ano de conclusão: 2018

Designação do Doutoramento:

Programa Doutoral em Informática (PDINF)

Universidade do Minho, 29/03/2018

Assinatura: [illegible]

# Acknowledgments

First of all, I would like to thank my family, especially my mother Nilva, that raised me and supported me so that I could pursue my academic goals, always believing in me. Mom, thank you very much for being there with me when I needed you, even more now. I missed you so much ever since I left home, but you can be sure that all your efforts have helped me to be where I am today. Thank you very much!

I would also like to thank my brother, Alexandre, who has always been by my side. In addition to making the role of brother in the best possible way, always, since graduation, has helped me to solve my academic issues. I wish one day I can repay all you've done for me. You're a real brother and I thank you every day for it! This thesis would not have been finished without your indispensable help. In the most critical moments, you were always by my side.

To my supervisor and friend, Professor Pedro Rangel Henriques, I would like to thank you immensely for what you have done for me since I arrived in Portugal. From the moment I met you, you were like a father to me. Professor Pedro, thank you very much for everything, for the knowledge transmitted, for the friendship, and for the concern you have always shown to me. Be assured that I will mirror myself, both professionally and personally, in you. Thank you very much.

To Cristiana Araújo, with whom I had the pleasure of working with. There were many funny moments, conversations and many, many cafes. Cristiana, thanks for your help. Those were good years of working together.

I would also like to thank Professors José João Almeida and Maria João Varanda, and the Archivist Mónica Guimarães for the collaborative publications. Thank you.

To the Department of Informatics, for the reception received during these 4 years, especially the employees Helena, Goretti, and Conceição.

To my co-supervisor, Giovani Rubert Librelotto, thank you for the effort you have made so that I could apply for a PhD in the University of Minho, Portugal. Your help was crucial for me to get where I am. Thank you.

To Luciana Probst, who was always by my side, even though she was geographically far away. Lu, thank you very much for the love you have shown in all these years, supporting me in my academic and professional career. For understanding my absence for 4 years of our lives. Be assured that this achieved goal is not only my merit, but yours too, because there were days and nights that you made me feel better on the worst days. Thank you so much for being such a special person in my life!

To my friends Lucas Gomes, Carlos Carvalho, Lip, Jonas Gassen, Renato Preigschadt de Azevedo, Isabela Finamor, Leandro Oliveira Freitas, Greice Zanini, Bruno Mozzaquatro, Karine Fontana, Walter Priesnitz Filho, Mariane Camargo Priesnitz, Ricardo Ravanello, Tatiana Rehbein, João Marco, Alice Balbé, Luis Miranda Ramos, Maiara Nascimento, Rafael Teodósio Pereira and Andriza Saraiva, for the funny moments, and trips around Europe. Thank you!

Last but not least, I would like to thank my favorite football team, Grêmio FBPA, by winning so many titles over the years I was away from my beloved Rio Grande do Sul! There were many commemorative dates. Thank you Tricolor, Three times Champion of America!!!

# Abstract

A good Learning Space (LS) should convey pertinent information to the visitors at the most adequate time and location to favor their knowledge acquisition. This statement justifies the relevance of virtual Learning Spaces.

Considering the consolidation of the Internet and the improvement of the interaction, searching, and learning mechanisms, this work proposes a generic architecture, called CaVa, to create Virtual Learning Spaces building upon cultural institution documents. More precisely, the proposal is to automatically generate ontology-based virtual learning environments from document repositories.

Thus, to impart relevant learning materials to the virtual LS, this proposal is based on using ontologies to represent the fundamental concepts and semantic relations in a user- and machine-understandable format. These concepts together with the data (extracted from the real documents) stored in a digital repository are displayed in a web-based LS that enables the visitors to use the available features and tools to learn about a specific domain.

According to the approach here discussed, each desired virtual LS must be specified rigorously through a Domain-Specific Language (DSL), called CaVa$^{DSL}$, designed and implemented in this work. Furthermore, a set of processors (generators) was developed. These generators have the duty, receiving a CaVa$^{DSL}$ specification as input, of transforming it into several web scripts to be recognized and rendered by a web browser, producing the final virtual LS.

Aiming at validating the proposed architecture, three real case studies – (1) Emigration Documents belonging to Fafe's Archive; (2) The prosopographical repository of the *Fasti Ecclesiae Portugaliae* project; and (3) Collection of life stories of the Museum of the Person – were used. These real scenarios are actually relevant as they promote the digital preservation and dissemination of Cultural Heritage, contributing to human welfare.

# Resumo

Um bom Espaço de Aprendizagem (LS – Learning Space) deve transmitir informações pertinentes aos visitantes no horário e local mais adequados para favorecer a aquisição de conhecimento. Esta afirmação justifica a relevância dos Espaços virtuais de Aprendizagem.

Considerando a consolidação da Internet e o aprimoramento dos mecanismos de interação, busca e aprendizagem, este trabalho propõe uma arquitetura genérica, denominada CaVa, para a criação de Espaços virtuais de Aprendizagem baseados em documentos de instituições culturais. Mais precisamente, a proposta é gerar automaticamente ambientes de aprendizagem virtual baseados em ontologias a partir de repositórios de documentos.

Assim, para transmitir materiais de aprendizagem relevantes para o LS virtual, esta proposta é baseada no uso de ontologias para representar os conceitos fundamentais e as relações semânticas em um formato compreensível pelo usuário e pela máquina. Esses conceitos, juntamente com os dados (extraídos dos documentos reais) armazenados em um repositório digital, são exibidos em um LS baseado na web que permite aos visitantes usarem os recursos e ferramentas disponíveis para aprenderem sobre um domínio específico.

Cada LS virtual desejado deve ser especificado rigorosamente por meio de uma Linguagem de Domínio Específico (DSL), chamada CaVa$^{\mathrm{DSL}}$, projetada e implementada neste trabalho. Além disso, um conjunto de processadores (geradores) foi desenvolvido. Esses geradores têm o dever de receber uma especificação CaVa$^{\mathrm{DSL}}$ como entrada e transformá-la em diversos web scripts para serem reconhecidos e renderizados por um navegador, produzindo o LS virtual final. Visando validar a arquitetura proposta, três estudos de caso reais foram usados. Esses cenários reais são realmente relevantes, pois promovem a preservação digital e a disseminação do Patrimônio Cultural, contribuindo para o bem-estar humano.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| ABS | Anti-lock Braking System |
| ANTLR | ANother Tool for Language Recognition |
| API | Application Programming Interface |
| AR | Augmented Reality |
| AJAX | Asynchronous Javascript and XML |
| CaVa | Criação de Ambientes Virtuais de Aprendizagem |
| CFG | Context Free Grammar |
| CH | Cultural Heritage |
| COTS | Commercial Off-The-Shelf |
| CRM | Conceptual Reference Model |
| CRUD | Create, Read, Update, and Delete |
| CSS | Cascading Style Sheets |
| CUP | Construction of Useful Parsers |
| DAO | Data Access Object |
| DC | Dublin Core |
| DDL | Data Definition Language |
| DIS | Document (Data) Ingestion System |
| DSL | Domain Specific Language |
| DTD | Document Type Definition |
| EBA | Emergency Braking Assistance |
| ER | Entity-Relationship |
| FEP | Fasti Ecclesiae Portugaliae |
| FOAF | Friend of a Friend |
| HP | Horsepower |
| HTML | HyperText Markup Language |
| ICOM | International Council of Museums |
| ICH | Intangible Cultural Heritage |
| ICT | Information and Communication Technologies |
| IFML | Interaction Flow Modeling Language |
| IR | Intermediate Representation |

| | |
|---|---|
| IRI | Internationalized Resource Identifier |
| ISO | International Organization for Standardization |
| JavaCC | Java Compiler Compiler |
| JDBC | Java Database Connectivity |
| JS | JavaScript |
| JSON | JavaScript Object Notation |
| LS | Learning Space |
| MP | Museum of the Person |
| MVC | Model View Controller |
| MVIF | Museu Virtual Interativo da Fotografia |
| MPD | Museu da Pessoa Dataset |
| OBDA | Ontology-Based Data Access |
| OCR | Optical Character Recognition |
| OWL | Ontology Web Language |
| PATHS | Personalised Access to Cultural Heritage Spaces |
| PHP | PHP: Hypertext Preprocessor |
| PK | Primary Key |
| R2RML | RDB to RDF Mapping Language |
| RDF | Resource Description Framework |
| RDFs | RDF Schema |
| SGPE | Sistema Gerenciador de Processos de Emigração |
| SKOS | Simple Knowledge Organization System |
| SPARQL | Simple Protocol and RDF Query Language |
| SQL | Structured Query Language |
| SQWRL | Semantic Query-Enhanced Web Rule Language |
| THB | Turbo High Pressure |
| UI | User Interface |
| UML | Unified Modeling Language |
| UNESCO | United Nations Economic, Social and Cultural Organization |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| VR | Virtual Reality |
| XML | eXtensible Markup Language |
| XSD | XML Schema Definition |

| | |
|---|---|
| YACC | Yet Another Compiler Compiler |
| Yii | Yes It Is! |

# Chapter 1

# Introduction

With the popularization of the Internet, its use in information access became common anywhere, to anyone and in many computing devices, such as smartphones, tablets, personal computers, etc. Moreover, many ways to store data in digital format emerged. These storage formats help to improve the accessibility to the information originally kept in physical documents belonging to museums, libraries or similar institutions. This storage is held in repositories of digital objects. Therefore, having information associated with digital objects on the Internet, they can be distributed and viewed through sites which possess features for viewing, interacting and navigating over these objects.

This context created the opportunity to evolve traditional exhibition spaces, as reading and show rooms, to virtual spaces on the web in order to enable new learning approaches.

Traditional Learning Spaces (LS) are physical locations, normally within schools and universities, exposing objects with information (whether material or immaterial things), arranging them in order to convey a message to the visitor of the Learning Space [Brown, 2005], [Goos, 2006]. These spaces generally contain groups of people (usually students) debating about a specific subject and someone (usually a professor) who leads the debate, organizing behaviors through formal methods of education, to impart knowledge to the group.

However, most people's knowledge is not acquired in formal methods of education and learning in a classroom, for example, but during their leisure time outside the classroom, using their laptop, smartphone or any device to socialize with other people, as well as snack bars at breakfast, travelling on a train, in moments of leisure with their family, in a simple walk and also in visits to museums, libraries and the like [Lomas and Oblinger, 2006]. Therefore, any physical space that features knowledge sharing can be considered a Learning Space.

In short, the previously cited features allow a range of people (not only students) to use these Learning Spaces to generate and acquire knowledge. Thus, the term e-Learning should not be applied in this context to avoid misunderstandings, because it is usually used to describe LS just for students enrolled in distance education, which is not our purpose in this thesis.

Following this thought, a Learning Space is like a website where the information is arranged in such a way that the visitor learns with it.

Virtual Learning Spaces, like virtual classrooms, virtual seminars, and virtual museums, improve learning experience by supporting learning at leisure time, i.e., at flexible locations and time.

To build these virtual Learning Spaces, data about the desired domain should be stored in a way that enables it to be processed and displayed to the learner at a later time in the best possible way.

So, in this work, it is proposed an architecture to automatically build virtual Learning Spaces formally specified in a specially tailored language that resorts to a domain ontology.

Aiming at validating the proposed architecture, three real case studies – (1) Emigration Documents belonging to Fafe's Archive; (2) The prosopographical repository of the *Fasti Eclesiae Portugaliae* project; and (3) Collection of life stories of the Museum of the Person – are presented and described step by step according to the proposed approach.

These real scenarios are actually relevant as they promote the digital preservation of our Cultural Heritage, contributing to human welfare. Storing data collected from archived documents in a computer is a secure way of preserving information. As said above, it opens many possibilities of disseminating the preserved knowledge.

## 1.1 Motivation

This PhD proposal is a challenging idea different from all similar projects found in the literature. It contributes to enrich the learning environment, as stated at [UNESCO, 1997].

It is important to emphasize that, it is not another work in the area of e-Learning (actually this work is not concerned with distance education targeted to closed sets of students enrolled in some course) and it is not a new approach in the area of creating digital versions of traditional museums.

Summing it up, this work is motivated by an unpublished idea of creating virtual Learning Spaces to impart knowledge of Cultural Heritage.

## 1.2 Objectives

This doctoral project has the goal of automating the creation of web-based virtual Learning Spaces using an ontology and a Domain-Specific Language (DSL). The ontology serves two purposes: first, to give semantics to the digital object repository; and second; to describe the information that must be displayed. The DSL allows a detailed description of the desired space.

The specific goals of this PhD research are listed below:

1. Create a formalism (DSL) to rigorously describe virtual Learning Spaces taking into account the domain ontology;

2. Develop a mechanism to automatically generate the virtual LS from its formal specification.

To achieve those objectives, the following tasks were performed:

1. A comprehensive study about each relevant subject of the thesis;
2. A deep research regarding the State of the Art, always seeking to study related work about the project in question;
3. The design of an architecture that reflects the study addressed;
4. The implementation of all the components included in the proposed architecture;
5. The validation of the framework implemented using three case studies (Emigration Documents belonging to Fafe's Archive; The prosopographical repository of the *Fasti Eclesiae Portugaliae* project; and The collection of life stories of the Museum of the Person).

## 1.3 Research Hypothesis

The PhD work here reported started from the subsequent Research Hypothesis: *it is feasible to automatically create virtual Learning Spaces, as web pages, based on an ontology — that describes an institutional information repository — and on a DSL specification — that defines which concepts should be exhibited and how they should be placed in the final virtual Learning Space.*

## 1.4 Document Organization

In order to introduce and discuss this PhD work, this document is organized in three different parts.

Part I is related to the State of the Art. Chapter 2 introduces and conceptualizes the Cultural Heritage term as well as the kinds of Cultural Heritage that exist and the importance of preserving it.

Chapter 3 presents what an Ontology is and its formal definition as well as its components. Also, the advantages of using ontologies are presented. Finally, a brief discussion about the Cultural Heritage ontologies is made, presenting the CIDOC Conceptual Reference Model (CRM) ontology specific for that domain.

Domain-Specific Languages (DSL) and their definitions are described in Chapter 4. The life cycle of DSLs and their advantages and disadvantages are pointed. Also in this chapter, DSLs to generate Web applications are presented as a part of the related work. To conclude the chapter, a relation between ontologies and DSLs is made.

To finish the State of the Art, Chapter 5 presents a bibliographic survey about Learning Spaces, more focused in the virtual ones. This chapter also outlines a comparison between traditional and virtual Learning Environments[1], taking into consideration a set of parameters found in the literature. Also, some advantages and disadvantages of these two kinds of LS (traditional and virtual) are characterized. To complete this study, a section describing projects related to the entire generation of virtual Learning Spaces is presented.

After concluding the literature review, Part II presents the proposal. Chapter 6 describes it, characterizing the components and the whole architecture in a general way.

Aiming at transmitting knowledge about a particular domain to the end-users through the virtual Learning Spaces, Chapter 7 presents the ontology component of CaVa. This element must describe and relate semantic concepts of a specific domain.

Chapter 8 outlines the initial phase of the project, which is the population of the domain repository, aiming at having all the data prepared to be used

[1] In this thesis, the *virtual Learning Environment* term is employed as a synonym of the*virtual Learning Space* term.

in the generation of the virtual Learning Spaces.

The CaVa$^{\text{DSL}}$ language is discussed in Chapter 9. This chapter explains the details about CaVa$^{\text{DSL}}$, as well as its syntax and components.

Chapter 10 details the heart, or core of the CaVa architecture. The main components of the set of processors (generators) developed to reach the final virtual LS are described in this chapter.

Chapter 11 describes the configuration and the script files generated by CaVa$^{\text{gen}}$, which aim to render the final virtual LS by the web browser.

Regarding the validation of the proposed architecture (CaVa), Part III describes all the three case studies in detail. Chapter 12 outlines the Emigration phenomena in Portugal through the emigration documents belonging to Fafe's Archive. Chapter 13 presents the case study related to the prosopographical repository of the *Fasti Eclesiae Portugaliae* project. The third case study, concerning the collection of life stories of the Museum of the Person, is described in Chapter 14.

Chapter 15 concludes this thesis. This chapter summarizes the work done, presents all the contributions, and finally, gives the directions for future research.

Figure 1.1 is a roadmap of the thesis, presenting the best options path based on the subject of each part of the thesis.

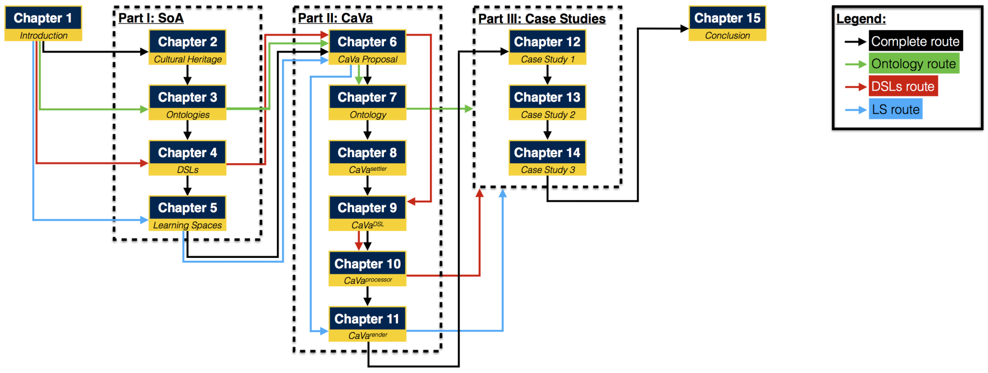


Figure 1.1: Thesis roadmap based on the subjects

# Part I

# State of the Art

# Chapter 2

# Cultural Heritage

This chapter introduces the topic Cultural Heritage from different viewpoints. The importance of cultural preservation and ways of preserving are discussed.

The definitions presented in this chapter are selected from different organizations in distinct periods in order to provide basic reference for the whole subject.

## 2.1 Definition of Cultural Heritage

Before defining Cultural Heritage, it is important to have the Curator term defined. According to the European Frame of reference for museum professions, a Curator "is the responsible for the collections in his/her charge" [Ruge, 2008].

Among the duties that the Curator needs to perform, he/she shall "define and conduct research projects, attend to the circulation of information and documentary materials on collections and exhibitions". In addition, the Curator "contributes to the designing and organizing of permanent and temporary exhibitions, publications and activities for the public" [Ruge, 2008]. For this work, the Curator is the responsible for the organization and specification of any kind of exhibition rooms.

To understand what Cultural Heritage is, a good approach is to define the two concepts (culture and heritage) separately.

In 1871, Edward Burnett Tylor, in his book entitled *Primitive Culture: Researches Into the Development of Mythology, Philosophy, Religion, Art, and Custom*, defined culture as "a complex whole which includes knowledge, belief, art, morals, law, custom, and any other capabilities and habits acquired by man as a member of society." [Tylor, 1871].

UNESCO defines culture "as the set of distinctive spiritual, material, intellectual and emotional features of society or a social group, that encompasses, not only art and literature, but lifestyles, ways of living together, value systems, traditions and beliefs." [Stenou and Unesco, 2002].

The term heritage can be understood as "our legacy from the past, what we live with today, and what we pass on to future generations." [Centre, 2005].

From the definition of culture and heritage, the Cultural Heritage term can be conceptualized, which is, according to UNESCO, the legacy of physical artifacts and intangible attributes of a group or society that are inherited from past generations, maintained in the present and bestowed for the benefit of future generations [UNESCO, 1989].

According to [Waterton and Watson, 2013], Cultural Heritage is not limited to material manifestations, such as monuments and objects that have been preserved over time. This notion also encompasses living expressions and the traditions that countless groups and communities worldwide have inherited from their ancestors and transmit to their descendants, in most cases orally.

There is a succinct definition by John Feather about Cultural Heritage that summarizes the concepts mentioned previously: "it is a human creation intended to inform." [Gorman and Shep, 2006] [Feather, 2006].

The definition of culture for this work is what the people live, encompassing the daily life, traditions, attitudes, and norms of the society. Culture changes according to people experiences, i.e., what each person learns.

The heritage is a patrimony that does not change, because it is what people inherit, i.e., what come from the past.

However, understanding the concepts of culture and heritage separately enables us to see that they are two concepts that complement each other, because it is through heritage of a person that can be passed the culture to another.

The culture of a person comes since birth and goes changing over the lifetime with new knowledge, often by Cultural Heritage.

## 2.2 Types of Cultural Heritage

There are many kinds of Cultural Heritage. In this section, Cultural Heritage is described in accordance with UNESCO, that defines two types – tangible[1] and intangible[2]:

- **Tangible**: includes buildings and historic places, monuments, artifacts, etc., which are considered worthy of preservation for the future. This kind of Cultural Heritage covers objects significant to the archaeology, architecture, science or technology of a specific culture;
- **Intangible**: means the practices, representations, expressions, knowledge, skills. This kind of Cultural Heritage covers oral traditions and expressions, performing arts, social practices, rituals and festive events, knowledge and practices concerning nature and the universe and traditional craftsmanship.

Summing it up, the tangible Cultural Heritage is palpable (material things) and can be movable or immovable, and the intangible Cultural Heritage (ICH) is impalpable (immaterial things).

1 Available at:
`http://www.unesco.org/new/en/cairo/culture/tangible-cultural-heritage/`

2 Available at:
`http://www.unesco.org/culture/ich/index.php?lg=en&pg=00022#art2`

These two concepts of Cultural Heritage (tangible and intangible) are always connected. According to Makio Matsuzono, director of National Museum of Ethinology in Japan, archives, museums, libraries and other physical institutions have been recognized as a place for preserving and exhibiting tangible objects. However, such objects can never be produced without involving various intangible cultural resources. If the knowledge is lost, little can be made and learned from an object that remains, but if an object is lost or has a limited period of use, people can reproduce the object if they have kept the necessary knowledge [Matsuzono, 2004].

The tangible is therefore always embedded in the intangible. It is unreasonable to divide the two forms of heritage [Matsuzono, 2004].

Because of this, the Cultural Heritage shall be preserved. Section 2.3 explains the importance of preserving Cultural Heritage. To the objectives of this thesis, the intangible Cultural Heritage shall receive more attention.

## 2.3 Importance of Preserving Cultural Heritage

As mentioned earlier, Cultural Heritage deals with the importance of the people and the patrimony that have to be passed down to the following generations. Because of this, the objects, the artifacts, among other types of Cultural Heritage shall be preserved.

In some regions, due to the rapid social changes, the intangible Cultural Heritage has been losing its original forms, and in many cases the heritage has been in danger of extinction. In view of this situation, prompt action is needed to preserve the intangible Cultural Heritage and to encourage such activities. Sharing information and having discussions to find effective ways and means to preserve and promote intangible Cultural Heritage as well as make concrete programmes in this field are considered to be urgent matters [ACCU, 1998].

The preservation of Cultural Heritage demonstrates recognition of the necessity of the past and of the things that tell the people's story. Preserved objects also validate memories; and the reality of the object draws people in and gives them a literal way of touching the past[3].

In accordance with UNESCO, it is the responsibility of people to preserve, transmit and leave this legacy to succeeding generations [UNESCO, 1989]. Work to safeguard and promote Cultural Heritage is important to bring together people, communities and societies [UNESCO, 2013].

Libraries, archives, and museums hold many different collections in a variety of media, presenting a vast body of knowledge accumulated over the institutions' history, and the mission of these institutions is to make their collections accessible to intended users [Ekwelem et al., 2011].

Therefore, access is just as important as preservation of ICH [Ivey, 2004]. It means that to get knowledge from something that was preserved it is necessary to have access to the source. An example of this can be something related to a museum. Imagine how something can be known about the masterpiece called *Mona Lisa* by Leonardo da Vinci without having access to the place (Louvre Museum) where it is.

### 2.3.1 Ways of Preserving Cultural Heritage

The importance of Cultural Heritage makes it important to preserve it in the best way. The way that cultural inheritance shall be preserved depends on its type. Special care must be taken since each type of cultural patrimony has its particularities.

Taking this into account, some ways can be taken for the tangible Cultural Heritage and others to the intangible. The tangible type includes buildings, historic places, monuments, among others; while intangible includes representations, knowledge, rituals, dancing, among other cultural patrimonies.

[3] Available at:
`http://www.unesco.org/new/en/cairo/culture/tangible-cultural-heritage/`

To exemplify, it can be asked: can a ritual or dance be preserved in a digital form? Yes, being recorded in a DVD or stored in some video format! but is this the best way to preserve it? Depends on how it will be used.

As aforementioned in Section 2.3, the importance of preserving the Cultural Heritage brings the necessity to create social programmes to try to circumvent the preservation problem [ACCU, 1998].

Concerning this PhD project, working with the intangible type of Cultural Heritage and with the goal to create virtual Learning Spaces to disseminate knowledge, the digital way is the best to preserve the cultural inheritance.

## 2.4 Summary

This chapter conceptualized different viewpoints of Cultural Heritage. Some concepts were presented, defining Cultural Heritage as two separated key concepts that work together, because to pass culture to another person, the heritage – patrimony, history – is needed.

Besides, this chapter described the types of Cultural Heritage, defining the palpable (tangible) and the immaterial (intangible) kinds and the crucial importance of why these two concepts should always be connected.

As Cultural Heritage is an important concept about the people and the patrimony that should be passed on, its preservation effect was also discussed, showing the ways to preserve it.

The next chapter presents the ontologies, a formal description of concepts related to a specific domain. So, its definitions, types of representation, components, advantages, and domain ontologies for Cultural Heritage context are presented.

# Chapter 3

# Ontologies

According to [Gruber, 1993], [Fensel, 2000] and [Gava and Menezes, 2003], the computer science area defines ontology as a formal and explicit specification of a shared conceptualization, where according to [Studer et al., 1998], conceptualization refers to an abstract model of some phenomenon in the world which identifies relevant concepts of the phenomenon itself; formal refers to the fact of the ontology be understood by the machine; and shared gives the notion that an ontology gets the presented knowledge not only by a single individual but by a group.

As stated in [Noy and Mcguinness, 2001], an ontology is a formal explicit description of concepts about a domain (sometimes called classes), properties of each concept describing the features and attributes of the concept (sometimes called roles or slots), and restrictions on slots (sometimes called role restrictions or facets).

The use of ontologies in computing has already been studied for several years. The importance of their use is the ability to represent hierarchies of object classes (taxonomies) and their relationships [Librelotto et al., 2008a].

Ontologies differ from other data models by their concern with concepts and their relationships, in which the semantics of these relationships is applied evenly [Librelotto et al., 2008b].

According to [Noy and Mcguinness, 2001], an ontology defines a common vocabulary for researchers who need to share information in a domain including machine understandable definitions of basic concepts in the domain and relations among them. Some examples of well-known ontologies are: FOAF[1], a project (ontology) faced to linking people and information using the web; DBPedia[2], a project (ontology) devoted in extract structured content from Wikipedia[3] information; and The Dublin Core (DC)[4] project, which is an ontology for describing generic metadata.

There are some reasons to develop and use an ontology. They can be summarized in five items [Noy and Mcguinness, 2001]:

- To share common understanding of the structure of information among people or software agents;
- To enable reuse of domain knowledge;
- To make domain assumptions explicit;
- To separate domain knowledge from the operational knowledge;
- To analyze domain knowledge.

The first item on the list above is one of the most common goals in developing ontologies [Gruber, 1993]. Suppose different web sites about a same specific domain. If the web sites share and publish the same ontology, with the same concepts and classifications (taxonomy), then the computer agents can educe and put together the information from these different web sites. The agents can use this aggregated information to answer user queries, for example [Noy and Mcguinness, 2001].

One of main reasons behind the ontology research is the second item on the list above. The reuse of a domain knowledge serves basically to not reinvent

[1] `http://xmlns.com/foaf/spec/`
[2] `http://wiki.dbpedia.org/`
[3] `https://www.wikipedia.org/`
[4] `http://dublincore.org/`

the wheel, i.e., before to create an ontology, a good thing to do is to search if an ontology about the same domain is already done, published and available to its reuse.

Taken the reuse of an ontology into account, the domain of this work, which is the Cultural Heritage presented in Chapter 2, need to be described in detail. Thus, Section 3.4 presents ontologies for Cultural Heritage, aiming at describing how ontologies could be involved to enhance information management and characteristics of ontologies for the Cultural Heritage domain. Namely, the CIDOC Conceptual Reference Model (CIDOC-CRM) ontology will be presented.

## 3.1 Ontology Representation

In order to define and use an ontology, it is necessary a concrete representation. There are a variety of languages which can be used for this purpose. They vary in terms of expressiveness. Some of these languages will be described in this section.

According to [McGuinness and Harmelen, 2004], an ontology can be represented in:

- XML (eXtensible Markup Language), which provides a syntax to structured documents, but has no semantic restriction;
- XML Schema, that is a description to restrict the XML documents structure; element data types are also defined;
- RDF[5] (Resource Description Framework), that is a standard model for data interchange on the web. This model has objects and relations between them and provides a simple semantic. Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications;

[5] `www.w3.org/RDF`

- RDF Schema (RDFs), that provides a data modelling vocabulary for RDF data. This vocabulary describes properties and classes[6];
- OWL (Web Ontology Language)[7], which adds more vocabulary for describing properties and classes, for example, relations between classes (like disjunction), cardinality, equality, among others.
- SKOS (Simple Knowledge Organization System)[8], which provides a standard, low-cost migration path for porting existing knowledge organization systems to the Semantic Web. SKOS also provides a lightweight, intuitive language for developing and sharing new knowledge organization systems. It may be used on its own, or in combination with formal knowledge representation languages such as OWL.

Describing a well-founded ontology by directly manipulating the language (OWL, RDF, etc.) is not a simple task, but there are many tools (free and non-free) to help and to assist the developers to create and manipulate ontologies. Some of these tools are Protégé[9], TopBraid Composer[10], Neon[11], Apollo[12], IsaViz[13], among others.

## 3.2 Components of an Ontology

As stated in [Smith et al., 2004], to describe ontologies, some elements are needed. The basic elements are classes (concepts), individuals (instances), properties and relations. Another component of an ontology can be the axioms.

[6] `www.w3.org/TR/rdf-schema/#ch_introduction`
[7] `www.w3.org/OWL/`
[8] `https://www.w3.org/TR/skos-reference/`
[9] Available at: `http://protege.stanford.edu/`
[10] Available at: `http://www.topquadrant.com/tools/IDE-topbraid-composer-maestro-edition/`
[11] Available at: `http://neon-toolkit.org/wiki/Main_Page`
[12] Available at: `http://apollo.open.ac.uk/`
[13] Available at: `http://www.w3.org/2001/11/IsaViz/`

Some authors define formally ontologies as a set of tuples [Nguyen et al., 2010], [Inukai et al., 2007] and [Serra and Girardi, 2011]. These formal definitions are more common in the area of Computer Science.

In [Nguyen et al., 2010], an ontology $\boldsymbol{O}$ as a 5-tuple $\boldsymbol{O = (T, I, \preccurlyeq, conf, B)}$ is defined, where:

1. $\boldsymbol{T}$ is a set of types, i.e., $T = T_C \cup T_R \cup T_{MR}$ with $T_C$ being the set of *concept types*, $T_R$ the set of *relation types*, and $T_{MR}$ the set of meta-relation types. In short, a concept type is a class of objects (entity), a relation type is a predicate on concept types and a meta-relation type is a predicate on relation types and concept types;

2. $\boldsymbol{I}$ is a set of individuals, or instances of a concept, relation and meta-relation types in $\boldsymbol{T}$, i.e., $I = I_C \cup I_R \cup I_{MR}$ with $I_C$ being the set of *instances* of concept types, $I_R$ the set of *instances* of relation types, and $I_{MR}$ the set of *instances* of meta-relation types. An *individual* may be a person, animal, object, or situation. This means that it is an *instance* of a class or type;

3. $\preccurlyeq$ is the *semantic subsumption relation* in $\boldsymbol{T}$, i.e., $\preccurlyeq$ is a subset of $(T_C \times T_C) \cup (T_R \times T_R) \cup (T_{MR} \times T_{MR})$, representing the specialization relationship (inheritance) between two concept types, two relation types or two meta-relation types. For example, *Elephant* $\preccurlyeq$ *Animal* means that *Animal* is a generalization of *Elephant*, or *Elephant* is a specialization of *Animal*;

4. ***conf*** is the *conformity function*, defining for each individual in $\boldsymbol{I}$, the infimum of all concept, relation or meta-relation types that could represent that individual. For example, *conf(Jumbo)* = *Elephant* as *Elephant* is the infimum of all concept types such as *LivingEntity*, *Animal*, *Mammal*, etc. that could represent *Jumbo*. For relation or meta-relation, *conf* could be, for instance, *conf(isSonOf(Tony, Mickael))* = *isSonOf(Person, Man)*;

5. ***B*** is the *canonical (model) basis function*, defining the *usage model* or *usage pattern* of a relation or meta-relation type. For example, *isDaughterOf(Female, Person)* is a relation type linking two arguments of which the first is the daughter and the second is the parent.

Further, [Serra and Girardi, 2011] defines an ontology ***O*** as a 6-tuple ***O = (C, H, I, R, P, A)***, where:

1. ***C*** is a set of concepts in the domain;
2. ***H*** is a set of taxonomic relations between concepts;
3. ***I*** is a set of relationships between classes and instances;
4. ***R*** is a set of other relationships;
5. ***P*** is a set of properties of classes; and
6. ***A*** is a set of axioms.

Most formal definitions of ontologies usually share characteristics, independently of the approach. Normally the use of *concepts (classes or entities)* is to group individuals (instances) that share properties (relations and literals); the use of *instances (individuals)* is to group them in classes; the use of *properties (object properties or data type properties)* is to define relations and individuals characteristics.

The basic elements, conforming to [Gruber, 1993], [Noy and Mcguinness, 2001] and [Smith et al., 2004] are:

- Classes (entities or concepts): a core element of most ontologies. A group of individuals is represented by a class, which share common characteristics. They define an object in the real world. For example: Car, Animal, Vehicle, etc;

- Individuals (instances): the basic unit of an ontology; they are the things that the ontology describes (members of classes). For example: BMW i8 (Car), AH-64 Apache (Helicopter), Jumbo – the Elephant (Animal), etc;

- Properties: the characteristics that classes and individuals may have; they let us assert general facts about the members of classes and specific facts about the individuals. A property is a binary relation between them. There are two kinds of properties:

  - Datatype property: a relation between classes, or instances of classes and RDF literals and XML Schema datatypes. For example: *BMW i8* is an instance of *Car* class and has the *color* attribute as datatype property with the value blue;
  - Object property: a relation between classes, or instances of classes, and other classes (or instances), i.e., relations between two or more individuals of same or different classes. For example: *Ross* and *Monica* are two different instances of *People* class and they have a relation between them to inform that they are siblings. This relation may be called *hasSibling* and it is an object property. This example can be understood as *Monica hasSibling Ross.*

- Axioms: logical rules relevant to the concerned topic that evidence a fact. They are the knowledge that is not explicit in the taxonomy of the ontology and may be inferred. An example of axiom is to assert that every person has a mother.

Based in the aforementioned works and items, in this PhD research an ontology is formally defined as $\boldsymbol{O}$ as a 4-tuple $\boldsymbol{O = (C, I,} P_{dtO}\boldsymbol{, A)}$, in which:

1. $\boldsymbol{C}$ is a set of concepts (classes or entities) of the domain. For example, *Person*, *Animal*, *Car*, *Pizza*, etc. Conventionally, the name of a class usually starts with a capital letter and shall all be either singular or plural;

2. ***I*** is a set of instances (individuals) of a class. For example, *Renato Portaluppi* and *Zidane* are individuals of the *Football Coach* class; *Grêmio FBPA*, *FC Porto*, and *Real Madrid FC* are instances of the *Football Team* class;

3. $P_{dtO}$ is a set of properties that individuals and classes may have. In relations, usually their names start with a lowercase. Two types of properties are identified:

   - $P_{dt}$ is a datatype property, i.e., is the relation between classes, or instances of classes and atomic values, such as strings, integers, etc. These data types characterize the classes, or individuals. For example, the instance *Renato Portaluppi* of the class *Football Coach* has birth in year *1962*, i.e., the atomic value *1962* defines a characteristic (birth) of the individual *Renato Portaluppi*;

   - $P_O$ is a object property, i.e., is a relation between classes, or instances of classes and other classes, or instances. These object properties include the hierarchical (or taxonomic) and the non-hierarchical relations. For example: the instance *Luan* has an object property called *playsIn* related to the instance *Grêmio FBPA*. This can be read as *Luan playsIn Grêmio FBPA*.

4. ***A*** is a set of axioms. They hold the knowledge that can be inferred and that is not clearly detailed in the taxonomy of the ontology, i.e., They provide consistency to the ontology and are used to infer new knowledge. For instance, *if two children have the same mother, then they are siblings*. If the facts *Ross hasMother Judy*[14] and *Monica hasMother Judy*[14] are described in the ontology, this implies the fact (nonexistent in the ontology) that the individual *Ross hasSibling Monica* which is a symmetric property (relation), i.e., *Monica hasSibling Ross*.

[14] *Judy* is the same individual (person) to all examples

## 3.3 Advantages of Using Ontologies

In the literature, many times the term ontology is linked to the concept and benefit of the re-use, i.e., the benefit of sharing and using again the vocabulary described by a domain ontology. However, this is not the only purpose of using ontologies.

This section presents a list of some benefits of using ontologies (including the reusability) according to [Uschold and Gruninger, 1996], [Guarino, 1997], [Uschold and Jasper, 1999] and [McGuinness and Harmelen, 2004]:

- Ontologies provide a vocabulary to the knowledge representation of a domain. This vocabulary helps the applications that use it to avoid ambiguous interpretations. For example: if two people are having a conversation and one says to the other the word *seat*, he may be talking about something designed to support a person in a sitting position, as a chair, for example, as well as about the brand of vehicles Seat. The interpretation of the word may be assigned to a concept or another; it depends on the mental condition of the person and the context;

- Ontologies allow knowledge sharing. Thus, if there is an ontology that adequately models a knowledge domain, it can be shared and used by developers inside the domain of the ontology. For example: consider that an ontology exists to describe a domain of restaurants. Since that ontology is available, many restaurants can make their menu (card) using the ontology's vocabulary provided without the necessity of redoing a restaurant domain analysis;

- An ontology may be used to search serving as metadata or an index into an information repository;

- Ontologies guide knowledge acquisition;

- Ontologies allow the automation of consistency verification, producing more reliable systems;

- Ontologies in systems development can render maintenance easier. Systems which are built using explicit ontologies serve to improve software documentation, which reduces maintenance costs. A nice example is the application of ontologies to help in program comprehension (the activity behind maintenance tasks) [Carvalho, 2014].

## 3.4 Ontologies for Cultural Heritage

This section characterizes the Cultural Heritage domain and presents the standard CIDOC-CRM ontology specialized in that domain. In that way, it reinforces the importance of ontologies (Section 3.3) in the context of ICH preservation (Section 2.3.1).

Cultural Heritage has its history (people, events, places, periods, etc.) and is related in many ways to the society as archival collections, for instance. Because semantic richness is a special feature of cultural collections content, it clearly shows the importance of using ontologies to describe the Cultural Heritage domain [Nisheva-Pavlova et al., 2008].

Besides this, Cultural Heritage is a promising application domain for semantic web technologies due the semantic richness and heterogeneity of cultural content [Hyvonen, 2009].

Some ontologies in which the Cultural Heritage domain can be described in are: ABC [Lagoze and Hunter, 2001], DOLCE [Masolo et al., 2003], CIDOC-CRM [ICOM/CIDOC, 2015], among others that can be seen in [Doerr, 2009]. Usually, these ontologies are used to facilitate the interoperability between vocabularies of metadata from distinct domains, but they can also be used to describe the museums, archives or libraries assets, for example. This PhD research proves this by using the CIDOC-CRM ontology to describe two of the case studies of this work.

As reported by [Noy and Mcguinness, 2001] and mentioned earlier in this chapter, an ontology is a formal explicit description of concepts about a domain. Considering this, the CIDOC-CRM ontology, that describes the concepts and relationships used in Cultural Heritage documentation, is used in this PhD work. CIDOC-CRM ontology is, since 2006, an official standard (ISO 21127:2006[15]) and will be described in Section 3.4.1.

### 3.4.1 CIDOC Conceptual Reference Model Ontology

The objective of CIDOC-CRM is to promote a shared understanding of the Cultural Heritage domain by providing a common and extensible semantic framework that any Cultural Heritage information can be mapped to. In this way, it can provide the *semantic glue* needed to mediate between different sources of information, such as that published by museums, libraries and archives [Doerr et al., 2003] [Oldman and Labs, 2014] [ICOM/CIDOC, 2015].

The first thing to know about an ontology is its scope (domain). Thus, those who want to use it, should know what it covers and what it does not cover. Following this, the scope of CIDOC-CRM is divided into two types (intended scope and practical scope) [ICOM/CIDOC, 2015]:

- *The intended scope* can be understood as all information required for the scientific documentation of Cultural Heritage collections;
    - The term *scientific documentation* in this context conveys that the information handled by CIDOC-CRM should be sufficient to the academic research. This does not exclude the question about the content presentation to the general public, but emphasizes that the ontology is intended to supply the information details corresponding to the level required by professionals of museums, libraries and archives;
    - The term *Cultural Heritage collections* is intended to cover all types of material collected and displayed by museums and related institutions, as defined by ICOM (International Council of Museums);

[15] `https://www.iso.org/standard/34424.html`

- The documentation of collections includes the detailed description of individual items within collections, groups of items and collections as a whole. CIDOC-CRM is specifically intended to cover contextual information: the historical, geographical and theoretical background that gives museum collections much of their cultural significance and value.

- *The practical scope* of CIDOC-CRM ontology is driven by the conceptual reference standards documentation (that has many versions and the last updated one is the version 6.2[16] of May 2015). It is used to guide and validate the CIDOC-CRM development, i.e., the data properly encoded according to these documentation standards may be a CIDOC-CRM compatible expression.

In summary, the scope of CIDOC-CRM is the exchange of relevant information with libraries, museums, and archives. The necessary information for the administration and management of cultural institutions, such as information related to personnel, accounting and visitor statistics are beyond the scope of CIDOC-CRM.

To understand the core of the CIDOC-CRM ontology, Figure 3.1 illustrates it.

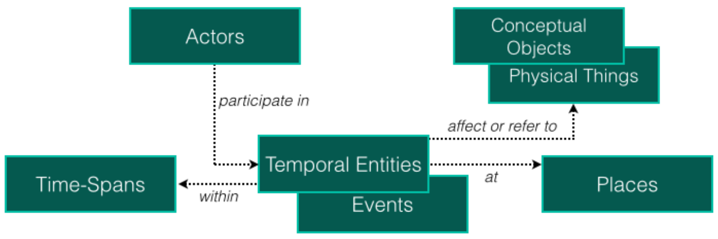


Figure 3.1: The core structure of the standard ontology CIDOC-CRM (adaptated from [Oldman and Labs, 2014])

Moreover, an explanation of each CIDOC-CRM's entity (core structure) is provided:

[16] Available at: `http://www.cidoc-crm.org/Version/version-6.2`

- The CIDOC-CRM is event-based. At the core of this event model are *Temporal Entities*, i.e., things that have happened in the past;
- As the *Temporal Entities* are classes related to time, they can have *Time Spans*. *Actors*, *Conceptual Objects*, *Physical Things* and *Places* can not be directly linked to time. Thus, they must be associated to an event, i.e., a *Temporal Entity*;
- A *Place* can be anything that describes a location, like a geographical location or a location defined inside a car or on the top of Mount Fitz Roy in Argentina/Chile;
- *Actors* are entities with legal responsibility. An actor can be an individual or a group. An individual may be a person and a group may be a company, for example. Actors interact with things (*Conceptual Objects* and *Physical Things*) through events;
- A *Physical Thing* is a thing that could be physically destroyed and transformed (created) into something else preserving parts of it;
- *Conceptual Objects* can not be destroyed, unless all carriers (a book, a painting, a computer disk, the human mind, etc.) of them are destroyed. Thus, to destroy a conceptual object, it is needed to destroy all of its carriers, including people;
- Things in CIDOC-CRM can have names, i.e., *Appellations*. An appellation is an object title (name), an identification number, etc;
- Different organizations have different classification systems. In CIDOC-CRM, classifications are called *Types* and they are referred to the classification of things. For instance wedding, earthquake, skirmish are types of events.

Furthermore, the CIDOC-CRM ontology has name conventions that should be followed. Any concept starts its name with the capital letter "E" (meaning Entity) followed by a numerical code (e.g. E39 Actor, E53 Place, etc.). The relations are no different, they start their names with the capital letter “P” (meaning Property) followed by a numerical code (e.g. P89 falls within, P131 is identified by, etc.).

## 3.5 Summary

This chapter described the definition of ontology, as well as its formal definition in the Computer Science area.

Thus, to represent an ontology computationally, the use of some languages are needed. They were listed and explained in order to show some forms in which ontologies can be described.

The components that belong to an ontology were presented. In addition, the classes, instances, and relations were depicted.

Furthermore, some advantages of using ontologies were shown, presenting a list of benefits and showing why ontologies should be used.

Concluding the chapter, an ontology for the Cultural Heritage domain, called CIDOC-CRM, which is a standard to describe issues on this topic, was presented.

The next chapter presents the Domain-Specific Language (DSL) concept, necessary to achieve the main goal of this PhD work. The Domain-Specific Language definitions, as well as the internal and external DSLs and the life cycle of them, are described. Also, some pros and cons of its use and implementation are discussed. Furthermore, some DSLs aiming at generating web applications are presented.

# Chapter 4

# Domain-Specific Languages (DSLs)

Domain-Specific Language (DSL) is an old term defined in general as a language targeted to solve a specific problem. Its development is hard because it requires both domain knowledge and language development expertise [Mernik et al., 2005].

As stated in [Ghosh, 2011], Domain-Specific Languages are designed to settle problems from a specific area. According to [Deursen et al., 2000], DSLs are used in Software Engineering in order to enhance quality, flexibility, and timely delivery of software systems, by taking advantage of specific properties of a particular application domain.

With an appropriate DSL, complete application programs for a specific domain can be developed more quickly and more effectively than with a general-purpose language [Hudak, 1998].

To make clear what is a DSL and what its purpose is, imagine that you want to buy a new car, then you go to some car dealership and specify to the salesman that you wish a car with Anti-lock Braking System (ABS) and the Emergency Braking Assistance (EBA). Besides, you specify the car should have a Turbo High Pressure (THB) engine with 165 horsepower (HP) and six speed automatic sequential transmission plus sport mode and Hill Assist system. The seller will show you a car model with the features according to your specification. You do not need to explain each term about the car. The dealer knows exactly what you have ordered, because you have used the precise language of that domain, which is shared between the two parts (the seller and the client). However, to others in a different context, the terms might not make sense.

Having the concepts introduced in mind, it is notable the accordance between the authors about the definition of Domain-Specific Languages. But to explain in detail what is a DSL, the more elaborated definition of Martin Fowler in his book entitled "Domain-Specific Languages" [Fowler, 2010] can be followed. Fowler defines DSL as a "computer programming language of limited expressiveness focused on a particular domain". Martin Fowler also explains that there are four key elements to his definition:

- *Computer Programming Language*: a DSL structure should be easily understood by humans, but it should also be readable to be executable in a computer, i.e., a DSL serves to instruct a computer to do some task;
- *Language Nature*: a DSL is a programming language where the expressiveness comes not just from individual expressions, but also from a set of expressions composed together;
- *Limited Expressiveness*: differently of a general-purpose language that provides lots of capabilities like supporting several datatypes, control, and abstraction structures, which is useful, but makes the language harder to be learned and used, a DSL provides a minimum of features needed to support its domain, i.e., a DSL has only the necessary features. Neither more nor less than it needs to solve the domain problem;
- *Domain Focus*: to be a worthwhile language, a DSL should have a particular focus turned to a small domain.

## 4.1 Classifying DSLs

The most common way to classify DSLs is associated to the form that they are implemented. As stated in [Fowler, 2010] and [Ghosh, 2011] and classified by almost all practitioners, DSLs are organized into two main categories:

- *Internal DSLs*: also known as embedded DSLs because they are implemented as an embedding within a host language (general-purpose language), i.e., an internal DSL is a valid code in its general-purpose language written in a syntax that extends the original one. One of the most popular internal DSLs is *Rails*[1], which is a DSL to develop web applications and is implemented on top of the Ruby[2] language, which is considered very suitable as a host for internal DSLs. Another example of DSL that is widely used is the Application Programming Interface (API) that covers a domain-specific vocabulary of class, method, and function names that becomes available by object creation and method invocation to any general-purpose language using the library;
- *External DSLs*: languages not incorporated in general-purpose languages, i.e., they do not use the syntax of a host language, but they can be based in a common language like XML (what often happens), for instance. This kind of DSL has its own structure with lexical analysis, parsing techniques, interpretation, compilation, and code generation. Summing it up, it is a new language built from scratch with its own syntax and semantics. Examples (divided by domains) are (not restricted to):
  - Queries: SQL (relational databases), SPARQL and SQWRL (ontologies), etc;
  - Markup Languages: HTML, XML-Schema, OWL, RDF, etc;
  - Stylesheet description languages: CSS;

[1] See more at: `rubyonrails.org`
[2] See more at: `www.ruby-lang.org`

– Parser-generator languages: ANTLR, YACC, etc.

## 4.2 Life Cycle of DSLs

After conceptualizing and classifying DSLs, this section will describe, in accordance to [Mernik et al., 2005], the five phases[3] (decision, analysis, design, implementation, and deployment) needed to develop a DSL.

### 4.2.1 *Phase One*: decision

To start the decision phase, it is necessary to think about making or using an existing DSL. This decision is a hard task, but it is a necessary step towards the next stages of the life cycle of DSLs.

Thus, if a domain is known and has a lot of knowledge about it (documentation and literature review, for example), maybe the creation of a DSL from scratch is the best choice. Otherwise, if a DSL that describes the domain in question exists, it becomes clear that reusing it requires less expertise than developing a new one.

If none of the cases above is what fits best, the general-purpose language should be taken into account, i.e., if the domain is new and little knowledge about it is available, it does not make sense to develop a DSL from scratch. To this case, the best choice is to determine the basic concepts of the area and try to develop a code base supported with libraries (APIs).

### 4.2.2 *Phase Two*: analysis

The goal in this second stage of the life cycle of DSLs is to identify the problem domain and collect the domain knowledge.

[3] These phases are not a sequential process and should be applied in an iterative way.

The analysis phase may have several sources as inputs, such as technical documents, knowledge provided by domain experts, etc; and the outputs, although vary, are formed basically of domain-specific terminology and semantics in a more or less abstract form. Usually this phase is done informally, but with the beginning of the exploration of knowledge representation and ontology development, the formal methods are potentially useful.

The analysis of a formal domain has as output a domain model, which consists of *the scope of the domain*, *the domain terminology like vocabularies, ontologies, etc.*, *the characterization of domain concepts* and *the commonalities and variabilities of domain concepts and their interdependencies.*

### 4.2.3 *Phase Three*: design

As stated in [Mernik et al., 2005], approaches to designing a DSL can be defined as *the relationship between the DSL and the existing languages* and *the formal nature.* Thus, a DSL can be composed from scratch or it can be based on an existing language.

To the first approach (*based on an existing language*), Mernik recognizes three different patterns of design, that are: (1) *piggyback* that is when an existing language is partially used; (2) *specialization* that is when an existing language is restricted; and (3) *extension* that is when an existing language is extended.

In contrast to the first approach, Mernik identifies for the second approach, two variations: (1) *informal* that is a DSL specified in natural language with examples; and (2) *formal* that is a DSL stated using an available syntactic or semantic specification method like rules, grammars, regular expressions, etc.

It is crucial to determine which approach to take into account for the design of a DSL. As reported by [Mernik et al., 2005], the easiest way to design a DSL is to base it on an existing language (if the users of it are also programmers who know the host language). The possible benefits are easier implementation and familiarity for users.

### 4.2.4 *Phase Four*: implementation

With the design phase done, a proper implementation mechanism should be chosen. In consonance with [Mernik et al., 2005] and the study carried out on [Kosar et al., 2008], the existing patterns about DSL implementation can be characterized in seven items described next.

1. *Interpreter* — DSL constructs are identified and interpreted using a standard instructions cycle (fetch-decode-execute). An advantage of the interpreter over the compiler is the simplicity, control over the execution environment, and easier extension;

2. *Compiler/Application generator* — DSL constructs are translated to a base programming language constructs and library calls. A complete static analysis can be done on the DSL program/specification;

   The disadvantages of compilers and interpreters is the cost of building them from scratch.

3. *Preprocessor* — DSL constructs are translated to constructs in an existing language (the host language). Static analysis is limited to that done by the host language processor. There are subpatterns like *macro processing* that is an expansion of macro definitions; *source-to-source transformation* that is when the DSL source code is transformed (translated) into a host language source code; *pipeline* where processors successively handle sublanguages of a DSL and translate them to the input language of the next stage; *lexical processing* where only simple lexical scanning is required, without complicated tree-based syntax analysis;

4. *Embedding* — new data types and operators are embedded in an existing host language. Application libraries are the basic form of embedding;

5. *Extensible compiler/interpreter* — A compiler/interpreter for a general-purpose language is extended with domain-specific optimization rules and domain-specific code generation. Interpreters are usually easy to extend, but compilers are hard to extend unless they were designed having extension availability;

6. *Commercial Off-The-Shelf (COTS)* — Existing tools and notations are applied to a specific domain;

7. *Hybrid* — A combination of the above.

### 4.2.5 ***Phase Five*: deployment**

In this phase, the implemented DSLs and applications built with them are used. Developers and domain experts use the DSLs to specify models. With one of the above mentioned patterns of implementation, these models are implemented, i.e., this phase is the stage where the system built through the models with a pattern implementation of DSLs is deployed and used by end-users.

In addition to these five phases, [Visser, 2008] yet adds one more stage, which he calls *maintenance*, which in turn is the phase where the corrections are made. Some substantial changes in the software may involve altering the DSL implementation.

## 4.3 DSL Design Guidelines

To achieve a better acceptance level among the users of the Domain-Specific Languages, and aiming at a higher language quality, some guidelines for the design phase were considered based on [Karsai et al., 2009].

The authors introduce 26 guidelines based on their experiences and in the literature, aiming at improving the design and the usability of DSLs. Guidelines to design a DSL should be followed, but some of them contradict each other. Therefore, which should be adopted need to be weighed and balanced according to the domain of use [Karsai et al., 2009].

Karsai et al., have classified the design guidelines into five main categories:

- Language Purpose – which introduces guidelines for the previous tasks of the language development phase. This category groups the guidelines for analyzing the purpose of the language;
- Language Realization – which groups guidelines to help how to implement the DSL;
- Language Content – which classifies guidelines targeting the elements of a DSL;
- Concrete Syntax – which establishes guidelines for the external representation of a DSL;
- Abstract Syntax – which groups the guidelines for the readable representation of a language (internal DSL).

From these 26 guidelines, 17 were selected as important for this doctoral work[4]:

- Language Purpose:
  - Guideline 1: identify language uses early helps determining the concepts the language will provide;
  - Guideline 2: after guideline 1 is done, it is necessary to ask some questions to determine who is going to use the language and to identify the complexity of the language;
  - Guideline 3: make the language consistent. As DSLs have a specific purpose, each feature of the language should support this purpose;

[4] As the DSL proposed in this PhD project is an external language, the "Abstract Syntax" category is not taken into account.

- Language Realization:
  - Guideline 4: decide to use graphical or textual representation of the language. Textual representations generally have a faster development and are platform and tool independent;
  - Guideline 6: reusing existing language definitions is a good solution, since adopting the definition of a well-established language to start the development of a new language is better than building a DSL from scratch. In this work, the proposed DSL is based on the JSON definition;
- Language Content:
  - Guideline 8: reflect only the necessary domain concepts. A set of concepts shall be defined, allowing the language user to declare all the essential domain concepts. This PhD research uses ontologies as the set of the domain concepts and relations;
  - Guideline 9: keep the language simple. The new DSL need to be as simple as possible. "Simplicity is one of the main targets in designing languages" [Karsai et al., 2009]. If the language is hard to understand and use, it will not be adopted;
  - Guideline 10: avoid irrelevant generality. This means that the language should be designed only with what is necessary. In this work, the ontology (as in guideline 8) controls the generality;
  - Guideline 11: limit the number of language components, since languages with many elements lead to a difficult understanding;
  - Guideline 12: avoid conceptual redundancy. This means that having several concepts to describe the same thing allows users to write it in different ways, which should be avoided;
  - Guideline 13: avoid inefficient language elements. The generation of efficient code is necessary, since it can lead to some problems like increase of memory usage;

- Concrete Syntax:
  - Guideline 14: existing notations should be adopted. As already mentioned in guideline 6, the JSON notation is followed in this work;
  - Guideline 15: use descriptive notations. This means that the keywords should not be difficult to identify. A good idea is to limit the number of keywords to easily remember;
  - Guideline 16: make elements distinguishable. In textual representations, keywords should be placed in certain positions of the concrete syntax;
  - Guideline 20: balance the compactness and comprehensibility of the notation, without much detail;
  - Guideline 21: use the same style in all elements and portions of the language. For example, if parentheses are used for a purpose, it is not suitable to use curly brackets for the same purpose in the language;
  - Guideline 22: identify usage conventions. To keep the language readable and easy to write, a good idea is to follow some conventions. For example, use uppercase or lowercase for identifiers, use quotation marks for the value of a key-value pair, etc.

These guidelines helped in the design phase of the proposed DSL. When a language is built from scratch, some guidelines can be adapted more easily than in the case of having a host language. Furthermore, depending on the domain, some guidelines may have more relevance than others, so they need to be weighed in order to know if the guidelines should be followed or not.

## 4.4 Advantages and Disadvantages of DSLs

Designing and implementing a DSL involves both benefits and disadvantages. Before deciding between implementing a DSL or not, it need to be weighed the pros and cons. This decision should be taken depending on each applicable circumstance.

Many authors describe the pros and cons based on the programmers and domain experts viewpoints as well as discriminate some viewpoints based on the development phase and use of the DSLs. In this section, all these aspects covered in [Deursen et al., 2000], [Oliveira et al., 2009], [Fowler, 2010] and [Ghosh, 2011] and the benefits and drawbacks are pointed.

### 4.4.1 Advantages

The most declared advantage of DSLs is the expressiveness that DSLs have. It enables DSL programs to be expressed in the level of abstraction of the problem domain. Thus, domain experts can understand, validate, and even specify DSL applications.

Because a DSL has a higher level of abstraction, learning and using it is more accessible than general-purpose languages. Furthermore, with this level of abstraction, the communication between the developers and domain experts can improve. This benefit is explained by Martin Fowler in [Fowler, 2010] as "some people find that trying to describe a domain using a DSL is useful even if the DSL is never implemented. It can be beneficial just as a platform for communication". The author concludes yet that the involvement of domain experts in a DSL is difficult, but has a high payoff (productivity and better description of the domain, for instance).

Other advantages that DSLs programs have are self-documenting and succinctness for several purposes. DSLs also have the benefit of enhancing productivity, maintainability, portability, and reliability. Furthermore, DSLs allow the reuse of the domain knowledge.

Finally, but not less important, is the fact about the development of a DSL that gives return when the complexity of the domain is high. This is an important aspect that should be taken into account before deciding to implement and use a DSL.

### 4.4.2 Disadvantages

The disadvantages of using a DSL are summarized by the costs of design, implementation and maintenance phases. Many times, the difficulty of finding the appropriate scope for a DSL and the difficulty of balancing between domain-specificity and general-purpose are common drawbacks.

Besides this, as stated in [Ghosh, 2011], an external DSL is another language to learn, i.e., any external DSL has to be learned separately by the developers.

Lastly, according to [Fowler, 2010] and [Ghosh, 2011], the development of a system normally uses more than one DSL and this makes it harder to understand what is going on in the project. The conclusion of this is that DSL composition is not easy, because individual DSLs tend to evolve independently of each other. This disadvantage is called by Martin Fowler as a *language cacophony problem*.

## 4.5 DSLs to generate web applications

There are some similar DSLs related to the purpose of this work: MobiDSL [Kejriwal and Bedekar, 2015], EngenDSL [da Purificação and Silva, 2009], WebDSL [Visser, 2008], WebML [Stefano Ceri and Matera, 2001], WeSiMoLa [Stibe and Bicevskis, 2009], among others. The three more interesting are described next.

In [Kejriwal and Bedekar, 2015], the author developed the MobiDSL language, which has the goal of specifying pages/screens to mobile applications, allowing the programmer to specify page structure, presentation (views), data retrieval, etc.

In [da Purificação and Silva, 2009], the author developed a DSL called EngenDSL which aids in developing applications by abstracting user interaction concepts based on the Interaction Flow Modeling Language (IFML) standard using the Apache Velocity template engine.

Another DSL for this purpose is WebDSL described in [Visser, 2008]. WebDSL is a language for developing dynamic web applications with a rich data model translating the specification to Java web applications.

These DSLs are languages to create, in an abstract way, web applications in general. Besides, most of them need expertise in technical tools and programming languages to specify the entire web application. The difference of these works to this one, is that they require expertise in the computer science area. In this PhD research, a DSL was created with an abstract definition level that the user does not need to know about data persistence, programming or markup languages. Furthermore, the designed DSL is focused on the vocabulary of the museum curators, facilitating the specification of the virtual Learning Space.

## 4.6 Relating ontologies and DSLs

Parenthetically, the relationship between DSL and ontologies is clear and necessary to this thesis. As stated in [Kosar et al., 2008], the *implementation* phase of DSLs has attracted a lot of research, but the *analysis* and *design* phases (less known and not closely examined) are as important as that.

Knowing the necessity of describing as much precisely as possible the domain, the *analysis* phase should be strictly examined. Thus, the knowledge of the problem domain and its definition is achieved. Therefore, the use of an ontology is a good way to solve the aforementioned problem because an ontology provides a vocabulary that represents the objects and concepts of a precise domain [Ceh et al., 2011].

As stated in [Oliveira, 2009], this relationship between both, ontologies and DSLs is so strong and relevant that some authors in their works, like [Ceh et al., 2011] and [Fonseca, 2014] decided to map formally ontologies to (DSL) grammars and develop tools to automatically generate the grammars for their domain ontologies. However, this approach is out of the range of our work.

As mentioned above, the phases of *analysis* and *design* of a DSL are important steps to determine the specific domain to be treated. However, in this PhD work it is intended to describe the Cultural Heritage domain in an ontology and relate it with a DSL description with the purpose of describing the information that must be displayed in the final virtual Learning Spaces. Thus, the DSL specification shall access the Cultural Heritage ontology to describe the concepts and relations desired to be shown in the virtual LS. This relation will be more detailed in Chapter 6.

## 4.7 Summary

This chapter described the definition of DSLs as a computer programming language aiming to solve a particular issue. This means that a DSL, with the minimum needed features, serves to guide a computer to take on a function.

Furthermore, the classification of DSLs associated to the form they are implemented was detailed, showing, on one hand, that the internal ones are also known as built-in DSLs in a general-purpose programming language. On the other hand, the external ones are not embedded in host languages using its syntax, but they can be based in a common language like XML, for example.

The chapter outlined the life cycle of DSLs, presenting its five phases: decision, analysis, design, implementation, and deployment. Besides this, the pros and cons of designing and implementing a DSL were discussed.

Finally, enclosing the chapter, a relation between ontologies and DSLs was discussed. This relation between both subjects is important to this PhD research due to the need that the DSL specification has to have access to the Cultural Heritage ontology domain to describe the concepts and relations desired to be presented in the virtual Learning Space.

The next chapter presents the Learning Spaces, showing the definition of the traditional ones and the virtual ones. Besides, some comparisons, benefits and drawbacks of them are detailed.

# Chapter 5

# Learning Spaces

The spread of the Internet and technology has changed the way of reading, writing, thinking, and learning [Rainie et al., 2010]. Furthermore, according to [MCEECDYA, 2008], the rapid and continuing advances in information and communication technologies (ICT) are changing the ways people share, use, develop and process information and technology.

As a consequence of this, the traditional way of learning is changing. The facilities to find the desired information nowadays grow up every day due to the Internet. Thus, the learning environments have also changed.

Traditional Learning Spaces, as mentioned in Chapter 1, are physical locations where people meet to learn about a specific subject with the aid of a person (leader), who transmits the knowledge about a specific domain.

The term Learning Space is usually used to describe environments for the students and professors in schools, universities, or similar educational institutions. However, this term can be perfectly used to describe the kind of learning environment that is sought in this work. As mentioned in Chapter 1, the term e-Learning or any subject related to learning management in educational institutions should not be applied in the context of this doctoral project. The Learning Spaces described here are concerned with people who wants to learn by visiting institutions like museums, libraries, and archives, that hold documents or objects as primary sources of information and so to transmit knowledge.

Knowing this and already having defined the traditional concept of LS, in Section 5.1 is defined the new concept of virtual Learning Spaces, which inherits much of the traditional environments, but is focused on a new way of learning.

## 5.1 Virtual Learning Spaces

Faced to the non-formal methods of learning, this section presents the virtual Learning Spaces focused on cultural institutions like museums, archives, and libraries.

As previously mentioned in this work, it is hard to explain the virtual Learning Spaces without refering to the educational LS, because that is the main domain where they are discussed in the literature. Most of the works are related to the virtual Learning Environments for students and professors where the educational scenario has a professor, tutor, or someone to lead and to impart knowledge to the whole class virtually. This scenario applies to educational institutions, like schools and universities, offering e-Learning and distance education. [Jaligama and Liarokapis, 2011], [Piccoli et al., 2001], [Callaghan et al., 2009], and [Sharma et al., 2015] are some of these works. In [Moore et al., 2011], the different terms used to the educational Learning Spaces are explained.

On the other hand, there are many works in the area of museums and cultural institutions focused on a 3D virtual tour or digitization of the documents and objects, like [Jaén et al., 2005], [Sacher et al., 2013], [Hess et al., 2015], among others. Another area that is being explored is the Virtual Reality (VR) and Augmented Reality (AR). Some works in this area are [Madsen and Madsen, 2015], [Hürst et al., 2016], and [Kersten et al., 2017].

In this PhD work, the objective is to create Learning Spaces to the end-users (the institution "visitors") without a tutor and unlike the virtual reality environments, like 3D virtual museums or analogous. So, the end-users should learn from provided information through non-formal methods, in their leisure time and in informal occasions and places.

So, the definition and practical applications of Learning Spaces can be useful both to the educational institutions and to the cultural institutions. The scope of this PhD work is restricted to the virtual Learning Spaces related to cultural institutions (museums, archives, and libraries), not dealing with educational environments like those concerned with distance education.

Thus, in this work, a virtual Learning Space is defined as a LS similar to a digital library. A digital library, according to [Witten et al., 2009], is a focused collection of digital objects that can include text, visual material, video material (stored in electronic media formats), along with means for organizing, archiving, and retrieving the files and media contained in the library collection.

Ian Witten adds that digital libraries can be maintained by individuals, organizations, or affiliated with established physical library buildings or institutions, or with academic researchers [Witten et al., 2009]. This definition can be easily adapted to other institutions that also have collections of information sources to be stored and managed.

Thinking in this way, a book – of a library –, a document – of an archive –, or an object – of a museum – are similar things that can be stored in collections. Each one is an information source; that information should be stored and displayed through the virtual environment so that the end-users can learn effectively.

Thus, in the context of this doctoral work, the specific definition of a virtual Learning Space is an environment similar to the traditional one, but without a person to lead the knowledge transmission; the information to be passed to the learners is arranged in such a way that the users can learn individually using the tools and features provided to them by the environment. These features and tools can be a powerful search mechanism where the information displayed is only what the users want. These Learning Spaces and their tools and features are explained better in Chapter 6.

## 5.2 Comparison between Traditional and Virtual Learning Spaces

This section describes the differences and makes a comparison between the traditional and virtual Learning Spaces based in [Piccoli et al., 2001]. In this work, the study was made relating the traditional and virtual environments in the educational domain. However, as mentioned before, the Learning Spaces described in this work can be defined and related in the same way as the educational ones.

The comparative study of [Piccoli et al., 2001] takes into account the parameters *Time*, *Place*, *Space*, *Technology*, *Interaction*, and *Control*. These six items are described:

- *Time*: the timing of the instruction, i.e., the control over the pace of the learning;
- *Place*: the physical location of instruction;
- *Space*: the collection of materials and resources available to the learner;
- *Technology*: the collection of tools used to deliver learning material and to facilitate communication among participants;
- *Interaction*: the degree of contact and educational exchange among learners and between learners and instructors; and
- *Control*: the pace in which the learner can control the instructional presentation.

In Table 5.1, there is a comparison between the traditional and virtual environments considering these six points.

Table 5.1: Comparison between Virtual and Traditional Learning Spaces (adapted from [Piccoli et al., 2001])

| | **Learning Space** | |
|---|---|---|
| **Topic** | **Traditional** | **Virtual** |
| **Time** | Visitors have a specific time to visit the institution depending on open hours | Visitors connect to the virtual (online) Learning Space when they choose |
| **Place** | Visitors should go to the physical institution | Visitors connect to the virtual (online) Learning Space where they are |
| **Space** | Visitors share physical space with others during the opening hours | Visitors do not share the resources |
| **Technology** | Visitors are limited to use the technology available in the physical space only | Visitors use the features and tools available in the virtual Learning Space |
| **Interaction** | Visitors are able to interact face-to-face with other visitors or contact directly with any kind of artifacts and their information | Visitors are able to interact with other people through asynchronous communication and access any kind of artifacts and their information via the tools and features of the virtual space |
| **Control** | Visitors do not have full control over the pace at which they access the information | Visitors have full control over the pace in the access of the information |

### 5.2.1 Advantages and Disadvantages of Traditional and Virtual Learning Spaces

The advantages and disadvantages pointed in this section are related to the virtual Learning Spaces in contrast to the traditional ones. This relation is important because is through the traditional (physical) LS that the virtual surfaced.

Considering the topics in Table 5.1, in this section, the parameters are grouped by similarity to identify the pros and cons about the two kinds of LS (traditional and virtual).

Thus, among these six items, the first three, called here group (a), are based on a space-time relation, i.e., the observed parameters *time*, *place*, and *space* converge to the same pros and cons, because one depends upon the other.

Discussing the pros and cons of group (a), in the traditional LS (e.g. museum) there are many people sharing the same space, i.e., the same place at the same time with the same collection (artifacts), which causes a certain discomfort to the visitors of the LS.

On one hand, the ease of connecting to the Internet to access the information with the available features and tools of the virtual LS at any time and any place is a great benefit of the virtual LS over the conventional ones.

On the other hand, there are losses. One of the great benefits of visiting conventional learning environments is the exchange of knowledge, discussing and giving opinions about a particular subject with other people, i.e., human contact. Besides that, the visit to the real object or information in a museum, for instance, is invaluable. The contact with others and with the books, sculptures, or any artifacts, brings an incalculable satisfaction and experience to the visitor, which does not fully happen in the virtual environment.

Still discussing the benefits and drawbacks of the LS, the last three items (group (b)), *technology*, *interaction*, and *control*, are those related to the way of using and controlling the pace of the interaction with the artifacts, the people, and the technology ready to be used in each (virtual and traditional) LS.

According to Table 5.1, group (b) also has an association with the space-time, i.e., group (a), because the interaction with people and the use of the technology are limited to the control of the space and time.

So, the benefit of visiting a traditional LS, from the technology viewpoint, is the way of using it to better explore the artifacts in the space and time of the institution. An example can be described as a visitor in a science and technology museum, where he can interact with the equipment available only at the physical location.

In the virtual space, the visitors are restricted to the features and tools of the LS, which can be a benefit or a disadvantage over the conventional Learning Environment.

Following this thought, the interaction point has both benefits and disadvantages when the visitor interacts with the artifacts, people, and/or technology in the same sense of the technology item, i.e., the interaction can be made in the limitations of time, space, place, and technology in the traditional LS, and may not have these limitations in the virtual environment.

Finally, the last topic, control, can be defined to the traditional space as a disadvantage to the visitor, because he does not have full control over the pace at which he accesses the information, i.e., the control depends, again, of the time, the space, and the conditions of the place (e.g. easy access (many people, catering, etc.), disasters (fire, flood, etc.), among others.). In the virtual space, the visitor has full control over the pace of the access to the information.

Summing it up, all these benefits and disadvantages are associated to the group (a), that depends on time, space, and location.

So, the actual advantages and disadvantages must be weighed. To this PhD research — Learning Spaces built on intangible Cultural Heritage — although it can be interesting to visit the physical place where the original documents reside, the essential and most important is the information that the documents contain. Therefore, it is a fact that the tools and features that the virtual Learning Spaces provide are more feasible and cosy to access and use.

## 5.3 Related Projects – Generation of Virtual Learning Spaces

Some projects have proposed the creation of virtual Learning Spaces not just for students as it happens with e-Learning, but for a large range of people. Some interesting projects exist, more related to this PhD proposal, in addition to those mentioned in the beginning of this chapter, concerning the 3D virtual spaces and e-Learning. Here are reported three of them:

- *The Domus Naturae Project*, which aims at creating a virtual museum application using tools for managing structured knowledge based on ontologies [Ghiselli et al., 2005]. This project is similar to CaVa, since it uses ontologies to describe the repository of digital objects allowing the navigation over such objects. However, the ontology of CaVa serves to describe and relate semantic concepts of a specific domain, connects such concepts with the digital objects stored in the database repository, and makes the automatic generation of a virtual LS closer to the knowledge that the end-user wants to get;
- *The Personalised Access to Cultural Heritage Spaces (PATHS[1])*, which proposes the creation of a system (virtual thematic paths) that acts as an interactive and personalized tour guide through all objects available in the European Digital Library (Europeana). The aim of this project is to make the exploration of Cultural Heritage collections easy for people, leading them along a pathway that can be created by themselves or experts. Even though this project is similar to CaVa, the creation of paths to connect collections of objects is not intended by this PhD research;
- *The MuseumFinland* is a semantic portal for publishing heterogeneous museum collections on the Semantic Web. The system allows making

[1] available at: `http://paths-project.eu/eng/Prototype`

collections semantically interoperable, and provides the museum visitors with intelligent content-based search and browsing services, thanks to a set of ontologies. This system uses ontologies to navigate through meta-information based on a central repository [Hyvönen et al., 2005]. This project is similar to CaVa, however, this PhD work intends to allow the automatic creation of virtual LS not focused on museums collections.

In short, the projects presented above regard the area of Learning Spaces as web-based learning environments for knowledge generation and dissemination. Compared to the projects here discussed, the main feature that CaVa will innovate is the automatic generation of virtual Learning Spaces through a specification (a DSL), based on an ontology that describes and organizes the information stored in a digital repository.

## 5.4 Summary

This chapter defined the concepts of Learning Spaces focused on the virtual ones, but always comparing with the traditional ones, because the virtual, in this case, is the evolution of the traditional way of learning. In this chapter, it was defined the term virtual Learning Spaces for this work, keeping in mind that they are not like 3D spaces, not even for those focused on educational purposes, but driven to the cultural institutions like museums, libraries, and archives.

An important topic discussed in this chapter was the comparison between the traditional and the virtual LS. To this study, it was taken into account six parameters: time, place, space, technology, interaction, and control. As a result of this comparison, the advantages and disadvantages of the two kinds of Learning Spaces (traditional and virtual) were outlined.

The next chapter presents the proposal of this work, depicting the task to achieve the final virtual Learning Spaces focused on cultural institutions.

# Part II

# CaVa

# Chapter 6

# CaVa – Proposal

In Chapters 2 to 5 the relevant terms to this work were introduced and described (definitions and examples were given) to promote a better understanding of the scope of the thesis. These concepts will be applied in this PhD proposal architecture, called CaVa, to achieve the objectives outlined in Chapter 1. After this preliminary research, the CaVa architecture was created and designed.

## 6.1 The architecture

This chapter starts describing the general architecture proposed to build the CaVa system, as illustrated in Figure 6.1, and then along Section 6.2, each module will be introduced separately. In addition to the various CaVa modules, there is a basilar component: the ontology (Chapter 7), that serves to specify the knowledge domain that corresponds to the institutional information repositories. The actual input to CaVa (associated to module B) is the Learning Space Specification, written in CaVa$^{\mathrm{DSL}}$ (Chapter 9), that determines which concepts should be exhibited and how they should be placed in the final virtual Learning Spaces by the Curator.

Module A (Chapter 8), that is a kind of back office of this framework, is composed of the *Database Repository* and the *Ingestion Function* with the aid of a *Document (Data) Ingestion System* (DIS) to upload the institution documents information and data management. Module B (Chapter 10) is made up of four components that are *CaVa*$^{gen}$ (a specification engine), *CaVa*$^{DSL}$ *Specification*, *CaVa*$^{grammar}$ and *CaVa*$^{grammar}$ *Processor*, which processes the grammar and generates the specification engine. Module C (Chapter 11) is composed of the generated *LS Scripts*, the *Web Browser*, and the virtual Learning Space, which is the target of this doctoral project.

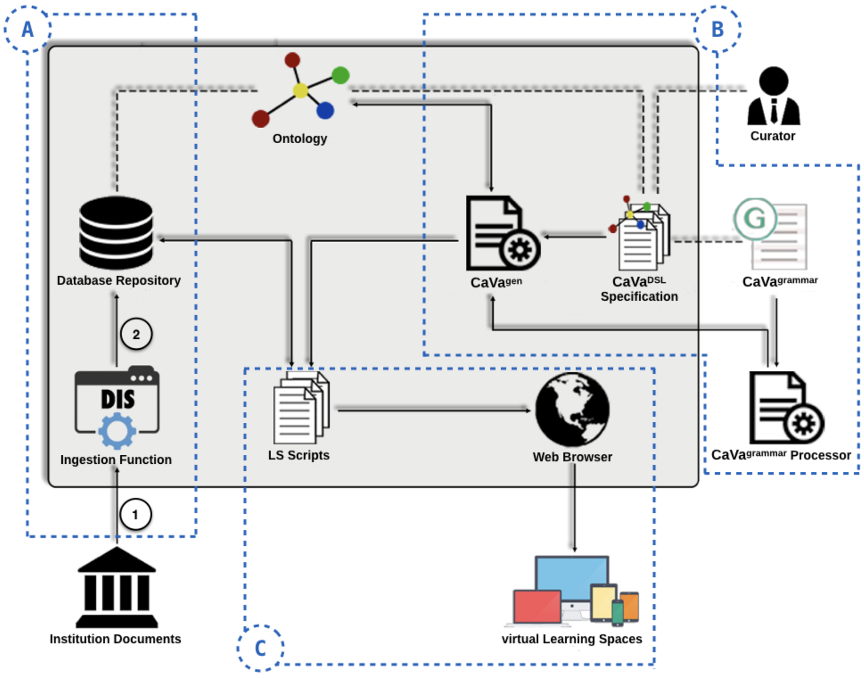


Figure 6.1: Proposed Architecture of the CaVa Project

This architecture was not designed for a specific domain. Instead, it shall support any knowledge domain associated with museums, libraries, archival or schools. Next, the components and the role of each module of the CaVa architecture will be detailed.

## 6.2 Components of the architecture

In this section, the requirements of each CaVa architecture's component are presented.

### Module A – CaVa$^{\text{settler}}$

The objective of CaVa$^{\text{settler}}$ is to collect physical documents information from the *institution Documents* (the actual input of a DIS) and populate the digital database repository aiming at displaying the document's knowledge in virtual Learning Spaces. These are the components for this module:

- *Ingestion Function (Document (Data) Ingestion System - DIS)*: it is the software (function) that performs the transport of the documents data to the database repository to be digitally stored. Basically, it extracts data from sources, storing the information into digital storage structures. Usually, this kind of system receives data through Optical Character Recognition (OCR) or forms that are manually filled;
- *Database Repository*: it is the digital storage structure that receives the data extracted (through a DIS) from the physical documents of organizations. These repositories can be classified as relational, object-oriented, triple store, etc.

### Ontology

To generate the virtual Learning Spaces, which have the objective of imparting knowledge about a specific domain to the end-users, it is necessary to describe the knowledge implicit in the information contained in the sources (database repositories). This is the purpose of the ontology in this PhD work.

Depending on the *Database Repository* type, some extra tasks are needed (e.g. if the repository is a relational database, it is necessary to do the mapping between the ontology and the database, aiming at achieving the database instances).

### Module B – CaVa$^{\text{processor}}$

CaVa$^{\text{processor}}$ contains the main components necessary to achieve the objective outlined in this thesis. CaVa$^{\text{processor}}$ is the machinery that interprets the CaVa$^{\text{DSL}}$ language specification (the CaVa input, manually written by the Curator) and uses CaVa$^{\text{gen}}$ to produce as output, the virtual LS scripts necessary to render the desired final virtual Learning Space. Remembering that the input of this module is a specification written in the specific language CaVa$^{\text{DSL}}$. To be clear, these specifications are built manually by the Curator that should have knowledge about (1) the language rules defined by CaVa$^{\text{grammar}}$, and (2) the main ontology that describes the documents repository. The components of the CaVa$^{\text{processor}}$ module are detailed below:

- CaVa$^{\text{DSL}}$: a Domain-Specific Language intended to aid Curators to create virtual Learning Spaces based on the museums assets and on the museological vocabulary;
- CaVa$^{\text{grammar}}$: a set of formal rules defining a formal language to be used to write sentences that specify a virtual exhibition. Phrases in CaVa$^{\text{DSL}}$ are recognized by the CaVa$^{\text{grammar}}$ Processor;
- CaVa$^{\text{grammar}}$ *Processor*: a parser generator and a compiler constructor that recognizes the CaVa$^{\text{grammar}}$ rules and generates CaVa$^{\text{gen}}$;
- CaVa$^{\text{gen}}$: a set of processors that performs two tasks before creating the LS Scripts: (1) recognize the CaVa$^{\text{DSL}}$ input specification of the LS, written by the Curator; and (2) access the main ontology to search for the concepts referred in the specification in order to identify their instances.

### Module C – CaVa$^{\text{render}}$

CaVa$^{\text{render}}$ is the module responsible for recognizing the LS Scripts generated by CaVa$^{\text{gen}}$ and rendering, via a web browser, the virtual Learning Space described in a CaVa$^{\text{DSL}}$ Specification. The components of the CaVa$^{\text{render}}$ module are detailed below:

- *LS Scripts*: these generated scripts describe the intended virtual LS in a less abstract level, i.e. they are scripts that configure parts of the LS, like the static structure and content, dynamic content (queries), and the webpage presentation style and client-side files. The LS Scripts should perform two tasks: (1) query the database repository to fetch the real information concerning the ontological concept instances; and (2) use this information to create the final virtual Learning Spaces through a web-browser;

- *Web Browser*: an application for retrieving and presenting document's content (information) in the World Wide Web. In this work, the web browser has the duty of receiving the LS Scripts generated by CaVa$^{\text{gen}}$ and rendering them, presenting the content of the exhibition rooms with the appropriate style, as described by the CaVa$^{\text{DSL}}$ Specification.

The *Virtual Learning Spaces*, the actual output of CaVa, are the final rendered Web pages that realize the virtual museum, library or archive. They are a place where the visitor can learn with the content at any time, any location and any device, without the aid of a tutor transmitting the knowledge.

Along this chapter, it was presented the proposal that supported this PhD work. The CaVa architecture was presented in a general way, describing the three main modules, which realize the aim of this project: the automatic generation of virtual Learning Spaces based on the CaVa$^{\text{DSL}}$ Specification and an ontology.

In addition to the general explanation, a detailed description of each component of the CaVa architecture, grouped by the modules, was presented, aiming at a better understanding of each role.

The "Museu Virtual Interativo da Fotografia[1]" (MVIF), that was developed by [Ravanello, 2018], will be used along this thesis to show how the proposed approach makes it possible to specify the virtual Learning Spaces in CaVa. MVIF will serve as a running example in the following chapters of II.

[1] This is a virtual museum that exposes the techniques, equipments, ideas and characters of the history of photography. In this work, the idea is not to reproduce the museum faithfully, but give a little example of the museum description, that is, how it would be specified in CaVa.

# Chapter 7

# Ontology

Working with cultural documents belonging to the repositories of archives or museums, aiming at transmitting the knowledge implicit in those documents to the public, is not a trivial task. To be successful, it is crucial to describe the knowledge contained in those documents.

To this work, it is important that the Curator, who holds the organization assets (documents) to exhibit to the public in virtual Learning Spaces, has the access to the documental information in an easy way. A good approach to achieve this is to describe the documents content in an abstract level, for example, through an ontology, which describes a particular domain with a specific vocabulary shared by all the community.

So the main objective of the ontology included in the CaVa platform, is to serve as a high abstraction level description of the documents content. As already mentioned in Chapter 1 (Section 1.2), the ontology has two purposes: (1) to give semantics to the database repository; and (2) to describe the information that must be displayed in the final virtual LS.

In addition to the mentioned advantages of using ontologies (Chapter 3 – Section 3.3) in this work, the ontology also accomplishes the important task of serving as a domain vocabulary specification to access the documents information stored in the digital repository. This is true, since there is no place in a relational database structure to capture, for example, business rules, create subsumption relationships and describe other key aspects of a conceptual model. Below, three reasons are outlined to justify the use of ontologies to access the sources instead of directly using the database:

- To define a standard vocabulary to describe a specific domain;
- To work with a large amount of data. According to [Rodríguez-Muro et al., 2013], an ontology defines a high-level global schema and provides a vocabulary for user queries. This isolates the end-user from the structure and data source details. Using a system that facilitates access to the database through ontologies is ideal, for example the Ontology-based Data Access (OBDA)[1], that transforms user queries into data vocabularies and then delegates the evaluation of the current query to the data sources;
- To group data under abstract concepts. An ontology organizes the information stored in a database, XML files, or any data source (computational structure), in a conceptual way, creating groups of restrictions over this information [Franconi, 2008].

To reproduce the "Museu Virtual Interativo da Fotografia" in CaVa, it is necessary to describe the context domain in which MVIF is enclosed. So, a portion of an ontology describing the MVIF museum will be presented. Figure 7.1 shows the two main concepts related to the MVIF domain, and four subclasses related with them.

[1] This is not needed in case of the database repository be a triple store.

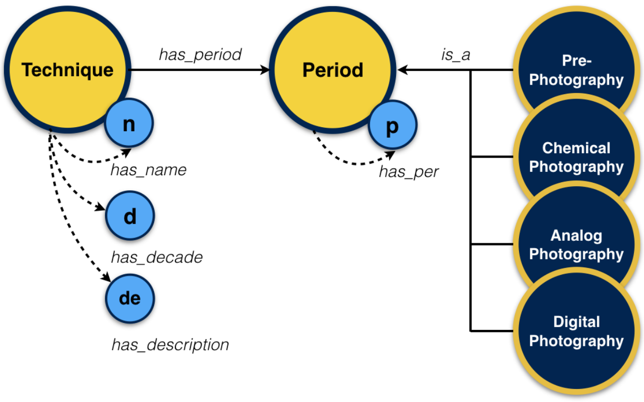


Figure 7.1: A portion of the “Museu Virtual Interativo da Fotografia” ontology

The ontology describing the domain of the running example contains: concepts like “Technique”, and “Period” and subclasses of “Period” like “Pre-Photography”, “Chemical Photography”, “Analog Photography, and “Digital Photography”; relations[2] like “has_period”, and “is_a” (that defines the “subclass of” object property); and datatype properties like “has_name”, “has_decade”, “has_description”, and “has_per” describing the attributes in the small circles (n), (d), (de), and (p), respectively (Figure 7.1).

So, to leave the user of CaVa free to use databases of any kind on any brand, a high-level abstraction layer over the database repository is needed, and the ontology plays that role.

[2] The inverse relations were not taken into account for the running example for the sake of simplicity

# Chapter 8

# Module A - CaVa$^{\text{settler}}$

The initial part of CaVa, shown in Figure 8.1, is defined by linking the *Institution Documents* and the *Ingestion Function* — DIS — for the purpose of collecting the documents (source) to be displayed in the final virtual LS.

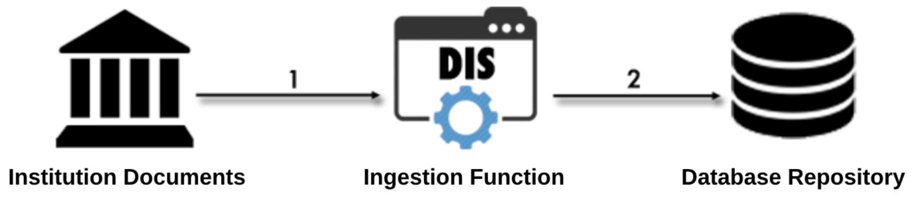


Figure 8.1: Module A of CaVa

The first important task to do in CaVa is access the sources, represented in Figure 8.1 by number (1), via a DIS, which, in turn is an application to aid and to intermediate the upload of the institution documents data into a database repository (relational, triple store, object oriented, etc.), represented by the edge (2). A DIS application can be any kind of system (e.g., a script, a web application, etc.) that performs that role (2). The database schema has the goal of storing the information extracted from the sources (archives, museums, libraries, etc.) through a DIS.

Notice that if the institution that retains the cultural documents already possess a *Database Repository*, this phase could be ignored.

After the population phase (Figure 8.1) is concluded, the ontology should be connected to the database repository to describe and relate semantic concepts of that specific domain, achieving its objective in this PhD research. Figure 8.2 depicts the mapping phase of the CaVa architecture. Notice that the *mapping* step, between the *Database Repository* and the *Ontology*, is optional, depending on the database type.

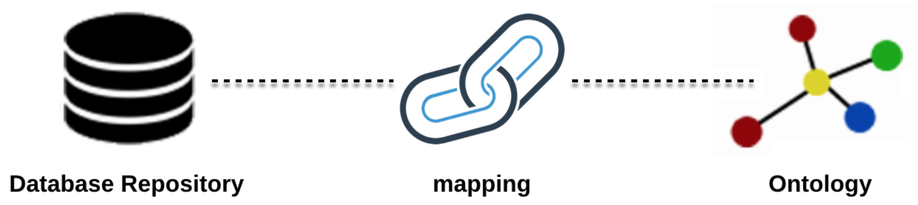


Figure 8.2: Bridging the gap between the *Database Repository* and the *Ontology*

To relate the semantic concepts of a particular ontology domain to the database repository, it is necessary to find out the type of the database. The mapping between the database and the ontology has the purpose of creating a pavement to establish a connection to query the repository based on the abstract level that the ontology's vocabulary provides. On one hand, if the digital repository is a triple store, it is already prepared to be queried via an ontology query language (e.g., SPARQL, SQWRL, etc.). On the other hand, if the database repository is a relational database or another kind of digital storage schema, it is necessary to link the ontology concepts and relations to the structure of that database. For example, if an object-oriented database is used, then it is necessary to map the classes and properties of the ontology to the classes and objects of the database. An approach to perform this kind of mapping is explained in [Bartalos and Bieliková, 2007].

If the database repository is a relational schema (most used nowadays), then the mapping between the ontology and the database should specify targets and sources. It means that the target (path on the ontology) needs to be specified to find the instances in that source (path on the database). Regardless of the database type used, the mapping phase allows the end-user

(Curator), to query the database repository through the ontology, leaving him/her more comfortable to work with abstract concepts, without the need to know the storage structure.

Some research has been done in this area, and the most known and used is R2RML[Das et al., 2012], a language for expressing customized mappings from relational databases to Resource Description Framework (RDF) datasets. Some frameworks have implemented the R2RML language, aiming at facilitating the use of the language through tools (e.g., Ontop[1], which is a platform to query databases as virtual graphs using SPARQL).

Continuing the running example about the "Museu Virtual Interativo da Fotografia", a relational database, called MVIF, was built. For this work, only a part of the database repository is shown, containing the tables "Periodo", and "Tecnica". A sample of the data contained in MVIF is shown in Tables 8.1 ("Periodo") and 8.2 ("Tecnica").

Table 8.1: "Periodo" table

| **Periodo** | |
|---|---|
| **id_per** | **per** |
| 1 | Período Pré-fotográfico |
| 2 | Período da fotografia Química |
| 3 | Período da fotografia Analógica |
| 4 | Período da fotografia Digital |

Table 8.2: "Tecnica" table

| **Tecnica** | | | | |
|---|---|---|---|---|
| **id_cat** | **cat** | **decada** | **desc** | **id_periodo** |
| 1 | Desenhos Fotogênicos | 1830 | Willian Henry ... | 2 |
| 2 | Daguerreótipo | 1830 | O Daguerreótipo ... | 2 |
| 3 | Fotografia com luz natural | 1830 | No início da foto ... | 2 |
| 4 | Longa Exposição | 1830 | Antes de ser uma ... | 2 |
| 5 | Positivo Direto | 1830 | O positivo direto ... | 2 |
| ... | ... | ... | ... | ... |
| 12 | Bokeh | 1920 | O Bokeh é a forma ... | 3 |
| ... | ... | ... | ... | ... |

[1] accessible at: `http://ontop.inf.unibz.it/`

Having the MVIF database and ontology (Chapter 7) presented, to relate the semantic concepts of the ontology and the relational database structure, with the purpose of creating a pavement to establish a connection to query the repository (represented by the data of Tables 8.1 and 8.2) based on the abstract level that the ontology's vocabulary provides, it is necessary a mapping between the two. For this purpose, Listing 8.1 shows a snippet of the mapping collection[2].

Listing 8.1: OBDA mapping file fragment – "*mvif.obda*"

```
[MappingDeclaration] @Collection [[

mappingId Technique
target :URI/Technique#{id_cat} a :Technique ;
        :has_period :URI/Period#{id_per} ;
        :has_name {cat} ;
        :has_decade {decada} ;
        :has_description {desc} .
source SELECT id_cat, id_periodo as id_per, cat, decada, desc FROM Tecnica, Periodo
        WHERE Tecnica.id_periodo = Periodo.id_per

mappingId Period
target :URI/Period#{id_per} a :Period ;
        :has_per {per} .
source SELECT id_per, per FROM Periodo

//other mapping rules ...
]]
```

The mapping collection of Listing 8.1 describes three ontological concepts of the running example. The first one is the "Technique" concept; according to the target statement, it encloses four relations (namely "has_period", "has_name", "has_decade", and "has_description") and three attributes: "cat" corresponding to the property "has_name", "decada" related to the property "has_decade", and "desc" associated to the property "has_description". The statement source (that actually realizes the mapping) is the SQL (Structured Query Language) command for querying the relational database retrieving the instances of "id_cat", "id_periodo", "cat", "decada", and "desc" fields from "Tecnica" table. Notice that the fields being selected in the SQL query are related to the placeholders[3] on the target clause.

[2] Notice that the mappings written for this running example were done with the aid of the Ontop framework based on OBDA axioms

[3] Placeholders (annotated as curly brackets {}) can be related to a literal (when it is only embraced with the curly brackets) or a relation (when it follows a pattern –

The second one is the "Period" concept, which, according to the target statement, contains only one attribute "per", corresponding to the property "has_per". The statement source is the SQL query to get the set of instances for the "id_per", and "per" fields from the "Periodo" database table. The third mapping is related to the "Pre_Photography" concept, a subclass of the "Period" class.

After explaining the role of the components of CaVa$^{\text{settler}}$ based on a relational database, imagine now that instead of having a relational schema, the database repository is a triple store. The difference is that in turtle (Listing 8.2), instead of placeholders, the literal value is embedded in the triples, the mapping to query the database not being necessary, because the data is already described in the same language as the ontology. Listing 8.2 presents the triples (in turtle syntax) generated for the running example.

Listing 8.2: Turtle triples file fragment

```
:Technique#1 a :Technique ;
        :has_period :Period#2 ;
        :has_name :"Desenhos Fotogênicos"^^xsd:string ;
        :has_decade 1830 ;
        :has_description :"Willian Henry ..."^^xsd:string ;

:Period#2 a :Period ;
        :has_per :"Período da fotografia Química"^^xsd:string ;

//other turtle instances declaration ...
```

The symbol colon (:), in both cases (Listings 8.1 and 8.2), is an alias that denotes the default prefix of the running example ontology (http://semanticweb.org/rgm/2018/mvif/).

The processing of such stored data will be seen in the following chapters. Chapter 10 will describe CaVa$^{\text{gen}}$ and the other components related to the mentioned task. The following chapter presents CaVa$^{\text{DSL}}$, a language to specify the virtual Learning Spaces.

concept+literal like in :URI/Period#{id_per}, which means the pattern will be a new rule on the OBDA model)

# Chapter 9

# CaVa$^{\text{DSL}}$– Learning Space Specification

Several applications are based on Domain-Specific Languages. They provide the right terminology to a peculiar problem/subject, because they use a particular domain vocabulary that defines abstract concepts, different from general-purpose languages.

Aiming at an easy generation of virtual Learning Spaces for the use of the person in charge of institutional archives or museums, an external domain-specific language was idealized and developed, called CaVa$^{\text{DSL}}$, to describe, in an abstract level, virtual exhibition rooms in the museum Curator's perspective, giving the Curator the possibility to specify the virtual LS upon a domain ontology vocabulary.

In accordance with Cadavid et al. thoughts, the web application assemble is a complex process, which demands a big effort to get several tasks done [Cadavid et al., 2009]. Even today, with powerful tools aiming at creating web applications (in this PhD work, learning spaces), a domain-specific language that empowers domain experts to specify virtual environments in their terminology, is an essential resource, easier than requesting the domain specialist to specify them in a general-purpose programming language. Most of the time, the domain expert should know or learn more than one general-purpose language to get a virtual LS done and working.

Thus, taking advantage of the development of domain-specific languages (Section 4.4) and following the design guidelines outlined in Section 4.3, CaVa$^{\text{DSL}}$ was created in order to foster the Cultural Heritage Curator to specify a virtual Learning Space, letting the CaVa$^{\text{processor}}$ conduct the automatic creation of that LS. CaVa$^{\text{DSL}}$ focuses on the exhibition rooms, so the main component of the language is a list of exhibitions – in the context of this PhD research, the term exhibition refers to "the outcome of the action of displaying something" [Desvallées, 2010]. According to Davallon, exhibition "means the act of displaying things to the public, the objects displayed (the exhibits), and the area where this display takes place" [Davallon, 1986]. The CaVa$^{\text{DSL}}$ syntax is presented in Section 9.1.

## 9.1 CaVa$^{\text{DSL}}$ Syntax

The syntax of CaVa$^{\text{DSL}}$ is similar to that of the JSON (JavaScript Object Notation) language, because it is based on the 'key-value pair' format and it is easy for humans to read and write. The structure of CaVa$^{\text{DSL}}$ is split into four main blocks that specify: the main configuration, the header, the content, and the footer of the virtual Learning Space, as specified by the derivation rule p0.

```
[p0] cava:  mainConfig header content footer ;
```

Next, the CaVa$^{\text{DSL}}$ specification will be presented, based on the MVIF running example, following the rules of CaVa$^{\text{grammar}}$[1].

[1] In this chapter, only a part (main productions rules) of CaVa$^{\text{grammar}}$ for the understanding of the chapter and the given examples is presented. Appendix A.1 shows the entire CaVa$^{\text{grammar}}$.

### *mainconfig*

Defines the virtual Learning Space title and main description, as well as, other components (e.g., carousel of images) related to the entire LS (not only about a specific page, like an exhibition, for example), and it is specified by the production rule p1:

```
[p1] mainConfig:  'mainconfig' '[' learningSpaceTitle learningSpaceAbout?
                                   learningSpaceCarousel?
                               ']' ;
```

To demonstrate how the production rule p1 is applied, an example is presented:

```
mainconfig [
          LS title:  "Museu Virtual Interativo da Fotografia",
          about [
                p:  "O museu virtual da fotografia se propõe a
                    organizar as informações históricas dentro de
                    temáticas curatoriais apresentadas em cronologia.",
                p:  "Partimos de uma convicção:  não é possível
                    compreender a importância, as possibilidades,
                    o presente e o futuro da fotografia sem compreende
                    as ideias e os conceitos, os equipamentos e as
                    tecnologias, as práticas e os usos, os personagens e
                    suas trajetórias pelos quais e por onde a fotografia
                    passou para chegar onde chegou nos dias de hoje.",
          ]
          carousel [
               interval:  5,
               images [
                   caption:  "Author - Ricardo Ravanello",
                   src:  "imagem-capa-mvif.png", active,
                   caption:  "Fornalha de um Hammam -
                             Marrocos, 2015 (Ricardo Ravanello)",
                   src:  "imagem2-mvif.jpg",
               ]
          ]
]
```

### *menu*

Defines the main menu of the virtual Learning Space, as specified by the production rule p2. The menu is composed of:

- brand, background and foreground colors;
- behaviour (if the menu should be fixed or if it should follow the scrolling of the page);
- type of the menu links (simple or dropdown), containing the label and the link.

```
[p2] header:  'menu' '[' (optionHeader)+ ']' ;
     optionHeader:  brand | backgroundColor | fontColor
                    | behaviourStat | items ;
```

To demonstrate how the production rule `p2` is applied, an example is presented:

```
menu [
    brand:  "Museu Virtual Interativo da Fotografia",
    background color:  green,
    foreground color:  white,
    behavior:  fixed,
    options [
            label:  "Exibições", dropdown [
                 dropdown label:  "Permanentes", url:  "permanentes",
                 dropdown label:  "Temporárias", url:  "temporarias",
                 dropdown label:  "Especiais", url:  "especiais",
                 dropdown label:  "Futuras", url:  "futuras",
            ]
            label:  "Temáticas", dropdown [
                 dropdown label:  "Técnicas Fotográficas", url:  "tecnicas",
                 dropdown label:  "Equipamentos", url:  "equipamentos",
                 dropdown label:  "Evolução Conceitual", url:  "evolucao",
                 dropdown label:  "Tipos de Intervenção", url:  "tipos",
            ]
            label:  "Sobre", url:  "sobre_mvif", extension:  php,
    ]
]
```

### *exhibitions*

A list of exhibitions, specified according to the production rule `p3`. Each exhibition is composed of:

- title, short description and icon;
- additional info with a title and a description;
- behaviour (if the component of the list should stay collapsed or expanded);
- exhibition type (must be one of the following: permanent, temporary[2], future, or special).
- query operator (specified according to the production rule `p4`):
  - all (search for all occurrences of a determined ontology concept declared and returns the set of instances);
  - one (search for only one object that corresponds to the conditional parameter and the ontology concept. It returns the first instance).
- SPARQL query (specified according to the production rule `p5`).

```
[p3] exhibitions:  ‘exhibitions’ ‘[’ (exhibition)+ ‘]’ ;
     exhibition:  ‘exhibition’ ‘[’ (optionExhibition)+ ‘]’ ;
     optionExhibition:  exhibitionTitle
                      | exhibitionShortDescription
                      | exhibitionIcon
                      | exhibitionBehaviour
                      | exhibitionAdditionalInfo
                      | exhibitionType
                      | exhibitionNotification
                      | (queryOperators | sparql) ;
```

The `queryOperators` rule is responsible for describing the operator to query the database repository through the ontology. It is specified by the production rule `p4`:

[2] If the type is set up to ‘temporary’, it is possible to configure a notification to the visitor of the LS based on the exhibition expiration date;

```
[p4] queryOperators:  all | one ;
     all:  CONCEPT '->' 'all' '(' parametersAll ')' labelsOptions ;
     parametersAll:  listName ',' mappingOrTriplesFileName ','
                     ontologyFileName ;
     listName:  TEXT ;
     mappingOrTriplesFileName:  TEXT ;
     ontologyFileName:  TEXT ;
     labelsOptions:  '[' (labelsExhibitionRoom)+ ']' ;
     labelsExhibitionRoom:  elem (',' elem)* ;
     elem:  TEXT ;
```

`sparql` is the production rule in charge to query the database repository through the ontology via the SPARQL query language. It is described according to the rule `p5`:

```
[p5] sparql:  'SPARQL' '[' sparqlStatement ']' '[' labelsOptions ']' ;
```

Notice that the production rule `sparqlStatement` is a declaration based and verified by the SPARQL grammar.

To demonstrate how the production rules `p3` and `p4` are applied, an example is presented. Notice that the lexer rule `CONCEPT` shall be related to a class of the used ontology.

```
exhibitions [
       exhibition [
             title:  “Técnicas da Fotografia”,
             short description:  “Dividimos a história da fotografia
                                 em três grandes períodos.  Essa classificação
                                 se justifica não apenas pela mudança dos
                                 suportes ou das técnicas, mas
                                 também pelo fato de que é possível
                                 diferenciar drasticamente todo o sistema
                                 em torno da fotografia em cada
                                 um dos três momentos.”,
             icon:  “camera-retro”,
             additional info [
                        title:  “1672-2010”,
                        description:  “Período”,
             ]
             behavior:  expanded,
             type:  permanent,
             Technique->all(“Técnicas”, “mvif.obda”,
                            “http://semanticweb.org/rgm/2018/mvif/”)
                            [headerOfEachElement:“Técnica”, “Década”,
                            “Descrição”, “Período”],
       ],
       # other exhibitions . . .
]
```

### *footer*

Defines an area at the bottom of the LS. It is defined by the production rule p6. The footer contains:

- images and date;
- company or developer name;
- behaviour (if the footer should be fixed or if it should follow the scrolling of the page);
- style (if the footer should be simple with the data above mentioned or it should be extended, having an array of links with title, subtitle, URL (Uniform Resource Locator), icon, etc). The extended footer is good for addresses, social network links and other important information related to the virtual Learning Space.

```
[p6] footer:  'footer' '[' (optionFooter)+ ']' ;
     optionFooter:  footerImage | footerFormatDate | footerDeveloper
                 | footerBehavior | footerStyle ;
```

An example illustrating the production rule p6 is presented:

```
footer [
      images [
             image:  "cava_logo.png",
             alignment:  right,
      ]
      format date:  "Y",
      developer [
                name:  "Ricardo Martini",
                alignment:  left,
      ]
      behavior:  fixed,
      style:  condensed,
]
```

Notice that each block and component specification starts with the left bracket "[" and closes with the right bracket "]", always embracing pairs consisting of plain text or built in terms (e.g., the exhibition's *behavior* attribute can be set up as 'collapsed' or 'expanded' values only).

To accomplish the given example, Chapter 11 presents the final version of the virtual Learning Space related to the "Museu Virtual Interativo da Fotografia", according to the rules of CaVa$^{\mathrm{grammar}}$. In other words, Chapter 11 will present the rendering of each of the four blocks of CaVa$^{\mathrm{DSL}}$. Chapter 10 describes the process of recognizing the CaVa$^{\mathrm{DSL}}$ specification and generating the LS Scripts according to that Learning Space specification.

# Chapter 10

# Module B - CaVa$^{\text{processor}}$

The application generation process from a formal specification comprises the stage of transforming an input (formal specification itself) into one or more outputs (application code). This means that to interpret and translate the formal specification (CaVa$^{\text{DSL}}$) into the application code (all the scripts or programs) needed to build a web-application[1], a transformation phase will be necessary, which is one of the components of a compiler.

A domain-specific language compiler holds the same internal structure of a regular compiler, with a front-end, an intermediate representation, and a back-end component (Figure 10.1).

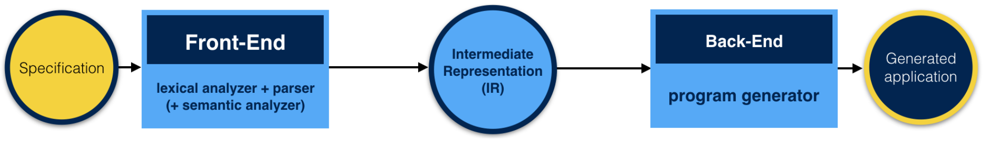


Figure 10.1: Block Diagram of a Compiler

The front-end, composed of the lexical, syntactic, and semantic analyzers receives the formal specification (in this case, a virtual LS description written in CaVa$^{\text{DSL}}$) as input, parsing and mapping it to an Intermediate Representation (IR).

[1] e.g., scripts in PHP, CSS, JS, Smarty template, HTML, etc.

The Intermediate Representation, usually a complex data structure, must not lose the meaning of the initial formal specification.

The back-end component receives an IR as input. From this IR, it is responsible for the generation of the final result (e.g., executable application).

Summing it up, according to Figure 10.1, from a *formal specification*, through the analyzers of the *front-end*, an *IR* is generated. In the next step, the *back-end* receives as input the generated *IR* and produces an *application*, which is written in a cocktail of languages (see footnote 1). When an application is well-defined (structure and behavior), it can be automated [Krstićev et al., 2016]. So, in this work, a set of application generators to automatically create virtual Learning Spaces based on the CaVa$^{\text{DSL}}$ specifications and ontologies was built.

This chapter presents the CaVa architecture's core. The CaVa$^{\text{processor}}$ contains the main components to achieve the objective outlined in this PhD work.

The CaVa$^{\text{processor}}$ consists of a set of processors (CaVa$^{\text{gen}}$) that deal with CaVa$^{\text{DSL}}$ and generate virtual Learning Spaces, making available the navigation over important and real information contained in archival documents to the public through virtual museums.

To parse CaVa$^{\text{DSL}}$ in order to produce the right configuration and script files understandable by the web browser with the objective of rendering the virtual Learning Spaces, this chapter presents a group of processors (CaVa$^{\text{gen}}$) that receive one or more input files and generate a set of files as output, like the view templates, server- and client-side files, webpage presentation style and the static content of the virtual LS.

CaVa$^{\text{processor}}$ has two objectives: (1) parsing the specification (manually written by the Curator or the CH institution) of a virtual LS based on CaVa$^{\text{DSL}}$, which is defined by CaVa$^{\text{grammar}}$; and (2) generating/setting up the LS scripts component of CaVa$^{\text{render}}$, that comprises the configuration and script files necessary to be rendered by the web browser to produce the final objective (the virtual Learning Space).

Figure 10.2 shows a simple CaVa workflow involving the three steps that shall be followed to achieve the main goal of this work.

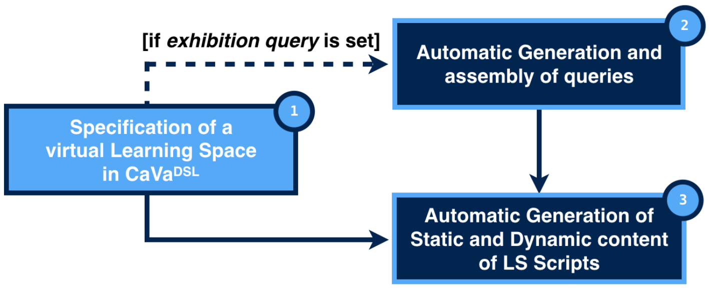

[if *exhibition query* is set]
2
Automatic Generation and assembly of queries
1
Specification of a virtual Learning Space in CaVa^DSL
3
Automatic Generation of Static and Dynamic content of LS Scripts

Figure 10.2: CaVa workflow: from the CaVa$^{\text{DSL}}$ Specification to the virtual LS automatic generation in three steps

- Step 1 (*Specification of a virtual Learning Space in CaVa$^{DSL}$*): the first task is to specify the virtual LS (this specification shall be done by the CH institution responsible), because it is from the virtual LS specification that CaVa$^{\text{gen}}$ advances to the other stages. Step 1 was described in Chapter 9;

- Step 2 (*Automatic generation and assembly of queries*): when a query is specified in CaVa$^{\text{DSL}}$, it is in this phase that the processor automatically creates the queries. CaVa$^{\text{gen}}$ generates, in an automatic way, ontology queries (e.g., SPARQL, SQWRL) based on that queries grammar (e.g., SPARQL, SQWRL grammar).

- Step 3 (*Automatic generation of static and dynamic content of LS Scripts*): it comprises the automatic creation of the configuration and script files. The configuration files are those that set up the whole virtual LS with component parameters and libraries needed to successfully run the project in the web browser. Besides, in this phase three kinds of files are created:

  – the static structure: view templates and server-side files (e.g., header, content, footer, etc.);

- the webpage presentation style and client-side files (e.g., Cascading Style Sheets – CSS, javascript – JS, etc.);
- the virtual LS webpage static content (e.g., plain text, images).

Moreover, the outcome of step 2 (query results) are used, as input, in step 3 to produce and to populate the client- and server-side files (for instance, the museum exhibitions' content).

From step 1, if there is at least one query operator specified at any exhibition room of the CaVa$^{\text{DSL}}$ specification, step 2 is executed, followed by step 3; otherwise, from step 1, step 3 is executed (excluding step 2 of running), generating only the static content of the LS. After executing these steps, the web browser (comprised in CaVa$^{\text{render}}$) will receive the necessary script files aiming at interpreting and rendering the final virtual Learning Space specified by the Curator of the Cultural Heritage institution. CaVa$^{\text{gen}}$ is presented in Section 10.1.

## 10.1 CaVa$^{\text{gen}}$

The CaVa$^{\text{gen}}$ set consist of four processors, namely CaVa$^{\text{structure}}$, CaVa$^{\text{queries}}$ (CaVa$^{\text{queriesTriple}}$ for triples, instead of mappings), and CaVa$^{\text{run}}$. The first one is responsible for the generation of the static content of the virtual LS. Moreover, CaVa$^{\text{structure}}$ has the task of executing the CaVa$^{\text{queries}}$ or CaVa$^{\text{queriesTriple}}$ processor (if at least one query operator is set up in the CaVa$^{\text{DSL}}$ Specification). The second and third one have the duty of assembling the ontology queries based on the query operator(s) specified in the CaVa$^{\text{DSL}}$ description. They shall handle the mapping or other intermediate file that links the ontology to the database (e.g., an OBDA mapping file, a Terse RDF Triple (turtle) file, etc). The last one, CaVa$^{\text{run}}$, is responsible for executing the queries mounted by CaVa$^{\text{queries}}$ or CaVa$^{\text{queriesTriple}}$ Processor and generates the queries results file to be used by the LS Scripts.

### CaVa$^{\text{structure}}$

The purpose of CaVa$^{\text{structure}}$ is to get the LS Specification (CaVa$^{\text{DSL}}$) as input and transform it into several web languages scripts (e.g., HTML, PHP, JS, template engines, CSS, etc.) and other kind of documents (e.g., state files[2]), that together make up multiple web pages, i.e., the complete virtual Learning Space. Figure 10.3 presents the CaVa$^{\text{structure}}$ schema.

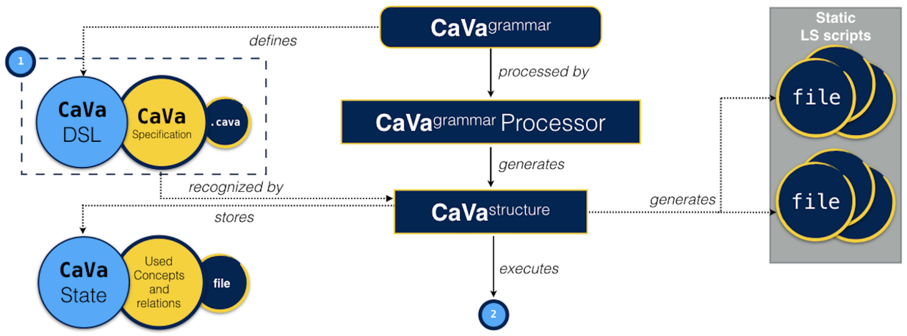


Figure 10.3: CaVa$^{\text{structure}}$ processor schema

The dashed rectangle represented in Figure 10.3 by number (1) is related to the step 1 of the CaVa workflow (Figure 10.2). This means that the circles *CaVa$^{DSL}$*, *CaVa Specification* and *.cava* are a specification file describing a virtual Learning Space according to the rules of CaVa$^{\text{DSL}}$ (Chapter 9), which is defined by *CaVa$^{grammar}$*, that in turn is processed by the CaVa$^{\text{grammar}}$ Processor. The duty of the CaVa$^{\text{grammar}}$ Processor is the processing of the grammar (CaVa$^{\text{grammar}}$) and the generation of the CaVa$^{\text{structure}}$ skeleton.

CaVa$^{\text{structure}}$ extracts the ontology concepts used in the *.cava* specification file and stores them in a *CaVa State* file (e.g., plain text, JSON, etc.), aiming at recognizing them in the mappings at step 2 (Figure 10.2) of the CaVa workflow, which generates and assembles the queries.

The skeleton generated is related to the actions that the parser must take when recognizing the parts of the input (CaVa$^{\text{DSL}}$ Specification).

[2] State files contain data to be used by the processors of CaVa system in order to retrieve important information to proceed with the execution and generation of the virtual LS.

If the parser generator is, for example, ANTLR, then the skeleton could be a set of listeners or visitors methods[3]. The skeleton that ANTLR generates is a Java Interface and a Java Class (empty implementation of all productions of CaVa$^{\text{grammar}}$ as methods, that implements the generated Java Interface). The empty class is extended according to the project needs to create a new class with the parser actions.

The listener implementation is responsible for the generation of the final virtual Learning Space. Listing 10.1 shows some listeners signature to illustrate the enter and exit events associated with the CaVa$^{\text{grammar}}$ nonterminals `cava`, `mainConfig`, `header`, and `all`.

Listing 10.1: Java Interface with listeners signatures based on CaVa$^{\text{grammar}}$ productions

```java
public interface CavaListener extends ParseTreeListener {
    void enterCava(CavaParser.CavaContext ctx);
    void exitCava(CavaParser.CavaContext ctx);
    void enterMainConfig(CavaParser.MainConfigContext ctx);
    void exitMainConfig(CavaParser.MainConfigContext ctx);
    void enterHeader(CavaParser.HeaderContext ctx);
    void exitHeader(CavaParser.HeaderContext ctx);
    void enterAll(CavaParser.AllContext ctx);
    void exitAll(CavaParser.AllContext ctx);
    //other signatures ...
}
```

To better understand how CaVa$^{\text{structure}}$ works, a practical example to generate the menu of the running example about the MVIF museum is shown. The output language chosen for this example is PHP.

The result of executing all listeners related to the generation of the LS menu is shown in Listing 10.2, configuring that output as the whole “header.php” script file.

Listing 10.2: Generated PHP code for creating the MVIF menu according to the CaVa$^{\text{DSL}}$ specification

```php
<?php
    $data = array(
        'brand'=>"Museu Virtual Interativo da Fotografia",
        'bgColor'=>"green",
        'fontColor'=>"white",
        'behaviour'=>"fixed",
```

[3] Listeners and visitors are objects that respond to rule entry and exit events for each nonterminal of the grammar.

```
        'options'=>array(
            array('label'=>"Exibições", 'dropdown'=>"true",
                'dropdownListItems'=>array(
                    array('labelDropDown'=>"Permanentes", 'urlDropDown'=>"permanentes"),
                    array('labelDropDown'=>"Temporárias", 'urlDropDown'=>"temporarias"),
                    array('labelDropDown'=>"Especiais", 'urlDropDown'=>"especiais"),
                    array('labelDropDown'=>"Futuras", 'urlDropDown'=>"futuras"),
                ),
            ),
            //dropdown ''Temáticas'' ...
            array('label'=>"Sobre", 'url'=>"sobre_mvif", 'dropdown'=>"false"),
        ),
    );
```

Besides the generation of static LS scripts, the execution is then passed to the CaVa$^{\text{queries}}$ or CaVa$^{\text{queriesTriple}}$ processors (notice that node number (2) in Figure 10.3 expands to the block diagram in Figure 10.4), to assembly the query statements. After that, the execution is returned to CaVa$^{\text{structure}}$ to proceed with the creation of the remaining files (namely, the exhibition room files), using the information generated in step 2 to populate the template files in the exhibition rooms. The automatic generation of the exhibition rooms by CaVa$^{\text{structure}}$ is shown in the CaVa$^{\text{run}}$ section, after the execution of the queries.

The output of CaVa$^{\text{structure}}$ is a set of static LS scripts (e.g., *.php*, *.html*, *.css* files), some of them related to the server side, and other to the client side.

### CaVa$^{\text{queries}}$ and CaVa$^{\text{queriesTriple}}$

CaVa$^{\text{gen}}$ has, besides the generator of static content (CaVa$^{\text{structure}}$), two processors to automatically create dynamic content based on the input file type. This means that a method that figures out the input file format exists, and depending on this, it is possible to call the right processor (CaVa$^{\text{queries}}$ or CaVa$^{\text{queriesTriple}}$) to recognize the intermediate (e.g., mapping or triples specification) source code file.

Figure 10.4 presents the whole schema for the two options (triple store or other kind of storage e.g., relational database, object-oriented database, etc.).

The CaVa$^{\text{queries}}$ processor deals with any kind of storage, except triple stores, which is managed by the CaVa$^{\text{queriesTriple}}$ processor.

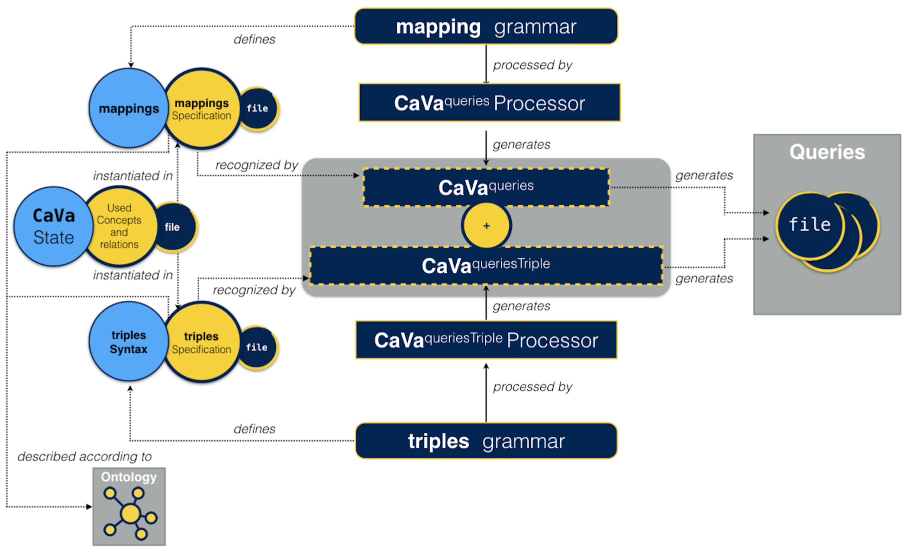


Figure 10.4: CaVa$^{queries}$ and CaVa$^{queriesTriple}$ processors

When CaVa$^{structure}$ parses and identifies the query operators in a CaVa$^{DSL}$ specification, a method is called, to choose and invoke the right query assembler processor. The input file must be written according to the ontology's description. This means that the specification file must include only concepts and properties that belong to the used ontology. For the running example, the ontology used is the one specified in Chapter 7, about the MVIF museum.

The approach to solve the automatic generation of queries reuses well-defined grammars (e.g., for the triple store, grammars like RDF, Turtle, etc.). This ensures that each token of the input file will be recognized based on the original grammar rules.

To generate the right query aiming at getting the database instances through the ontology, to populate the exhibition rooms of the virtual LS, five steps, performed by CaVa$^{queries}$ or CaVa$^{queriesTriple}$, have to be executed:

1. Searches for occurrences of the ontology concept (like in the statement `Technique->all()`) in the intermediate file (e.g., OBDA mapping, RDF triples file, etc.);

2. Expands each mapping axiom related to the query imposed in the CaVa$^{\text{DSL}}$ specification (like in `Technique->all()` statement) – it is possible due to the CaVa State file that stores the concept appointed in the operator of CaVa$^{\text{DSL}}$, in the case of the running example, *all()*, selecting only those related to the concept that the Curator wants;

3. Stores all the mapping axioms related to the desired query, expanded in task 2;

4. Mount the query statement. For example, to SPARQL queries and OBDA mapping file, an approach could be the transformation of each placeholder related to a literal (i.e. those annotated only as curly brackets {}) in a *SELECT* clause variable with name ?p__0, ?p__1, depending how many literals are found; Also transforms each placeholder (*concept+literal*) in a new variable of the *WHERE* clause;

5. Creates and writes the generated query file (e.g., ".rq").

To explain how the automatic generation of the queries could be made in CaVa, based on these five steps, the use of the running example is taken. The approach here explained is based on the use of the OBDA mapping file ("mvif.obda") described in Listing 8.1. The desired output is a SPARQL query (".rq") file.

First of all, the concept declared in the example shown in Section 9.1 (`Technique->all(‘‘params’’)[‘‘list of attributes’’]`), when recognized by the application generator in charge, is stored in a CaVa State file to be searched in the mapping axioms of the entire mapping file ("*mvif.obda*" – parameter of the operator *all()*).

After finding the mapping axioms related to that concept, which must exist in the ontology passed in the third parameter of the operator *all()*, the expansion should be done, visiting each placeholder in the format of *concept + literal*.

So, being the *Technique* concept the main concept in the CaVa$^{\text{DSL}}$ specification, when the application generator detects a pattern like ":Period#{id_per}", this means that a new mapping rule with the *Period* concept is declared in the mapping file, so it needs to be stored, as explained in task 3 of the five steps. The search for new patterns continues, trying to find and expand new mapping rules.

Besides the expansion of each pattern found, every placeholder should be transformed into a variable in the SPARQL query. A SPARQL query is basically composed of two parts: the *SELECT* clause, which identifies the variables to appear in the query results; and the *WHERE* clause, which defines where to find values for the variables defined in the *SELECT* clause. So, the placeholders related to a literal should be transformed into a variable of the *SELECT* clause. For the running example, the literal {cat} is transformed into a variable with the identification ?p__0. The next literal found shall be identified as ?p__1 and so on.

The variables of the *WHERE* clause, which forms the triples (graph pattern), are identified by the application generator. The name of the variables are given based on the identification of the placeholder pattern (the part between ":" and "#" for example). To exemplify how the variable name is given, from the pattern ":Period#{id_per}", the variable "?Period" is defined, as the variable "?Pre_Photography" is determined from the ":Pre_Photography#{id_per}" pattern.

To finish, the CaVa$^{queries}$ processor creates and writes the ".rq" file containing the prefixes[4] and clauses (SELECT and WHERE). In Listing 10.3 the generated SPARQL query is presented.

Listing 10.3: Generated query file fragment – "*queryAll1.rq*"

```
PREFIX    :  <http://semanticweb.org/rgm/2018/mvif/>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>

SELECT ?p___0,?p___1,?p___2,?p___3
WHERE {
    ?Technique a :Technique ;
    :has_period ?Period ; :has_name ?p___0 ;
    :has_decade ?p___1 ; :has_description ?p___2 .

    ?Period a :Period ;
```

[4] The prefixes in the OBDA file are taken from a section called [prefixDeclaration].

```
        :has_per ?p___3 .

        //other graph pattern declarations (triples)...
        //...  according to the mapping axioms
    }
```

In order to execute the generated query (Listing 10.3), the CaVa$^{\text{run}}$ processor of CaVa$^{\text{gen}}$ is necessary. It is explained next.

### CaVa$^{\text{run}}$

CaVa$^{\text{run}}$ is a query answering system[5] that handles the output of the CaVa$^{\text{queries}}$ and CaVa$^{\text{queriesTriple}}$ processors. Figure 10.5 lays out the schema to execute the queries in the CaVa system.

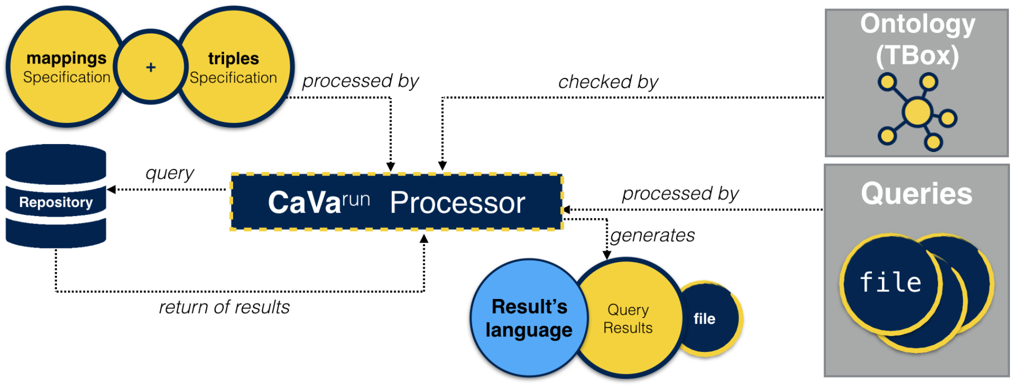


Figure 10.5: CaVa$^{\text{run}}$ schema

To achieve the results of the generated query by CaVa$^{\text{queries}}$ or CaVa$^{\text{queriesTriple}}$, CaVa$^{\text{run}}$ needs to perform six tasks:

1. Receives as input the mapping or other intermediate file (mappings or triples specification);

2. Receives as input the ontology (e.g., an external IRI (International Resource Identifier)). More specifically, the terminological components of

[5] Some of the most known systems for this purpose are Virtuoso, D2RQ, Triplify, etc.

the ontology (TBox) like classes and properties (the controlled vocabulary);

3. Prepares the configuration for the ontology reasoner instance, depending on the Database Repository type;

4. Creates the instance of the ontology reasoner. For example, if the Database Repository is a relational schema, it is necessary to set up the connectivity driver (e.g., JDBC);

5. Prepares the data connection for querying and executes the query;

6. Writes the query execution result into a file to be read by the specific exhibition room that called the query operator in CaVa$^{\text{DSL}}$ specification.

After the execution of these six steps, the generated exhibition room shall deal with the query result file in order to present the result set in the final view (exhibition room rendered).

The application of these six steps for the running example about the MVIF museum is demonstrated applying the following configuration. Notice that as the Ontop Framework is applied, the Quest query answering system, explained in [Rodríguez-Muro et al., 2012], performs the role of CaVa$^{\text{run}}$.

1. Receives as input the mapping axioms based on the OBDA model (Ontop is engaged to aid in the mapping between the relational database and the ontology) as described in Listing 8.1 (“mvif.obda”);

2. Receives the MVIF ontology described in Figure 7.1 using the IRI `http://semanticweb.org/rgm/2018/mvif/`;

3. Prepares the configuration for the Quest instance. A “virtual ABox mode”[6] can be used.

[6] Read more at: `https://github.com/ontop/ontop/wiki/ObdalibQuestIntro`

4. Creates the instance of the ontology reasoner (Quest). It is based on the OBDA model of step 1, the ontology of step 2, the configuration of the Quest instance of step 3, and the connectivity driver of the relational database (e.g., MySQL – com.mysql.jdbc.Driver);

5. Prepares the data connection for querying (database credentials like user, host, port, and password) and the generated SPARQL query of Listing 10.3;

6. Writes the query execution result in a JSON file.

The JSON result file generated is presented in Listing 10.4.

Listing 10.4: Snippet of JSON result file generated by CaVa$^{\text{run}}$– "*queryAll1.json*"

```
{
    "0": {
        "0": "Desenhos Fotogênicos",
        "1": "1830",
        "2": "Willian Henry ...",
        "3": "Período da fotografia Química"
    },
    "1": {
        "0": "Daguerreótipo",
        "1": "1830",
        "2": "O Daguerreótipo...",
        "3": "Período da fotografia Química"
    },
    //other JSON objects ...
}
```

To achieve the result shown in Listing 10.4, the exhibition room generated shall call the command to execute the query generated by CaVa$^{\text{queries}}$ or CaVa$^{\text{queriesTriple}}$.

In this stage, CaVa$^{\text{structure}}$ gets back the flow of execution and generates the exhibition room server-side files (for the running example, a single file in PHP called "exhibition1.php") consisting of all data necessary to be rendered. The exhibition room file related to the running example presented is shown in Listing 10.5.

Listing 10.5: Automatically generated PHP code for an exhibition room – "*exhibition1.php*"

```php
<?php
    $sparqlQuery = "queryAll1.rq";
    $sparqlResult = "queryAll1.json";
    $mappingOrTriplesFile = "mvif.obda";
    $IRIOntology = "http://semanticweb.org/rgm/2018/mvif/";
    $jarFilePath = ".../mappingOnto2Database.jar ";
    shell_exec($jarFilePath . " " . $sparqlQuery . " " . $sparqlResult
                    . " " . $mappingOrTriplesFile . " " . $IRIOntology);
    $json = file_get_contents($sparqlResult);
    $data = json_decode($json, TRUE);
    $data ['labels'] = array(
        0 => "Técnica",
        1 => "Década",
        2 => "Descrição",
        3 => "Período",
    );
    $data ['collapsed'] = "expanded";
    $tpl = new SMTemplate();
    $tpl->render('exhibition1', $data);
```

The code of the"*exhibition1.php*" file basically (a) gets as input: (1) the generated SPARQL query "*queryAll1.rq*"; (2) a file ("*queryAll1.json*") to store the result of the execution of the query; (3) the mapping or triples file ("*mvif.obda*"); (4) the IRI ontology ( *"http://semanticweb.org/rgm/2018/mvif/"*); and (5) the jar file that effectively performs the mapping and the execution of the generated query of step (1) – CaVa$^{\text{run}}$ performed by Ontop (Quest reasoner); (b) executes the *shell_exec()* command with the inputs; (c) gets the content of the JSON result file ("*queryAll1.json*"), which was filled by the executed jar; (d) decodes and stores the JSON result file content in the variable *$data*; (e) stores some configurations related to the exhibition room, as the labels chosen in the operator *all()* and sets a flag to define if the User Interface (UI) component (in this case, an accordion) will appear expanded or collapsed; and (f) finally renders the exhibition room from the *$data* variable content.

Notice that the exhibition room code calls the command to execute the query (the line with *shell_exec(...)* command), every time the page is requested.

The next section will describe CaVa$^{\text{render}}$, responsible for rendering the final virtual Learning Spaces.

# Chapter 11

# Module C - CaVa$^{\text{render}}$

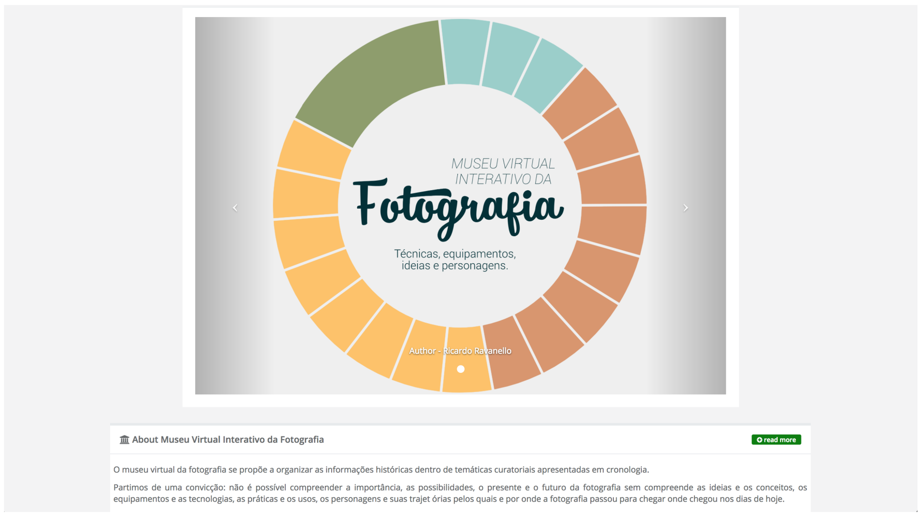
As aforementioned, a virtual Learning Space is basically a web environment where the information available about a specific domain is disposed for the users to learn the content displayed in a friendly and interesting way. Thus, the last block of the architecture aims at generating, from the LS scripts, a virtual Learning Space which is a web page that contains all the information that is desired to be shown to the visitors.

The LS scripts generated by CaVa$^{\text{structure}}$, CaVa$^{\text{queries}}$ or CaVa$^{\text{queriesTriple}}$, and CaVa$^{\text{run}}$, describe the intended LS in a less abstract level, more precisely in a programming language style. They state how to get the instances in the database repository with the actual data that shall be displayed in the final virtual Learning Environment. After that, a browser must interpret the LS Scripts to exhibit the web page that constitutes the final virtual Learning Space.

So, this chapter shows each stage of the running example described, presenting some images related to the LS Scripts, which constitute the final virtual LS.

Through the CaVa$^{\text{DSL}}$ Specification (Chapter 9), LS Scripts were automatically generated. In Section 9.1, the virtual LS was specified part by part. So, based on the Section 9.1 organization (production rules and specification examples) and Chapter 10 LS Scripts, the rendering parts of the entire virtual Learning Space are shown.

The main configuration become rendered as presented in Figure 11.1.


MUSEU VIRTUAL
INTERATIVO DA
Fotografia
Técnicas, equipamentos, ideias e personagens.
Author - Ricardo Ravanello
read more

About Museu Virtual Interativo da Fotografia

O museu virtual da fotografia se propõe a organizar as informações históricas dentro de temáticas curatoriais apresentadas em cronologia.

Partimos de uma convicção: não é possível compreender a importância, as possibilidades, o presente e o futuro da fotografia sem compreende as ideias e os conceitos, os equipamentos e as tecnologias, as práticas e os usos, os personagens e suas trajet órias pelos quais e por onde a fotografia passou para chegar onde chegou nos dias de hoje.

Figure 11.1: Main configuration rendered

Figure 11.1 shows the main components (LS title, about and carousel) as specified in MVIF CaVa$^{\text{DSL}}$ Specification running example for the whole LS.

The MVIF menu becomes rendered, based on the code of Listing 10.2, as presented in Figure 11.2.

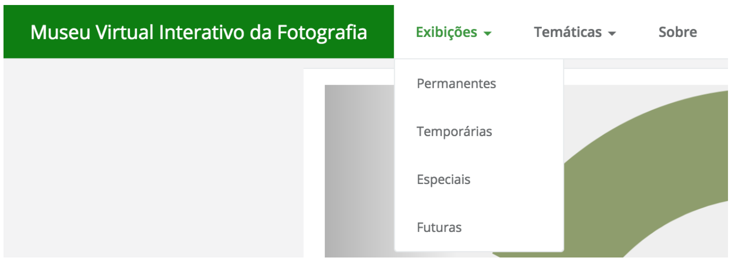
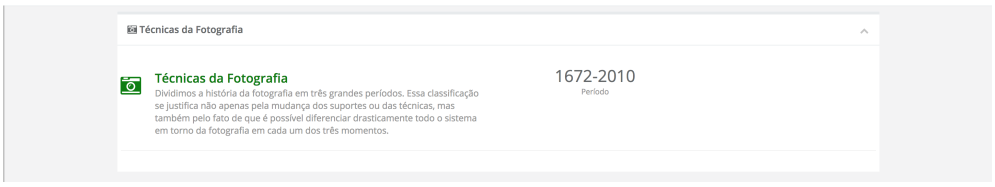


Figure 11.2: MVIF Menu rendered

The MVIF menu, according to the CaVa$^{\text{DSL}}$ Specification, has a "brand", that configures the title "Museu Virtual Interativo da Fotografia", and three menu options. Two of them are dropdown lists ("Exibições" and "Temáticas") and one is a simple menu ("Sobre").

The exhibitions are separated by type. As only one "permanent" exhibition was specified in the running example, it appears in the dropdown menu option "Exibições−>Permanentes". Figure 11.3 shows the exhibition room according to the CaVa$^{\mathrm{DSL}}$ Specification about the exhibitions part in Section 9.1 and other LS Scripts.

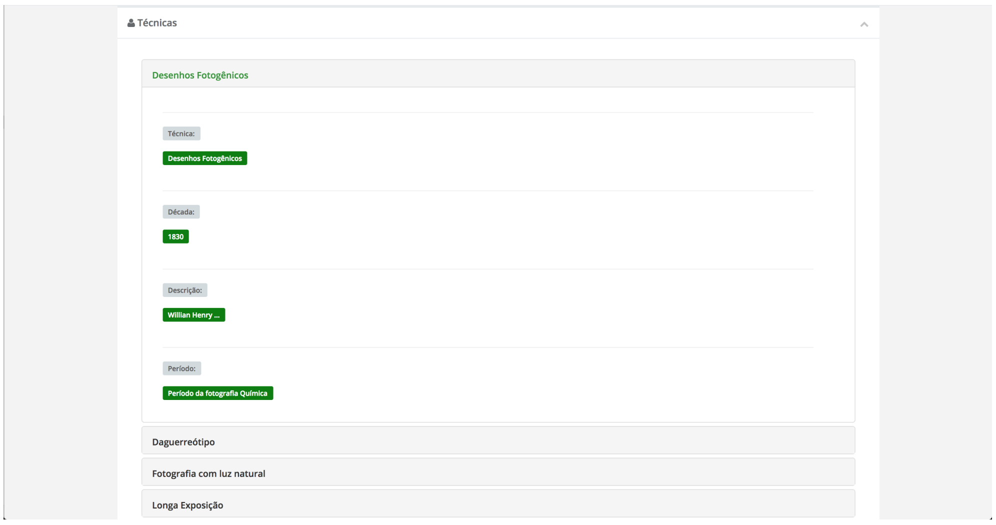

Técnicas da Fotografia

Técnicas da Fotografia
Dividimos a história da fotografia em três grandes períodos. Essa classificação se justifica não apenas pela mudança dos suportes ou das técnicas, mas também pelo fato de que é possível diferenciar drasticamente todo o sistema em torno da fotografia em cada um dos três momentos.

1672-2010
Período

Figure 11.3: Exhibitions list rendered

The permanent exhibition list contains only one exhibition rendered in accordance with the CaVa$^{\mathrm{DSL}}$ Specification running example. In Figure 11.3, the specified visible attributes are the "title", "short description", "icon" (camera-retro), "additional info" (1672-2010 Período) and "behavior" (expanded, denoted by the chevron-up icon in the top right corner). The rest of the attributes and query operator result are visible inside the exhibition room, as seen in Figure 11.4, which presents the final exhibition room rendered according to the code of Listing 10.5.


Técnicas

Desenhos Fotogênicos

Técnica:
Desenhos Fotogênicos

Década:
1830

Descrição:
Willian Henry ...

Período:
Período da fotografia Química

Daguerreótipo

Fotografia com luz natural

Longa Exposição

Figure 11.4: Permanent exhibition Room (*exhibition1.php*) rendered

The “exhibition1.php” file content contains a list of photography techniques based on Listing 10.5 and the query results of Listing 10.4.

The footer of MVIF running example, described according to CaVa$^{\text{DSL}}$ Specification is shown in Figure 11.5.

**Copyright** - Ricardo Martini © 2018 powered by CaVa

Figure 11.5: MVIF footer rendered

The footer contains images, alignments, format date, behavior (fixed) and style (condensed), specified in accordance with the CaVa$^{\text{DSL}}$ specification of the MVIF running example.

Next, in Part III, the case studies are described applied in the CaVa architecture, showing all the stages needed to achieve the final goal of this PhD research, the virtual Learning Spaces.

# Part III

# Case Studies

# Chapter 12

# Case Study 1 – Emigration Documents belonging to Fafe's Archive

The act of leaving a country (usually of origin) towards another, to establish residence, is called emigration. This case study is applied in the context of the emigration phenomena in Portugal, more specifically the emigration movement from Fafe (a city in the north of Portugal) towards several countries, dated from 1960 until 1970.

The main concept for this case study is the emigrant, a person who leaves his country to live in another. Most of them do it to improve their economic conditions, search for jobs, escape from an area of natural disaster or devastated by a war, etc.

This work had the cooperation of the municipal Archive of Fafe, which is liable for the preservation, organization, and dissemination of the physical emigration documents[1] content.

This collaboration has brought many benefits in the scope of the development of this case study. Working with professionals of the domain like archivists,

[1] The documental series "processos de emigração" contains more than 6400 records.

has brought the opportunity to learn and to improve the knowledge about the documents and what they include as well as the chance to exchange information and experience with the experts in terms of the emigration documents.

The municipal Archive of Fafe holds fonds related to the emigration domain like passport application forms, ship routes, biographies, almanacs, among others. At this moment, for this case study, the passport application forms are the only object of study. Although not created as works of art, those documents contain information that after processed can be considered as historic cultural material and in that sense can integrate an immaterial museum collection about Human Beings.

This chapter is organized as follows. Section 12.1 describes the structure of the passport application forms. Section 12.2 discusses the design of the relational database repository (bdME), as well as its final schema. Section 12.3 presents the web-based information system (SGPE) to assist in the recovery of the emigrant's documents. Section 12.4 presents the construction of a Reduced CRM-compatible form ontology for the emigration domain (OntoME) based on the international standard for museum ontologies, CIDOC-CRM. In Section 12.5 a mapping is outlined – with the aid of the Ontop framework – as a solution for the communication problem between bdME and OntoME. Section 12.6 lays out a CaVa$^{\text{DSL}}$ specification for specifying a virtual Learning Space for the emigration domain (a virtual museum) from the curator's perspective. To cope with the CaVa$^{\text{DSL}}$ specification to automatically generate the emigration virtual LS, a set of processors (CaVa$^{\text{gen}}$) is applied and explained in Section 12.7. Section 12.8 points out the rendering of the final virtual LS about the emigration phenomena automatically generated by the CaVa system. Finally, Section 12.9 summarizes the chapter and analyzes the results.

## 12.1 The structure of the emigration documents

The concept of emigrant has undergone changes over the years. According to the Portuguese Decree-law 34 330[2] of December 27th, 1944, an emigrant is defined as the citizen who leaves the national territory to work in another country, women who will accompany or join the emigrant husband, relatives by consanguinity in any degree of the direct line or up to the third degree of cross-line of any emigrant wishing to accompany or to join him. Years later, another Decree-law was established. The Portuguese Decree-law number 44 428[3] of June 29th, 1962, added a clause about the definition of the emigrant: those who transfer their residence to foreign country benefiting from the quality of emigrant or equivalent.

In order to define the principles for the protection of the emigrant, the Portuguese Decree-Law 36 558 of October, 28th, 1947 created *Junta da Emigração*, a department to treat of "all due diligence and preparatory formalities of boarding of any emigrant and of their application forms."[4] [Ribeiro, 1986], but later, this department was unable to deal with all the applications of the emigration in Portugal. Then its tasks were reduced to the appreciation of passport applications that started to be processed and archived locally at the Municipal Departments [da Silva Nascimento, 2009].

Therefore, the emigration documents characterized in this case study are restricted to those that describe the emigrants that apply for a passport in that period (1960-70). These files contain information such as general data of the document itself (document identification), general data about the emigrant, list of attached documents, family members accompanying the emigrant, desired type of transportation, qualifications (literary and professional) and criminal records, family members in charge of the emigrants dependents (who remain in the country), previous trips abroad, details of the person calling

[2] Available at: `https://dre.pt/application/file/568641`
[3] Available at: `https://dre.pt/application/file/164552`
[4] See also: `https://dre.pt/application/file/635905`

the emigrant, employment contract, aid in the destination country, among other information. All these data items (more than 80) must be provided to get a passport.

Each item to get a passport will be described in Appendix B in order to clarify its meaning and the data contained.

This large set of data items has a big potential to describe each individual integrated in the society of his epoch, but it also provides knowledge about the society in a precise context of the country/world history. This justifies the relevance of the dissemination of the knowledge enclosed in that particular collection of Fafe's Archive.

## 12.2 bdME, a Database to store the emigration documents Repository

This section has the goal of describing the design of a database dedicated to the analysis of the emigration documents. To design this kind of database it is required knowledge about the domain and also access to the organization that provides these documents. As mentioned, this was possible due to the cooperation with the Municipal Archive of Fafe. The design of the emigration database was already published in [Martini et al., 2015].

The logical model of the database covers the data items, described in Section 12.1, that are required to obtain an emigration passport. This model with sixteen tables is the result of the normalization of the high-level conceptual schema firstly drawn centered on tables *Processo* and *IdentificacaoEmigrante*. From that, the relational database (physical model) was developed in MySQL.

The design of this database was of great importance for this project, because storing the emigration documents in a database is a good way to preserve them and also to later create the Exhibition Rooms, disseminating the emigrations data. This information shall be displayed in the best possible way, making the Exhibition Rooms easier to handle than the physical documents.

The following subsections show how the relational database is organized (conceptual, logical, and physical models).

### 12.2.1 Conceptual Model

The data described in Section 12.1 was modified and rearranged (grouped or separated) and is described below in a conceptual model:

- Data about the emigrants documents:
  - Registration number at City Hall, registration number at the Board of Emigration and year;
  - Age and civil status of the emigrant at the document;
  - Destination (country and location);
  - Dispatching (date, craft and passport);
  - Boarding data (type of the transport, company, date, location, designation of the transport, whether or not the pass is paid, landing port, and with who the applicant travels);
  - Qualifications (literary qualifications, occupation and workplace);
  - Economic conditions (remuneration, working days, reason for the emigration and expenses of travel);
  - Criminal record (whether or not he/she is judged and if the applicant or his relatives have any case pending);
  - Aid in the destination country (name of the people who will give the aids, kinship, residence, occupation and aid provided);
  - Kinship of the emigrant with the caller;
  - Employment contract (how the emigrant got the employment contract, occupation and salary);

  - Women and minors employed (whether or not he/she already worked for the contractor or for the family of him/her, duration of the work and occupation);
  - Declaration of the wife (knowledge of the wife about the target location of her husband and if the wife considers that her maintenance is assured by the husband in the country that she stays).

- Residence (Location) of the emigrant: county, district, parish, and street;
- Data about the emigrant: emigrant's identification, name, place and date of birth, filiation and spouse;
- Filiation of the emigrant: name of the parents of the emigrant;
- Data about the caller: name, residence, time that he/she is in the abroad and passport;
- Data about the contractor: name, residence and if the caller knows the emigrant;
- Data about the intermediary: name, residence, occupation, parents' name, kinship with the emigrant, passport and how much was paid to the intermediary;
- Description of the attached documents;
- Data about the accompanying: name, age, date of birth, kinship with the emigrant and qualifications;
- Data about the people (family) who stay in the country: name, age of the dependent, kinship with the emigrant, if the person is in charge of the emigrant and residence;
- Description of other information (notes) about the emigration documents; and

- Previous travel abroad: whether or not it is the first time that the applicant is traveling, date on which he/she returned, passport number and date, and entity that issued the passport.

The database was built taking into account the data listed above (conceptual model); the following Subsections 12.2.2 and 12.2.3 describe the technical part of the database.

### 12.2.2 Logical Model

The logical model shows the relations between the database tables as well as primary and foreign keys, i.e., the technical part of the database, but not yet the physical model, which defines how the database was built.

Figure C.1 of Appendix C illustrates the logical model with the primary and foreign keys, relations between the tables, and the cardinality of these relations.

As already mentioned, this step contains the primary and foreign keys of the database tables. Each class in Figure C.1 has a primary key and some classes have foreign keys assigned to it. To deal with relations with cardinality `0..*` to `0..*` (many to many) were created association classes that specify the common data between tables in the relation. The tables that hold these characteristics are `ProcessoAcomp, ProcessoAnexo` and `ProcessoPessoasFam`.

After describing the logical model of the database, the physical model will be discussed in Subsection 12.2.3.

### 12.2.3 Physical Model

The physical model describes the implementation of the final database in detail. Taking into account that model, it is possible to create the database and their tables, executing statements written in SQL (Structured Query Language). The database engine chosen for this project was MySQL.

Figure C.2 of Appendix C shows the physical model of the database with all the characteristics mentioned above.

As said above, the diagram shown in Figure C.2 can be translated into DDL (Data Definition Language) statements of SQL to create the schema. Listing 12.1 is an example of a DDL statement, in this case to create the table `anexo`:

Listing 12.1: Snippet DDL to create table `anexo`

```sql
CREATE TABLE 'anexo' (
    'idAnexo' int(11) NOT NULL AUTO_INCREMENT,
    'descricao' varchar(128) DEFAULT NULL,
    PRIMARY_KEY ('idAnexo')
);
```

The physical construction of the database will be completed after the execution of the DDL script written according to the description of all the tables in Figure C.2. With this stage done, the population of the database is explained next.

## 12.3 SGPE, a DIS to populate the bdME Database

This section is concerned with the development of an information system built in the context of a municipal archive aiming at supporting the preservation of physical documents and at facilitating the information extraction and dissemination. Saving individual records, like emigration documentation is essential for end-users and History or Social Sciences Researchers as they can know more about each record and they can learn about the society.

The emergence of the Internet provides access to the desired information to anyone, anywhere, at anytime. As a consequence, many information sources like libraries, museums, and other similar institutions began storing documents digitally. Thus, the data stored in digital format may be brought forward on Web pages, displaying it to the end-users and enabling them to learn and interact with the available information. So, this section aims at presenting the SGPE (Sistema Gerenciador de Processos de

Emigração)[5] web-based information system to assist in the population of the bdME database with data recovered from the emigrant's documents. The design and development of the SGPE system were already published in [Martini et al., 2015]. Also the SGPE application is available at `http://www4.di.uminho.pt/~gepl/museu_emigracao`.

SGPE is an essential and initial part of CaVa, because it is a component that is responsible for the digital storage of the emigration documents that later are used to generate the virtual Learning Spaces (remember that it is a component of Module A, Chapter 6).

Analyzing, understanding and transcribing the correct data of the emigration documents are difficult tasks when those records are written by hand or typewritten.

Typesetting all data items mentioned in Section 12.1 and Appendix B, to transform the original paper documents into electronic documents (for safety and further processing), must be a user-friendly and secure task.

Typesetting errors should be avoided by reducing the text required to enter, and by validating data fields as soon as they are fulfilled. Comboboxes (drop down lists) offering lists of predefined values to choose, and similar user interaction techniques should be carefully identified and provided.

Thus, to transcribe and to populate the emigration database (bdME) in an easier way, SGPE was built, designed to deal adequately with the typesetting problems mentioned previously. SGPE was projected to cope with the emigration documents dated from the 1960s until the 1970s. However, the application can be used for any time period. This is true because the interface does not impose any constraints concerning the documents date. Moreover SGPE is not oriented to a special kind of form. This means that SGPE can cope with data collected from similar documents with different formats produced by Junta da Emigração along the years.

The web-based application was developed in the PHP programming lan-

[5] Emigration Documents Management System.

guage, jQuery and JavaScript and with the aid of a high-performance PHP framework (Yii[6]). The framework comes with rich features like MVC (Model-View-Controller), DAO (Data Access Object) / ActiveRecord, I18N/L10N, among others, which can significantly reduce development time. Furthermore, Yii also helps on the development of the four basic operations of a database, automatically generating code for the CRUD (Create, Read, Update and Delete) operations.

To create an Yii application, it is necessary to follow three steps: (i) Create the database (already explained in Section 12.2); (ii) Generate the PHP code; (iii) Customize the code to fit the needs of the application. For the second step, Yii has a module called Gii that provides web-based code generation capabilities. The purpose of Gii is to create the fundamental model, view, and controller files required by the web-based application [Ullman, 2013].

As referred earlier, typesetting errors can occur and, to prevent them, SGPE has applied the interaction techniques previously mentioned. To illustrate this, Figure 12.1 presents three examples of preselected value components that help the user to enter the correct data.

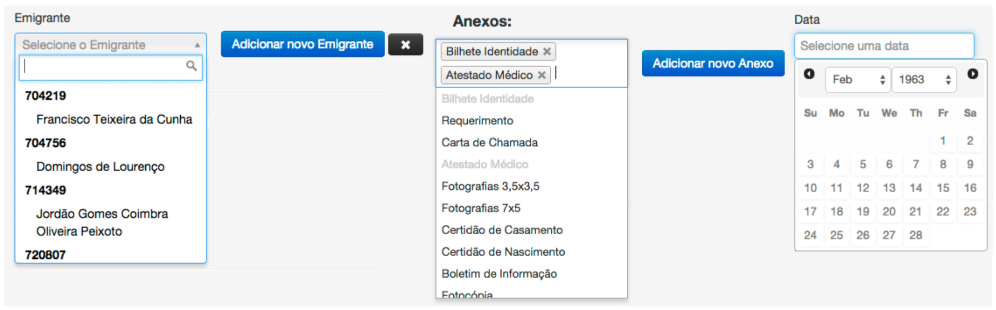


Figure 12.1: Components with preselected values

The first example is a component that shows a list of emigrants with their identification numbers to select one of them. The second is a widget that presents a predefined list of attached documents that can be appended to the emigration records and the third is a component to select a date. These

6 Available at: http://www.yiiframework.com/

interface components exemplify three situations that are error prone. All widgets have the same purpose that is to let the user choose one option to avoid duplicate data or mistyping. If the user needs to enter new data, then the buttons on the right side of the component enable this feature (except for the date widget).

In the Gii module two options were executed: (1) the generation of models; and (2) the generation of CRUD operations. For the generation of models, the only thing that is needed is to enter the table names of the database and the result are PHP files with the name of the tables. Each PHP file created has a PHP class associated with the same name. The models in Yii store data and define the business rules for that data. To handle these features, each class has some methods, the most important are called `rules()` and `relations()`.

The `rules()` method returns an array of rules by which the model data must abide. This method represents a key benefit of using a framework: built-in data validation. The `relations()` method is used to indicate a relationship from one model to others, i.e., the relations between models with the foreign keys [Ullman, 2013].

For instance, a model will receive the name identificacaoEmigrante if it represents the table identificacaoEmigrante of the database. From this model, the Gii tool will create a file named `identificacaoEmigrante.php` that will contain the methods mentioned previously, as shown in Listings 12.2 and 12.3.

Listing 12.2: **rules():**

```php
<?php
    public function rules() {
        return array(
            array('idEmigrante, nome', 'required',
                  'message' => 'O campo <b>{attribute}</b> não pode ser vazio.'),
            array('idEmigrante, idFiliacao, idNaturalidade', 'numerical', 'integerOnly'),
            array('nome, nomeConj', 'length', 'max' => 64),
            array('idConj', 'length', 'max' => 10),
            array('dtNasc', 'safe'),

            array('idEmigrante, nome, dtNasc, idConj, nomeConj, idFiliacao, idNaturalidade', 'safe',
                  'on' => 'search'),
        );
    }
```

The `rules()` method presented in Listing 12.2 has the array to be returned with the attributes of the identificacaoEmigrante table with the rules for each attribute. For `idEmigrante` and `nome` attributes, there is a rule that identifies these attributes as required by the model, i.e., in the final application, these attributes are mandatory to fill. If these attributes were not filled, the message: "`O campo <idEmigrante OR nome> não pode ser vazio`" will be shown. The other rules in this function are similar and their objective is to set the type and the maximum length (`max`) of all the table fields.

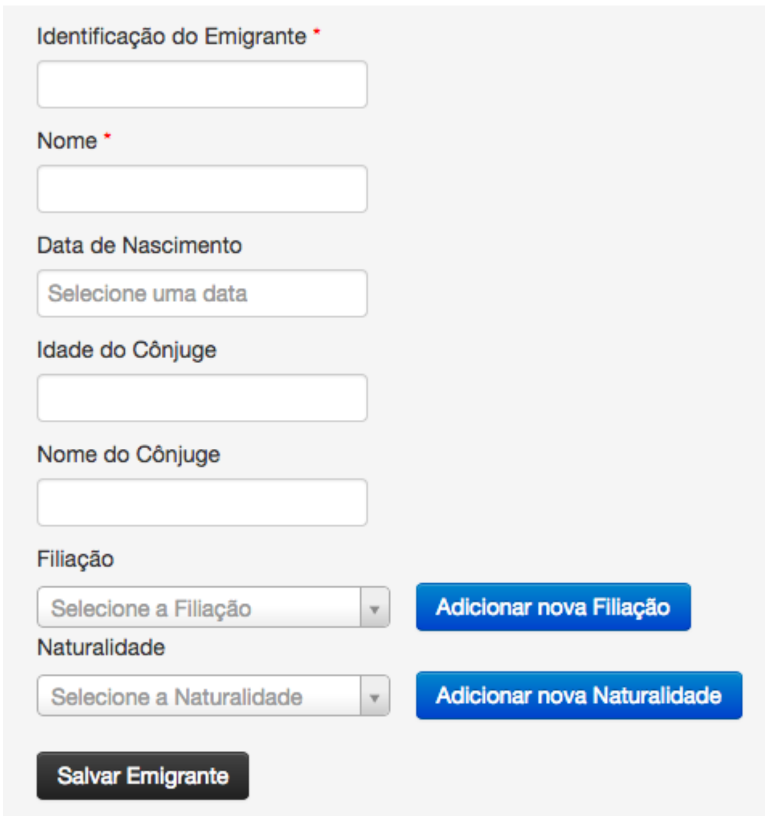
Listing 12.3: **relations():**

```
1  <?php
2      public function relations() {
3          return array(
4              'idFiliacao' => array(self::BELONGS_TO, 'Filiacao', 'idFiliacao'),
5              'idNaturalidade' => array(self::BELONGS_TO, 'Localidade', 'idNaturalidade'),
6              'processos' => array(self::HAS_MANY, 'Processo', 'idEmigrante'),
7          );
8      }
```

The `relations()` method presented in Listing 12.3 has the array to be returned with the attributes of the identificacaoEmigrante table with the relations between the identificacaoEmigrante model and other models, in this case, the Filiacao (idFiliacao), Localidade (idNaturalidade) and Processo (processos) models. Each element of the "returned array" (lines 4 to 6), describes the kind of the relation. Lines 4 and 5 have a kind of relation named `BELONGS_TO` that means that attribute "idFiliacao" and "idNaturalidade" belongs to the model Filiacao and Localidade, respectively. Line 6 has another kind of relationship. The relation established is `HAS_MANY` that represents the cardinality 1 to *, i.e., one emigrant (identificacaoEmigrante) can have many documents, but one document just refers one emigrant.

In addition to creating the model, the views and controllers are also created in the second step. The views are PHP files containing the code to display the forms and graphical components to the user, i.e., the code responsible for the user interaction with the application. In a web-based application, like SGPE, the views are a combination of HTML and PHP code that create the screen that the user will see in the browser [Ullman, 2013].

A view of SGPE can be seen in Figure 12.2. This is a form for identificacaoEmigrante model. Note the red stars to advise the user that the field is mandatory (required), as seen previously in the rules of the model (Listing 12.2).



Identificação do Emigrante *

Nome *

Data de Nascimento

Selecione uma data

Idade do Cônjuge

Nome do Cônjuge

Filiação

Selecione a Filiação

Naturalidade

Selecione a Naturalidade

Figure 12.2: Screen to register Emigrants (corresponding to the "identificacaoEmigrante" model)

As mentioned before, there are also the controllers, that are intermediary agents that handle user and other actions [Ullman, 2013].

A controller is a PHP file, just like models and views. The controller has some methods that can be understood as actions. These actions can be, for example, create, update, delete and show a record.

The corresponding method names always start with the word "action" as seen below [Ullman, 2013]:

- `actionCreate()`: for creating new model records;
- `actionIndex()`: for listing every model record;
- `actionView()`: for listing a single model record;
- `actionUpdate()`: for updating a single model record;
- `actionDelete()`: for deleting a single model record; and
- `actionAdmin()`: for showing every model record in a format designed for administrators.

To better understand the controller part of the MVC, an example of a controller (identificacaoEmigranteController.php) will be detailed showing the actions performed by the application. For the sake of space and simplicity, only `actionCreate()` will be shown.

Listing 12.4: **actionCreate():**

```php
?php
    public function actionCreateEmigranteProcesso() {
        $model = new identificacaoEmigrante;
        $this->performAjaxValidation($model);
        if((isset($_POST['idEmigrante']) && isset($_POST['nome']))) {
            $model->idEmigrante = $_POST['idEmigrante'];
            $model->nome = $_POST['nome'];
            $model->dtNasc = $_POST['dtNasc'];
            $model->idConj = $_POST['idConj'];
            $model->nomeConj = $_POST['nomeConj'];
            $model->idFiliacao = $_POST['idCarregaFiliacaoEmigrante'];
            $model->idNaturalidade = $_POST['idCarregaLocalidadeEmigrante'];

            if($model->save()) {
                $id = $model->getPrimaryKey();
                $_SESSION['codEmigranteAjax'] = $id;
            }
        }
        if(!isset($_POST['inserindoViaAjax'])) {
            $this->render('createEmigranteProcesso', array('model'=>$model));
        }
    }
```

As seen in the code of Listing 12.4, first a new model is instantiated (in this case, the identificacaoEmigrante model). This is necessary to access the fields to create a new emigrant in the database (model). After that, if the $_POST values are not empty, each field of the model receives this value, that is the value of the components (HTML forms) and at last, the model is saved (line 14). Note the word “action” at the beginning of the method name.

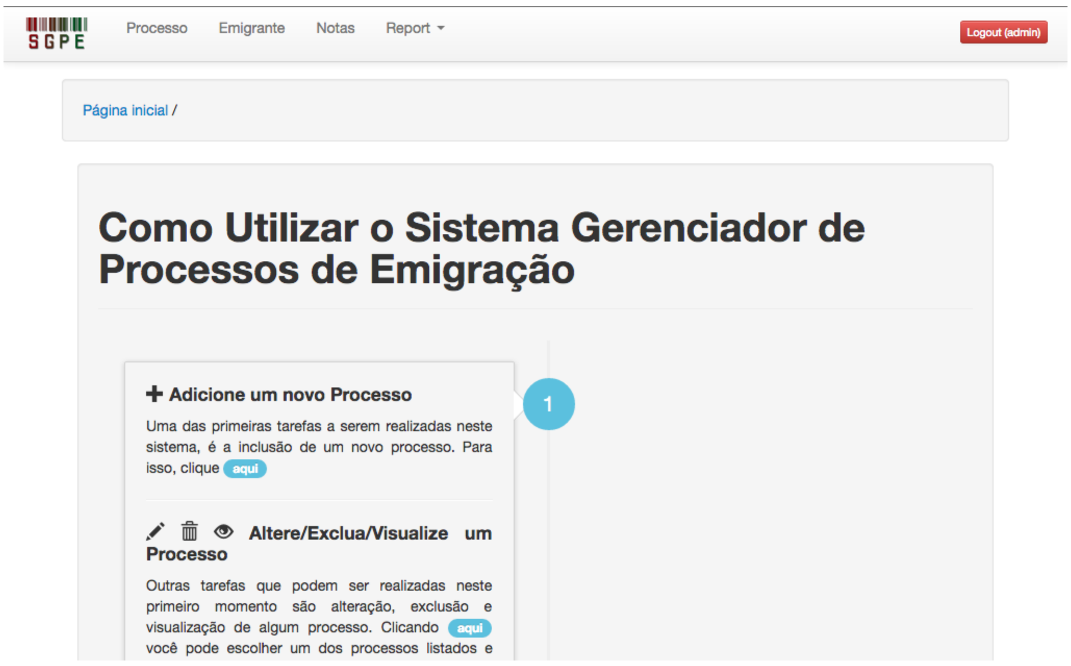
After creating the fundamental models, views and controllers, in the third step, it is necessary to customize the application with the particular needs.

As seen in Figure 12.2, the components (buttons, textfields, dropdown menus, etc.) are not the built-in components of Yii. Yii framework has several extensions for various purposes. To customize the application components in the views, the Yiibooster[7] extension was used. This extension uses components similar to the well known framework bootstrap[8]. Figures 12.3 to 12.8 show some screenshots of the SGPE application.

When the user opens the SGPE, the first page that he accesses is a tutorial of “How to use the Management System of the Emigration Documents” with some steps that show the flow of the system, describing the functionality (creation, update and view) of all models of the application. These models are “identificacaoEmigrante”, “Notas”, “Filiacao”, “Acompanhante”, “Anexo”, “Chamante”, “Contratante”, “Intermediario”, “Localidade”, “Lugar”, “PessoasFamFicamPais” and “DeslAnteriores”.

Figure 12.3 illustrates the main page. The navigation bar on the top of the page is the same for all views in the application. This navigation bar is a menu that gives access to the models “Processo”, “identificacaoEmigrante” and “Notas”, because the rest of the models are accessible from “Processo” model. There is still an item called “Report” that will be presented later.

[7] To learn more visit: http://yiibooster.clevertech.biz/
[8] To learn more visit: http://getbootstrap.com/



Como Utilizar o Sistema Gerenciador de Processos de Emigração

Adicione um novo Processo

Uma das primeiras tarefas a serem realizadas neste sistema, é a inclusão de um novo processo. Para isso, clique aqui

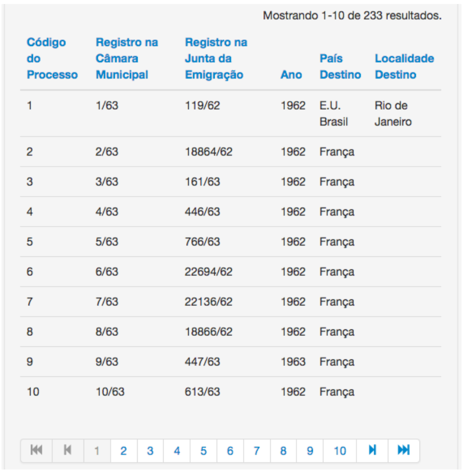
Altere/Exclua/Visualize um Processo

Outras tarefas que podem ser realizadas neste primeiro momento são alteração, exclusão e visualização de algum processo. Clicando aqui você pode escolher um dos processos listados e

Figure 12.3: Timeline of the main page of the SGPE

From the main page, the user can access the views (pages) to create, update, visualize or remove any item of the navigation bar. Along the steps of the main page, it is also possible to activate any action, except the creation of the models that are not in the navigation bar (Processo, Emigrante and Notas).

Following the steps of the initial page, the first thing to do is to access the documents of the emigrants ("Processo") page to create an emigrant record. Accessing the "Processo" page, the first thing that appears is a list of emigrant's documents (see Figure 12.4) that shows a summary of the documents (10 per page).



Mostrando 1-10 de 233 resultados.

| Código do Processo | Registro na Câmara Municipal | Registro na Junta da Emigração | Ano | País Destino | Localidade Destino |
|---|---|---|---|---|---|
| 1 | 1/63 | 119/62 | 1962 | E.U. Brasil | Rio de Janeiro |
| 2 | 2/63 | 18864/62 | 1962 | França | |
| 3 | 3/63 | 161/63 | 1962 | França | |
| 4 | 4/63 | 446/63 | 1962 | França | |
| 5 | 5/63 | 766/63 | 1962 | França | |
| 6 | 6/63 | 22694/62 | 1962 | França | |
| 7 | 7/63 | 22136/62 | 1962 | França | |
| 8 | 8/63 | 18866/62 | 1962 | França | |
| 9 | 9/63 | 447/63 | 1963 | França | |
| 10 | 10/63 | 613/63 | 1962 | França | |

Figure 12.4: List of Emigrant's documents

There is also a generic menu that shows the operation available to handle the emigrant's documents. However, this lateral menu is similar for all other models, always with the same options adapted to the model involved.

Depending on which page the user is viewing, the generic menu shows the appropriated options. For example, if the user is on the page to create an emigrant document, the options are 'list' or 'manage' documents, but if he is on the page to view an emigrant document, the list of options are 'list', 'add', 'update', and 'delete' the selected record. As "Processo" is the main model of SGPE, there are two exclusive navigation items to facilitate the view of a record on the generic menu that correspond to the previous and the next documents, respectively.

In addition to the CRUD operations, the last item in the navigation bar (on the top of SGPE main screen) gives access to the reports concerning the present stats of the database. One feature that SGPE provides is a summary of the number of the records in the database. Figure 12.5 shows the graphic with the number of records in each table of the database.

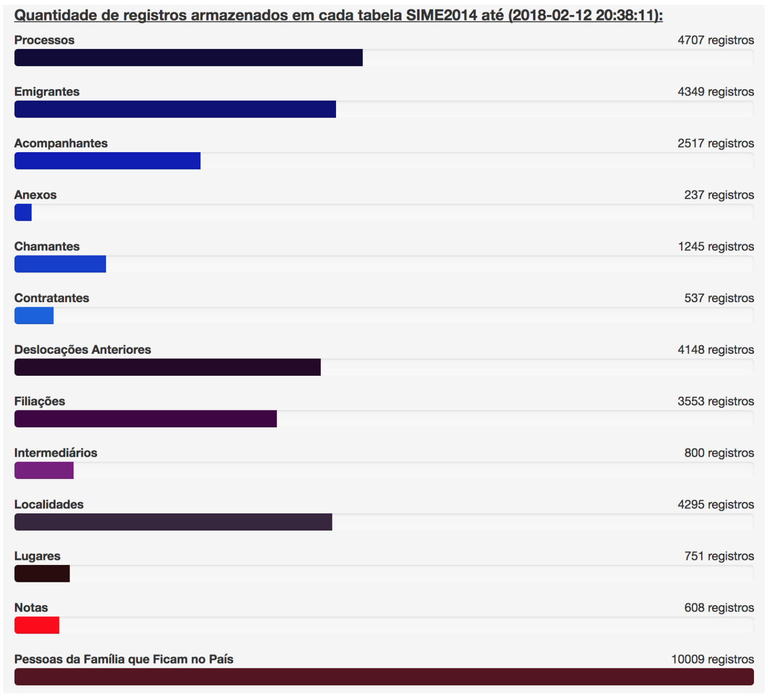


Figure 12.5: Graphic showing the count of the records in the database

The same page also exhibits another way (percentage's stack) to view the number of records in the database. It can be seen in Figure 12.6.

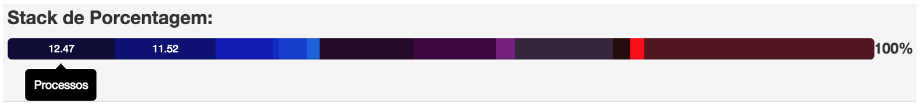


Figure 12.6: Stack percentage of records

Subsection 12.3.1 presents some obstacles found in the development of SGPE.

### 12.3.1 Problems found

The Yii framework helps to reduce the development time, however the use of such tools has some drawbacks. Mainly when these tools are opened for developers to extend the framework. In the development of SGPE, most of the issues found were related to extensions namely concerned with the use of jQuery.

In online discussion forums, many problems are reported with Yii framework and jQuery. As SGPE has large forms to be filled, a way to fill them easily is using Ajax/jQuery, because the content of the pages (forms) can be rendered without the need to reload the entire page — this is a very important feature of SGPE.

The Yii framework has some known bugs and problems with some extensions. The Yiibooster extension working together with the Ajax/jQuery brings some problems when rendering the components. Figure 12.7 shows in (a), the correct form of the presentation of the components and in (b), the incorrect form, without the rendering of the extension component.

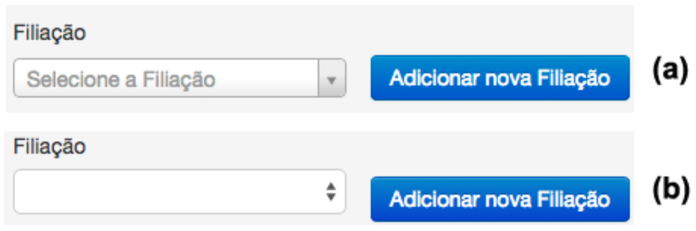


Figure 12.7: Rendering the correct component (a) and rendering the incorrect component (b)

Looking at Figure 12.7, there are not many differences between the component rendered in (a) and that rendered in (b), but in Figure 12.8 it is possible to see that the component rendered in (a) has some features to help the user to enter the correct information and in (b), this does not occur. The extensions, frameworks and applications should provide these mechanisms to actually aid the end-user or else, it would not be necessary to have an application.

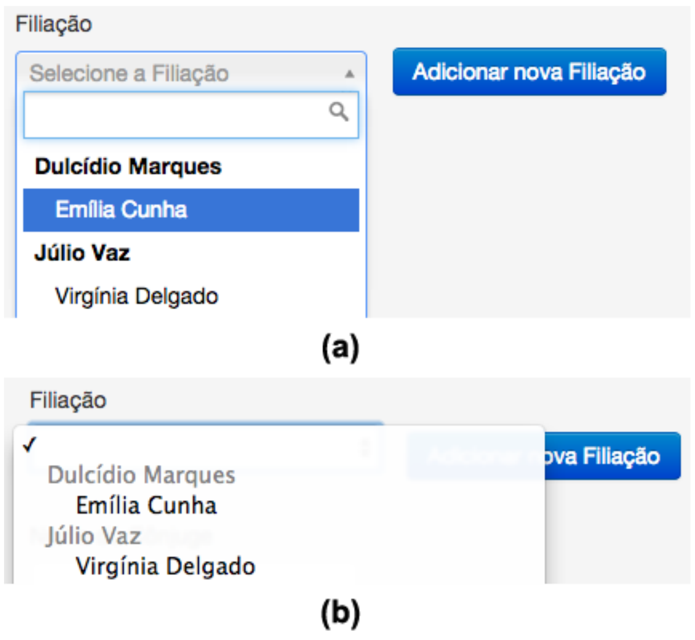


Figure 12.8: Correct component (a) and incorrect component (b)

The problem illustrated in Figures 12.7 and 12.8 happened because the Yii framework was loading jQuery twice or more. This problem was circumvented by not allowing the Yii framework to load it more than once. In Listing 12.5, there is a code snippet showing the solution for this problem.

Listing 12.5: **actionCarregaFiliacoes():**

```php
<?php
    public function actionCarregaFiliacoes()
        $model = new filiacao;

        Yii::app()->clientscript->scriptMap['jquery.js'] = false;
        $this->renderPartial("carregaFiliacoes", array('model'=>$model), false, true);
        Yii::app()->clientscript->scriptMap['jquery.js'] = true;
    }
```

The code of Listing 12.5 presents a function to:

1. disable jQuery:
   `Yii::app()->clientscript->scriptMap['jquery.js'] = false;`
2. render a view, not rendering the entire layout:
   `$this->renderPartial("carregaFiliacoes",..., false, true);`.

The two last parameters of the renderPartial function serve to control whether or not the result is returned or displayed to end users. So, this could be used to produce some solutions on Ajax/jQuery requests;

3. enable jQuery:
   `Yii::app()->clientscript->scriptMap['jquery.js'] = true;`

Thus, disabling, rendering a view, and enabling jQuery again, the components that are called by Ajax appear correctly. The only thing to do is to create a separate PHP file with the widget to be rendered and in the Ajax/jQuery code, call in Ajax `success` function, the jQuery `load` method with the URL to the PHP file of the widget to be rendered.

### 12.3.2 Summary

The population of bdME through SGPE was done with the cooperation of the Municipal Archive of Fafe due a project accepted by the Iberarchivos-ADAI Program. In the application year (2015), 36 projects, among the 17 Iberarchivos-ADAI program countries, were approved. From Portugal, only four projects were accepted, being this one of the approved works. In addition to being selected, the project entitled "Reflexos da emigração no concelho de Fafe nas décadas de 60 a 80 do séc. XX: um projeto de conservação, informatização e divulgação" was listed as one of the 4 most relevant[9]. The project had as result a website[10] to propagate the emigration phenomena studied.

More than 4000 documents were added to the database (see Figure 12.5). At this moment, 74% of all documents were inserted into the database. To achieve the 6400 (100%) documents, the work of populating is already being done.

[9] Accessible at:
`http://segib.org/wp-content/uploads/Informe_2015_Iberarchivos.pdf`

[10] Accessible through the iberarchivos.org: `http://bit.ly/2p3tC2S`

Next, the OntoME ontology is explained, aiming at describing the emigration domain.

## 12.4 OntoME, an ontology for the emigration domain

This section discusses the use of a Reduced CRM-compatible form ontology for the virtual Emigration Museum based on the international standard for museum ontologies, CIDOC-CRM. This work was already published in [Martini et al., 2016b].

To extract knowledge from the information of the virtual Emigration Museum when navigating through it, abstract data models should be used to conceptualize the emigration documents stored in the relational database (bdME). In that way, resorting to an ontology (as an abstract layer), the information contained in those documents can be accessed by the end-users (the museum visitors) to learn about the emigration phenomena.

After a CIDOC-CRM in-depth analysis (Section 3.4.1), it was possible to correlate the compatible entities of the ontology with the emigration documental fond. In that way, to demonstrate how the emigration documents that belong to the Municipal Archive of Fafe fit in CIDOC-CRM, Figure 12.9 shows an example based on the CIDOC-CRM ontology core (Figure 3.1) instantiated with the information collected from the bdME database about the emigration movement of a person (José Carlos Magalhães). As in this work the CIDOC-CRM standard ontology was used, examples are the best way to demonstrate its use with the bdME instances. Notice that the use of data from the bdME database is done just for exemplifying the use of CIDOC-CRM. The relation between the instances stored in the database and the ontology concepts is done through a mapping, which is presented in Section 12.5.

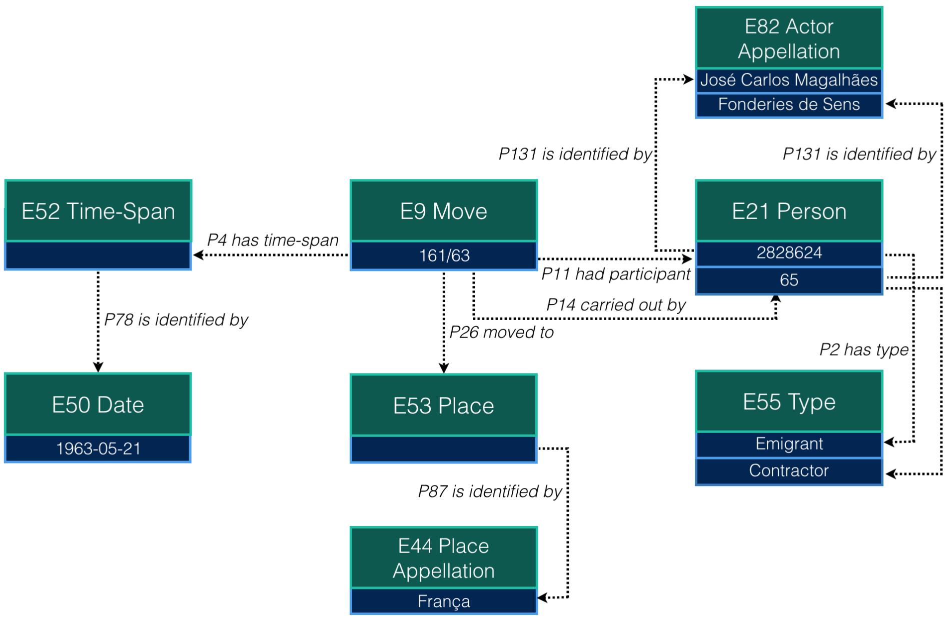


Figure 12.9: Reduced CRM-Compatible Form instantiation example

As shown in Figure 12.9, the main entity is an event of type movement (*E9 Move*), which refers to the emigration document that reflects a passport application form identified by the number '161/63'. *E9 Move* has four relations specifying:

**when the movement has occurred:** described by *E52 Time-Span*, which in this case *(P78) is identified by* '1963-05-21', an *E50 Date*;

**where the emigrant moved to:** described by *E53 Place*, which in this case *(P87) is identified by* 'França', an *E44 Place Appellation*;

**who emigrated:** described by *E21 Person* named '2828624', which in this case *(P131) is identified by* 'José Carlos Magalhães', an *E82 Actor Appellation*. *E21 Person* has a type to identify its role in *E9 Move*. So person '2828624' *(P2) has type* 'Emigrant';

**who carried out** [11]**:** described by *E21 Person* named '65', which in this case *(P131) is identified by* 'Fonderies de Sens', an *E82 Actor Appel-*

[11] Notice that exist other objects related to this same property, with the difference in

*lation*. *E21 Person* has a type to identify its role in *E9 Move*. So person '65' *(P2) has type* 'Contractor'. Notice that it is not possible to determine, from the sources, whether the contractor is a person or a company (*E74 Group*). So, it is always described as an *E21 Person*.

Aiming at bridging the gap between bdME and OntoME, Section 12.5 presents the OBDA mapping developed.

## 12.5 Bridging the gap between bdME and OntoME

The Semantic Web aims at building a Web where data is enriched with meaningful annotations. In other words, data is semantically organized in such a way that both human and machine can understand and query it, aiming at the creation of dynamic Web pages.

Ontologies, as a keystone of the Semantic Web, have gained an ample acceptance as an information model, which can be used for several purposes, such as information retrieval in the Web. However, data is normally stored in databases, due to their acceptance based on their adaptability, effectiveness, and performance for representing and managing data, but they also present various problems in the Semantic Web context, because data is not semantically annotated [Martinez-Cruz et al., 2012]. To allow a sustainable growth of an ontology, efficient persistent storage of ontology concepts and data is essential [Gali et al., 2004].

Aiming at retrieving rich results in the sense of meaning, several ways of relating databases with ontologies have emerged. This section presents a mapping – with the aid of a framework (Ontop) – as a solution for the communication problem between the bdME relational database, presented

the *E55 Type*. In this project there are types like: who intermediates the emigration movement (the intermedary); who is calling the emigrant (the caller); etc.

in Section 12.2, and the ontology of the Emigration Museum (OntoME), presented in Section 12.4, which describes the Cultural Heritage domain.

To better explain the mapping work done, Subsection 12.5.1 presents an example of instances from bdME and an example of the OntoME ontology schema to describe the bdME example data (Table 12.1). These examples serve as a basis to illustrate the OBDA mapping that will be presented in Subsection 12.5.2.

## 12.5.1 bdME data and OntoME schema

An example of the data contained in bdME is shown in Table 12.1. This dataset excerpt will be used to demonstrate the mapping in Subsection 12.5.2.

Table 12.1: “identificacaoEmigrante”, “filiacao”, and “localidade” tables

*identificacaoEmigrante*

| *idEmigrante* | **nome** | **dtNasc** | *idFiliacao* | *idNaturalidade* |
|---|---|---|---|---|
| 713204 | Aníbal de Castro | 1926-10-30 | 265 | 30712 |
| 720807 | Lucinda Ribeiro | 1935-07-29 | 37 | 30728 |
| 2665155 | Manuel Vaz | 1924-03-26 | 3 | 30717 |
| 2828624 | José Carlos Magalhães | 1944-09-27 | 161 | 30712 |

*filiacao*

| *idFiliacao* | **nomePai** | **nomeMae** |
|---|---|---|
| 3 | Júlio Vaz | Virgínia Delgado |
| 37 | Júlio Ribeiro | Maria Nogueira |
| 161 | Serafim de Magalhães | Felismina de Freitas |
| 265 | – | Rosa de Castro |

*localidade*

| *idLocalidade* | **freguesia** | **concelho** | **distrito** |
|---|---|---|---|
| 30712 | Fornelos | Fafe | Braga |
| 30717 | Monte | Fafe | Braga |
| 30728 | São Gens | Fafe | Braga |

Table 12.1 shows three tables of bdME that describe the data about the emigrant’s identification (*identificacaoEmigrante*). The *identificacaoEmigrante* table comprises the identification number (idEmigrante), name (nome), birthdate (dtNasc), and two foreign keys that are related to the filiation (*filiacao* table) referenced by *idFiliacao* and birthplace (*localidade* table) referenced by *idNaturalidade*.

So, for instance, the emigrant identified by *2665155* has name *Manuel Vaz*, his birthdate is *1924-03-26* and has parents (filiation) identified by *3* – which corresponds to father *Júlio Vaz* and mother *Virgínia Delgado* from "filiacao" table – and his birthplace is referenced by *30717* – which corresponds to *Monte* parish, *Fafe* council, and *Braga* district, that together form a geographical location.

Looking at Table 12.1, the data can be understood and knowledge can be infered from that data, but usually, the user has no access to the schema and data of the databases. According to [Martinez-Cruz et al., 2012], information contained in databases cannot be semantically annotated. The authors describe this as "on one hand, the content of these databases is only shown when a query is performed in the database, and on the other hand, the semantic description of the database is represented using its schema, often unavailable or even useless because it can not be explored depending of the format chosen to represent it". So, this means that without both the data and schema, it is difficult to extract some information from relational databases.

Taking the data of Table 12.1 as an example, it can be noted that *idFiliacao* in *identificacaoEmigrante* table has values like *3*, *37*, *161*, and *265*. Those values are references to *filiacao* table. The value *3*, for instance, is a reference to a tuple that has two other values: *Júlio Vaz* and *Virgínia Delgado*. Those data items are sequences of characters and can mean anything. Can those two values be the father and mother of the emigrant identified by *2665155* or his children?

Looking now to the *localidade* table, an identifier can be seen (*idLocalidade*) and three other fields that correspond to parish (*freguesia*), council (*concelho*), and district (*distrito*). Those data items, together, can be understood as a locality (geographical place) and, indeed, they are, but do those places correspond to the birthplace or to the emigrant's home address? What do they really mean?

As said before, it is difficult to know the precise meaning of each data item without the association of the related concept chosen in a known vocabulary. It is only data, not information i.e., without a context, the data meaning can be messy.

However, having access to the conceptual layer, the user can query the database through the ontology concepts (known vocabulary) and the system should reason and translate the query into appropriate database questions [Poggi et al., 2008]. An example of good questions to answer the previously mentioned issues are: who are the parents of the emigrant identified by *2665155*? What is the birthdate of the emigrant identified by *2665155*? These kind of questions involve concepts (parents, emigrant, and birth) that are well known, because people share a common vocabulary related to a specific context, i.e., the ontology vocabulary.

To represent the bdME data of Table 12.1 in OntoME, Figure 12.10 presents the ontology schema extended to comprise the concepts of the emigration domain.

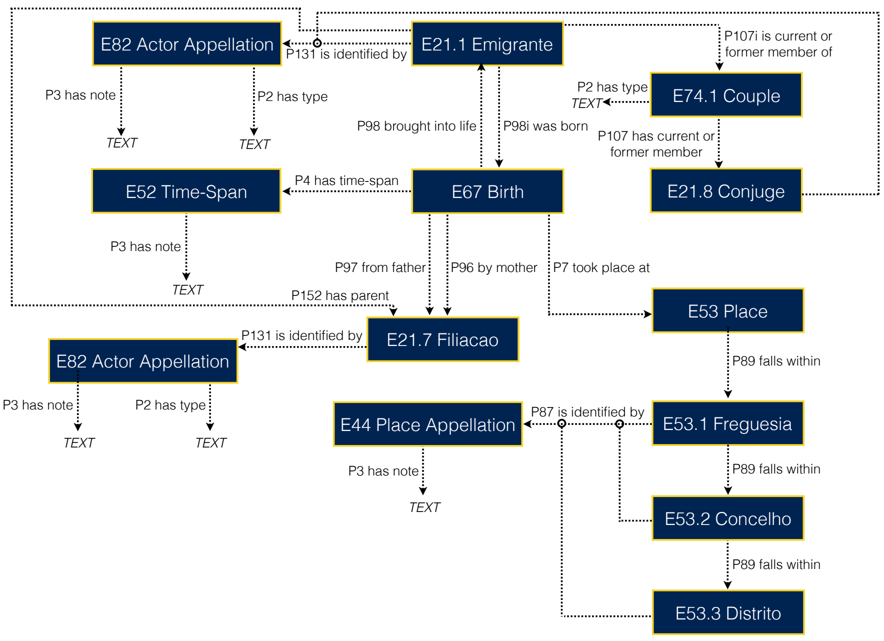


Figure 12.10: Example describing the OntoME concepts for the emigrant's birth event

Figure 12.10 illustrates the birth of an emigrant, linking the birth event (*E67 Birth*) to the other related entities:

- *E21.1 Emigrante*: who was born;
- *E21.1 Filiacao*: mother and father;
- *E53 Place*: the birthplace;
- *E52 Time-Span*: the birthdate (can include the time).

The ontology schema presented in Figure 12.10 has seven extended classes, namely (1) *E21.1 Emigrante*, (2) *E21.7 Filiacao*, and (3) *E21.8 Conjuge*, which are subclasses of *E21 Person*; (4) *E74.1 Couple*, that is subclass of *E74 Group*; (5) *E53.1 Freguesia*, (6) *E53.2 Concelho*, and (7) *E53.3 Distrito*, which are subclasses of *E53 Place*.

According to the Definition of the CIDOC Conceptual Reference Model (available at: `http://www.cidoc-crm.org/html/5.0.4/cidoc-crm.html#_Toc310250800`), "the CRM is intended to focus on the high-level entities and relationships needed to describe data structures. Consequently, it does not specialize entities any further than is required for this immediate purpose. However, entities in the isA hierarchy of the CRM may by specialized into any number of sub entities". This explains the use of the property *P2 has type*, serving as a label for the specialization.

Using the schema of Figure 12.10 and the data of Table 12.1, Subsection 12.5.2 points out the mapping between bdME and OntoME.

### 12.5.2 bdME2OntoME Mapping

The main purpose of creating a pavement between the repository and the ontology is to query the data layer (sources) through a conceptual layer. Ontology-Based Data Access (OBDA) is a paradigm that provides semantic access to databases by means of ontologies [Kogalovsky, 2012].

OBDA has various attractive features, many of them have been already proved effective in managing complex information systems [Lenzerini, 2011].

In OBDA, an abstract layer exists as an ontology, which defines a shared vocabulary of a specific domain. OBDA hides the database repository structure, and with this, it can enrich incomplete data with background knowledge [Kontchakov et al., 2014].

OBDA is based on a three-level architecture established by an ontology (the main component and a formal description of the domain of interest), data sources, and the mappings linking the first two [Lenzerini, 2011].

So, to provide the Curator with access to the factual data in the database through the lens of the ontology, this subsection details the developed mapping between the bdME database and the OntoME ontology using the Ontop framework. In that way, the user can access the source through the OntoME's vocabulary.

The Ontop mapping needs to connect the classes and properties (datatype and object properties) of the ontology with views (SQL) over the repository's data via a specification, as shown in Listing 12.6.

Listing 12.6 presents the mapping declarations for the bdME *identificacaoEmigrante*, *filiacao*, and *localidade* tables previously shown in Table 12.1. The specification of the mapping is composed of one or more mapping axioms[12] that intend to transform the repository's data into a set of RDF triples. An axiom has three fields:

- *mappingId*: a string that identifies the mapping axiom;
- *target*: an ontology triple template (subject, predicate, object) which references column names used in the database (source) query through placeholders (terms between curly brackets);

[12] More about the mapping axioms:
`https://github.com/ontop/ontop/wiki/ontopOBDAModel#Mapping_axioms`

- *source*: a SQL query over the database, which should contain the column names used in the target’s placeholders.

Listing 12.6: Mapping of bdME “identificacaoEmigrante”, “filiacao”, and “localidade” tables

```
[PrefixDeclaration]
...
[SourceDeclaration]
...
[MappingDeclaration] @Collection [[

mappingId E21.1_Emigrante
target :URI/Emigrante#{idEmigrante} a :E21.1_Emigrante ;
        :P3_has_note {idEmigrante} ;
        :P131_is_identified_by :URI/nomeEmigrante#{idEmigrante} ;
        :P98i_was_born :URI/nascimentoEmigrante#{idEmigrante} ;
        :P107_is_current_or_former_member_of :URI/Couple#{idEmigrante} ;
source SELECT idEmigrante FROM identificacaoEmigrante

mappingId EmigranteAppellation
target :URI/nomeEmigrante#{idEmigrante} a :E82_Actor_Appellation ;
        :P3_has_note {nome} ;
        :P2_has_type “Emigrante” ;
source SELECT idEmigrante, nome FROM identificacaoEmigrante

mappingId NascimentoEmigrante
target :URI/nascimentoEmigrante#{idEmigrante} a :E67_Birth ;
        :P4_has_time-span :URI/dataNascimentoEmigrante#{dtNasc} ;
        :P96_by_mother :URI/Filiacao#{idFiliacao} ;
        :P97_from_father :URI/Filiacao#{idFiliacao} ;
        :P98_brought_into_life :URI/Emigrante#{idEmigrante} ;
        :P7_took_place_at :URI/Naturalidade#{idLocalidade} ;
source SELECT idEmigrante, dtNasc, idFiliacao, idNaturalidade as idLocalidade FROM identificacaoEmigrante

//other mapping rules ...
]]
```

As already mentioned in Section 10.1, the patterns found in the mapping axioms shall be expanded to new rules composed of *mappingId*, *target*, and *source*. So the other mapping axioms related to the placeholders with the `URI`[13] word (e.g. `:URI/Filiacao#{idFiliacao}`, etc.) must be done in the same OBDA file.

In addition to the mapping declaration of the OBDA model, there are two other sections for declaring the prefixes (`[PrefixDeclaration]`) and the source (`[SourceDeclaration]`) needed by the mapping statement. This should be done before the `[MappingDeclaration]` section (see Listing 12.6).

[13] Notice that the use of the `URI` word in this mapping collection was decision of the author.

To better illustrate the mapping axioms, Figure 12.11 presents the same mapping declarations of Listing 12.6, but in a graphical format.

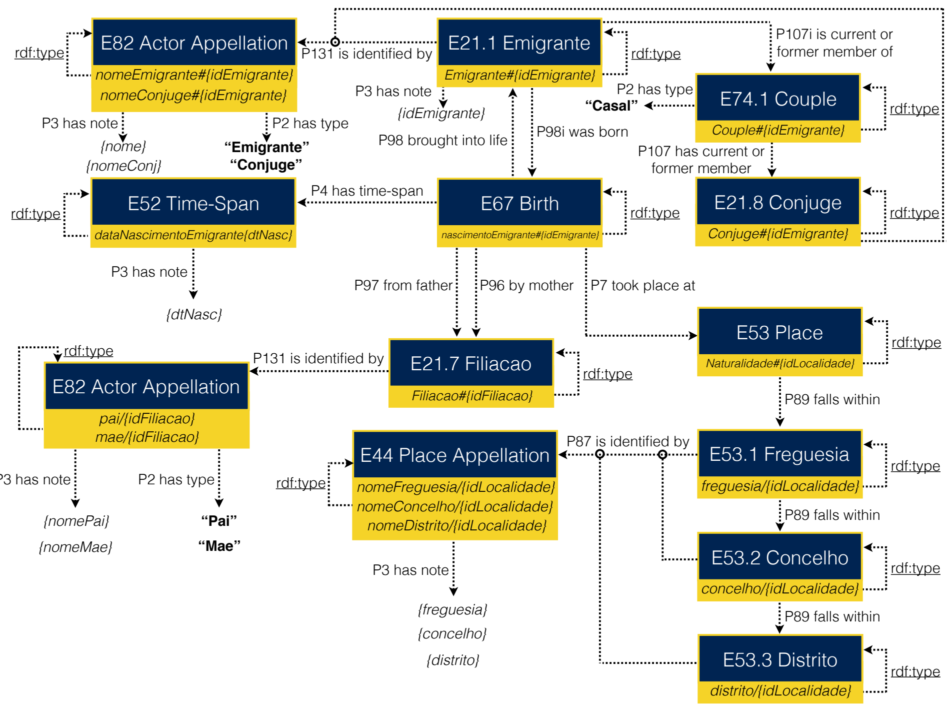


Figure 12.11: Graphical example of the mapping axioms from Listing 12.6

The documental fond can be described in different ways. In this example, the OntoME ontology was extended to cover seven new domain concepts. Besides, the property *P2 has type*, a form of specialization, was used.

Th next section presents the CaVa$^{\text{DSL}}$ specification to describe the Emigration virtual LS based on the OntoME ontology's vocabulary.

## 12.6 A CaVa$^{DSL}$ specification for the Emigration virtual Learning Space

After getting all the above prerequisites to specify a virtual Learning Space, i.e., after building the bdME relational database (Section 12.2), the OntoME ontology (Section 12.4) and the mapping (Section 12.5) between them, it is time to write the specification of the Emigration virtual Learning Space in CaVa$^{DSL}$ (file named *mef.cava*). That specification will be presented in this section.

As this work was done in cooperation with the Municipal Archive of Fafe, the responsible archivist Mónica Guimarães specified the Emigration virtual LS according to her experience in the emigration documents based on the OntoME's vocabulary and the CaVa$^{DSL}$ rules.

The outcome of this specification, for this specific domain, was built up from four separated blocks, following the structure of CaVa$^{DSL}$ (described in Chapter 9).

### The main configuration (*mainconfig* element):

Listing 12.7: *mainconfig* element

```
mainconfig [
    LS title:  “Museu da Emigração de Fafe”,
    about [
        p:  “A emigração sempre foi e continua a ser uma das
            constantes da História de Portugal, desde a epopeia
            das Descobertas à actualidade.”,
        p:  “Todavia, apesar das raízes multisseculares,
            apenas a partir da segunda metade do século XIX e após
            a independência do Brasil (1822) e a emergência do
            Liberalismo, esse movimento humano, a partir de Portugal,
            se intensifica, sobretudo rumo a Terras de Vera Cruz.”,
        p:  “A causa geral do fenómeno migratório num país de
            ‘estrutura agrária ainda rotineira ou insuficientemente
            inovadora’, como o nosso, era ‘o baixo nível económico da
            população rural’ (Joel Serrão), o crescimento demasiado lento
            e incapaz de acorrer às necessidades da população.  Dadas as
            facilidades concedidas pelas vias férreas de ligação ao litoral
            e aos portos de embarque para aqueles que almejavam por uma
            ascensão rápida do ponto de vista económico, a emigração foi
            engrossando, mormente para o outro lado do Atlântico, após
            um fenómeno que havia de ser decisivo na mudança dos
            acontecimentos:  a extinção da escravatura e a necessidade
            de atrair mão-de-obra agrícola para a substituir.  Porém, seria
            nas cidades e nas actividades mercantis, sobretudo no comércio
            a retalho, que muitos ‘brasileiros de torna viagem’ enriqueceriam.",
        ]
        carousel [
            interval:  8,
            images [
                caption:  "Centro da Vila de Fafe", src:  "carousel1.png",
                caption:  "Passaporte", src:  "carousel2.png", active,
                caption:  "Registo de emigrantes", src:  "carousel3.png",
                caption:  "Museu da Emigração de Fafe", src:  "carousel4.png",
            ]
        ]
    ]
```

The *mainconfig* element describes the most generic characteristics of the virtual LS. The specification of Listing 12.7 describes the title and the three paragraphs about the virtual LS domain (emigration) that will be designed in the entrance room (the website homepage). Additionally, a carousel of images was defined. This carousel, in addition to the caption and source of each image, also sets a time interval (in seconds) for the transition between each of them. The “active” value defines the initial image in the carousel when the main page of the virtual LS is loaded.

After describing the main configuration of the virtual Learning Space, it is time to define the header (*menu*).

### The header (*menu* element):

Listing 12.8: *menu* element

```
menu [
    brand:  "Museu da Emigração de Fafe",
    background color:  crimson,
    foreground color:  white,
    behavior:  fixed,
    options [
        label:  "Exibições", dropdown [
            dropdown label:  "Todas", url:  "exhibitions",
            dropdown label:  "Permanentes", url:  "permanent_exhibitions",
            dropdown label:  "Temporárias", url:  "temporary_exhibitions",
            dropdown label:  "Especiais", url:  "especial_exhibitions",
            dropdown label:  "Futuras", url:  "future_exhibitions",
        ]
        label:  "Sobre", url:  "about", extension:  php,
    ]
]
```

The *menu* specification shown in Listing 12.8 specifies the header of the entire virtual Learning Space. This component described the brand, the colors, the behavior and the options of the menu (dropdown and simple). Notice that the URL must be defined, so that the generator (CaVa$^{gen}$) can create the right pages to jump to when the option is selected.

After specifying the header, the content (a list of exhibitions) is described in Listing 12.9.

### The content (*exhibitions* element):

Listing 12.9: *exhibitions* element

```
exhibitions [
    exhibition [
        title:  "Reflexos da Emigração no concelho de Fafe nas décadas,
                de 60 a 80 do século XX",
        short description:  "Mostramos nessa sala de exibição
                            a lista de emigrantes, os quais emigraram
                            para diversos destinos, dentre eles:
                            França, Brasil, África, Canadá, etc.
                            Nessa sala de exibição você poderá
                            encontrar os dados demográficos
                            de cada emigrante.",
        icon:  "file-o",
        additional info [
            title:  "4000+",
            description:  "Processos",
        ]
        behavior:  expanded,
        type:  permanent,
        E21.1_Emigrante->all("Listagem de Emigrantes", "mef.obda",
                             "http://semanticweb.org/rgm/2018/ontoME/")"
                             ["Identificação do Emigrante",
                             headerOfEachElement:"Nome do Emigrante",
                             "Data de Nascimento", "Mãe", "Pai",
                             "Freguesia", "Concelho", "Distrito",
                             "Nome do Cônjuge"],
    ]
    # other exhibitions . . .
]
```

The content of a CaVa virtual Learning Space is specified through a list of exhibitions. Each exhibition is declared inside the *exhibitions* element. In this particular case, there is only one exhibition specified. It is a permanent type exhibition room. The referred description includes the definition of the exhibition title, an exhibition short description, the icon to be used, the additional information, and the expected component (accordion) behavior, that in this case is "expanded".

In addition to the static content of this exhibition room, a query operator (*all()*) was provided. The concept passed to it is *E21.1_Emigrante*, which means that the CaVa system needs to search for all occurrences of the specified concept in the bdME instances based on the mappings described in Subsection 12.5.2. The parameters are the title of the instances list, the OBDA mapping file, and the OntoME ontology. As optional, if the order of the queried fields is known, they can be specified, as seen in the list of labels inside the brackets after the third parameter. If the list is not specified, the CaVa system will take the property annotations (`rdfs:label`) of the ontology (e.g. P131_is_identified_by with the annotation "is identified by") as labels for the final rendering of the virtual LS. If there is no annotation, the name of the property is taken. To conclude the virtual Learning Space with all elements, the footer is specified in Listing 12.10.

### The footer (*footer* element):

Listing 12.10: *footer* element

```
footer [
    images [
        image:  "cava_logo.png",
        alignment:  right,
    ]
    format date:  "Y",
    developer [
        name:  "Ricardo Giuliani Martini",
        alignment:  left,
    ]
    behavior:  fixed,
    style:  extended [
        title:  "Redes Sociais", subtitle:  "veja também"[
            label:  "facebook/municipiofafe",
            link:  "https://www.facebook.com/municipiofafe/",
            icon:  "facebook", icon color:  blue,
        ]
        title:  "Parceiros", subtitle:  "visite"[
            label:  "Museu das Migrações e das Comunidades",
            link:  "http://www.museu-emigrantes.org/",
            icon:  "institution", icon color:  black,
        ]
        title:  "Município de Fafe",
            subtitle:  "Avenida 5 de Outubro - 4824-501 Fafe"[
            label:  "TELF. 253 700 400",
            link:  "tel:+351253700400",
            icon:  "phone", icon color:  black,

            label:  "TELF. 253 700 409",
            link:  "tel:+351253700409",
            icon:  "fax", icon color:  black,

            label:  "geral@cm-fafe.pt",
            link:  "mailto:geral@cm-fafe.pt",
                icon:  "envelope", icon color:  black,
            ]
            title:  "Arquivo Municipal de Fafe",
                subtitle:  "Rua Major Miguel Ferreira - 4820 Fafe"[
                label:  "TELF. 253 700 470",
                link:  "tel:+351253700470",
                icon:  "phone", icon color:  black,

                label:  "arquivo@cm-fafe.pt",
                link:  "mailto:arquivo@cm-fafe.pt",
                icon:  "envelope", icon color:  black,
            ]
        ]
    ]
```

The *footer* element shown in Listing 12.10 describes the footer of the whole virtual Learning Space. This footer is composed of the CaVa logo aligned on the right side, the current year, the developer name aligned on the left side, and a behavior. The style of the footer was set up as extended, which means that the element contains more options. In this case, the options were defined as “Redes Sociais” (Social Networks), “Parceiros” (Partners), “Município de Fafe” (the address and contacts of the Fafe county), and “Arquivo Municipal de Fafe” (the address and contacts of the Municipal Archive of Fafe).

Note that the four blocks described must be composed into a unique *.cava* file in the order that they were specified in this section.

The processing of the CaVa$^{\text{DSL}}$ specification here described is done by CaVa$^{\text{gen}}$, which is explained in Section 12.7.

## 12.7 CaVa$^{\text{gen}}$ applied to the automatic generation of the Emigration virtual LS

As the CaVa$^{\text{DSL}}$ specification (*mef.cava*) for the emigration virtual LS is written, this section reports on the processing of such specification through the processors of CaVa$^{\text{gen}}$ already presented in Chapter 10, aiming at generating the final virtual LS.

Based on the workflow presented in Figure 10.2, after step 1 is done (Section 12.6), it is needed to transform the input (*mef.cava*) into the outputs generated by step 2 and step 3 of the CaVa workflow. These actions should be done with the aid of CaVa$^{gen}$. Next, the gradual processing of *mef.cava*, until reaching the final virtual Learning Space about the emigration phenomena in Portugal, is explained.

The first task to be performed is to feed CaVa$^{structure}$ with the *mef.cava* file as input (represented by the edge (1) of Figure 10.3). The abstract CaVa$^{structure}$ as sketched in Figure 10.3 was realized in this concrete case study by the following implementation:

- CaVa$^{DSL}$ Specification $\mapsto$ *mef.cava* file;
- CaVa$^{grammar}$ Processor $\mapsto$ ANTLR;
- CaVa State file $\mapsto$ plain text (*.txt*);

Notice that CaVa$^{grammar}$, a Context Free Grammar (CFG) listed in Appendix A.1 is fixed (always the same) and is not instantiated in each project.

Figure 12.12 depicts CaVa$^{structure}$ in a concrete form, presenting the filetypes and technologies used.

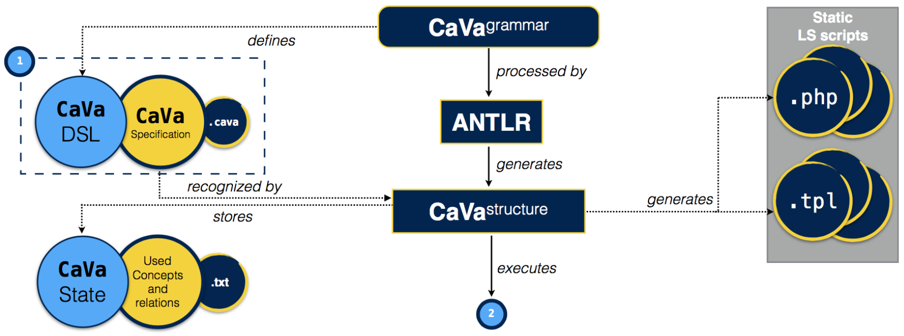


Figure 12.12: Concrete instance of the CaVa$^{structure}$ processor schema

It is important to note that CaVa$^{grammar}$– ANTLR version – has no embedded actions (it is a pure CFG). This results in a clean grammar (i.e. easy to read) that keeps the application-specific code out of the grammar [Parr, 2013]. This approach supports multiple uses, independent of the context and implementation (in this case the set of application generators ($\text{CaVa}^{\text{gen}}$) was developed based on the concept of ANTLR Listeners (see more in [Parr, 2013])).

So, to recognize the *mef.cava* input, ANTLR needs to process the $\text{CaVa}^{\text{grammar}}$ to generate the $\text{CaVa}^{\text{structure}}$ skeleton. The generated skeleton is composed of a Java listeners Interface (named `CavaListener`) with all methods based on the $\text{CaVa}^{\text{grammar}}$ rules, and an empty implementation of the `CavaListener` (named `CavaBaseListener`), which is extended with the creation of a new class (`CavaInterfaceListener`) that implements this bare bone code, overriding the original methods to deal with a subset of the available listeners.

The $\text{CaVa}^{\text{structure}}$ processor, when recognizing the *mef.cava* input file, assigns and executes the defined actions of the corresponding listener for each production rule of $\text{CaVa}^{\text{grammar}}$. To exemplify, at the beginning of the input recognition, the method `enterCava()` is triggered, because it is the initial rule of $\text{CaVa}^{\text{grammar}}$. This listener creates the `header`, `content`, and `footer` files to be filled by each responsible listener.

After the main files are created, the next production rule to be recognized is 'mainConfig', which has an associated listener, called `enterMainConfig()`. Thus, recognizing each production rule and executing the respective listener actions, the final virtual Learning Space is generated.

The generation of code through listeners for the 'mainConfig' rule is shown in Listing 12.11. Note that there are other code blocks to be generated, depending on the remaining methods to be executed. The listeners presented in Listing 12.11 are only an excerpt of the generation of content (PHP code) for the 'mainconfig' element of $\text{CaVa}^{\text{DSL}}$. As the 'mainconfig' element is considered content of the virtual LS (different from the header and footer), a file named `learningSpaceAndExhibitionSettings.php` was created to handle it and the exhibitions.

Listing 12.11: The implementation of the ‘mainConfig’ production rule methods of the *CavaInterfaceListener* class

```java
public class CavaInterfaceListener extends CavaBaseListener {
    @Override public void enterMainConfig(CavaParser.MainConfigContext ctx) {
        lsAndExhibitionSettings += "\t$data = array(\n";
    }

    @Override public void enterLearningSpaceTitle (CavaParser.LearningSpaceTitleContext ctx) {
        lsAndExhibitionSettings += "\t\t'lsTitle' => ";
        lsAndExhibitionSettings += ctx.TEXT().getText() + ",\n";
    }

    @Override public void enterLearningSpaceAbout (CavaParser.LearningSpaceAboutContext ctx) {
        lsAndExhibitionSettings += "\t\t'about' => array(\n";
    }

    //other overriding methods ...
}
```

The method *enterMainConfig()* concatenates the string “\t$data = array(\n” to the variable *lsAndExhibitionSettings*. The listener *enterLearningSpaceTitle()*, in addition to concatenate the ‘lsTitle =>’ string to the *lsAndExhibitionSettings*, it asks the context object for the string `TEXT()` token being matched by the parser for the invocation rule `LearningSpaceTitle` (in this case the text is “Museu da Emigração de Fafe”). According to [Parr, 2013], “context objects record everything that happens during the recognition of a rule”. A similar process is executed for the other overridden methods for generating the Static LS Scripts.

After parsing all the *mef.cava* input file (excluding the exhibitions query operators), CaVa$^{structure}$ generates the ‘Static LS Scripts’ for the Emigration virtual Learning Space.

Listing 12.12 shows an example of an PHP output file, `learningSpaceAnd ExhibitionSettings.php` created from the CaVa$^{DSL}$ specification of Listing 12.7 using ANTLR Listeners, like the excerpt in Listing 12.11.

Listing 12.12: Generated PHP code for creating the mainconfig element according to *mef.cava*

```
<?php
    $data = array(
        'lsTitle'=>"Museu da Emigração de Fafe",
        'about'=>array(
            array('p'=>"A emigração sempre foi e continua a ser ..."),
            array('p'=>"Todavia, apesar das raízes ..."),
            array('p'=>"A causa geral do fenómeno migratório num ..."),
        ),
        'carousel'=>array(
            'active'=>"true",
            'interval'=>8000,
            'images'=>array(
                array('caption'=>"Centro da Vila de Fafe", 'src'=>"carousel1.png", 'active'=>"false"),
                array('caption'=>"Passaporte", 'src'=>"carousel2.png", 'active'=>"true"),
                array('caption'=>"Registo de emigrantes", 'src'=>"carousel3.png", 'active'=>"false"),
                array('caption'=>"Museu da Emigração de Fafe", 'src'=>"carousel4.png", 'active'=>"false"),
            ),
        ),
    //continues with the exhibitions list element PHP code ...
```

However, when CaVa$^{\text{structure}}$ recognizes the query operator in the *exhibitions* list element, it reads and stores the used concept as a CaVa State file (*concept.txt*) to recognize later that concept on the mappings specification created during step 2 (Figure 10.2) of the CaVa workflow.

When the operator *all()* is recognized in the specification of a virtual LS, as seen in the statement "`E21.1_Emigrante->all()[]`" in line 19 of Listing 12.9, the Java method *enterAll()* (associated to the respective grammar rule) is executed, calling the *generateSPARQL()* method. This method receives two parameters:

1. the first one specifies the mapping or triples file, depending on whether the digital repository is a relational database (as it happens to be in this case study) or a triple store. For this case study, the OBDA mapping file (*mef.obda*) is referenced;

2. the second one defines the concept recognized as the left hand side of the statement. In this case, the concept `E21.1_Emigrante`.

These two parameters are recognized through the context reached from the *mappingOrTriplesFileName()* method of the *parametersAll()* listener, that in this case is *CavaParser.AllContext*.

After detecting the filetype of the first parameter, *generateSPARQL()* calls the method *mainCaVaSPARQL()*, which is represented by the node with number (2) in Figure 12.12 (step 2 in Figure 10.2). It is the main method related to the processors that generate and assemble the queries in the CaVa system. As the data of this case study is stored in a relational database, the CaVa$^{queries}$ processor is committed to performing this task.

The input of the CaVa$^{queries}$ generator must be a mapping file, which in this case is the *mef.obda* file described in Listing 12.6. The abstract CaVa$^{queries}$ processor outlined in Figure 10.4 is realized in this case study by the following implementation:

- mappings specification $\mapsto$ *mef.obda* file;
- CaVa$^{queries}$ Processor $\mapsto$ ANTLR;
- CaVa State $\mapsto$ *concept.txt* file;
- Ontology $\mapsto$ OntoME;
- Queries $\mapsto$ SPARQL (*.rq*).

Notice that the mapping grammar, a CFG grammar (*cavaSPARQL*) developed specially for this project and presented in Appendix A.2, belongs to the CaVa framework, but is not instantiated (it is a fixed element).

The filetypes and technologies used for the CaVa$^{queries}$ processor are depicted in Figure 12.13.

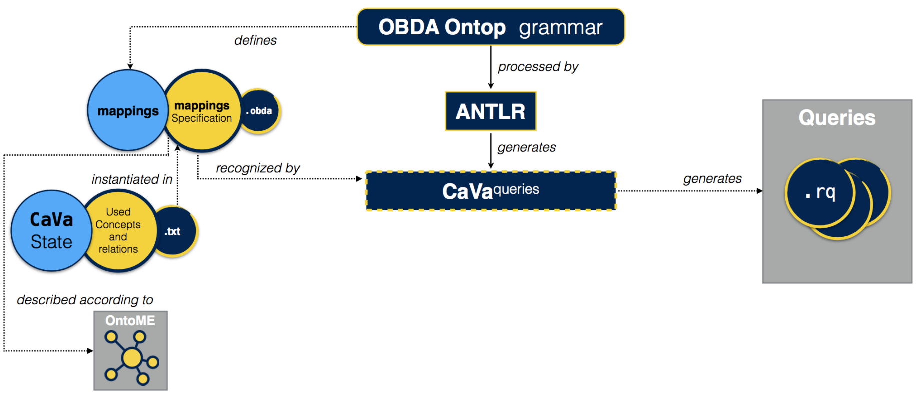


Figure 12.13: Concrete instance of the CaVa$^{\text{queries}}$ processor schema

Like CaVa$^{\text{grammar}}$, the OBDA Ontop grammar (named *cavaSPARQL*) is an ANTLR version once again without embedded semantic actions. As already said, along this PhD project, it was followed, as much as possible, a policy of reuse of well-defined grammars. In this specific case, for the OBDA format, as there is no access to its grammar, it was decided to create a new one to recognize the OBDA mapping axioms. This new grammar was created based on a study over several OBDA example files available from the Ontop project[14].

The main production rules of OBDA Ontop grammar are shown in Listing 12.13.

Listing 12.13: Main production rules of *cavaSPARQL* grammar

```
grammar cavaSPARQL ;
cavaSPARQL: (mappingid comma target)+ ;
target:  uri relation object (relations)* period ;
```

Similar activities performed to generate the skeleton for the CaVa$^{\text{structure}}$ processor are used for CaVa$^{\text{queries}}$. The skeleton is composed of a Java Interface (`CavaSPARQLListener`) and the empty implementation of the Interface (named `CavaSPARQLBaseListener`), which is extended with the creation of a new class, named `cavaSPARQLInterfaceListener`, which implements the empty base class.

[14] Website and github available at: `https://github.com/ontop`

The CaVa$^{queries}$ generator recognizes the *mef.obda* input file and executes the right actions of the corresponding listener for each production rule of the *cavaSPARQL* grammar until reaching the final SPARQL query file.

As already reported in Chapter 10 (section about CaVa$^{queries}$), five steps are needed to generate the query:

1. Searches for occurrences of the ontology concept (like in line 20 of Listing 12.9 – `E21.1_Emigrante`) in the mapping file *mef.obda* through a method called `checkConceptFile(String pConcept)`;
2. Expands each mapping axiom related to the query imposed in *mef.cava*;
3. Stores all the mapping axioms related to the desired query in a Java List, expanded in task 2;
4. Transforms each placeholder related to a literal in a *SELECT* clause variable with name ?p___0, ?p___1, depending on how many literals are found; Also, transforms each placeholder (*concept+literal*) in a new variable of the *WHERE* clause;
5. Creates and writes the generated SPARQL query file.

For the expansion step of task 2, from the first mapping axiom found (in this case the mapping rule with mappingId *E21.1_Emigrante* of Listing 12.6), it is necessary to extend each placeholder that is related to a *concept + literal*, i.e., all the placeholders with the `URI` word. The result is a Java List. The content of this list is shown in Table 12.2.

Table 12.2: Expansion of the mapping axioms to create the SPARQL query

| *Expanding mapping rules* | | | | |
|---|---|---|---|---|
| **Initial var to be derived** | **rdftype** | **Concept** | **Object Property** | **var to be derived or final value** |
| ?Emigrante | a | E21.1_Emigrante | P3_has_note | ?p___0 |
| - | - | - | P131_is_identified_by | ?nomeEmigrante |
| - | - | - | P98i_was_born | ?nascimentoEmigrante |
| - | - | - | P107i_is_current_or _former_member_of | ?Couple |
| ?nomeEmigrante | a | E82_Actor_Appellation | P3_has_note | ?p___1 |
| - | - | - | P2_has_type | "Emigrante" |
| ?nascimentoEmigrante | a | E67_Birth | P4_has_time-span | ?dataNascimentoEmigrante |
| - | - | - | P96_by_mother | ?mae |
| - | - | - | P97_from_father | ?pai |
| - | - | - | P98_brought_into_life | ?Emigrante |
| - | - | - | P7_took_place_at | ?Naturalidade |
| ?dataNascimentoEmigrante | a | E52_Time-Span | P3_has_note | ?p___2 |
| ?mae | a | E82_Actor_Appellation | P3_has_note | ?p___3 |
| - | - | - | P2_has_type | "Mae" |
| ?pai | a | E82_Actor_Appellation | P3_has_note | ?p___4 |
| - | - | - | P2_has_type | "Pai" |
| ?Naturalidade | a | E53_Place | P89_falls_within | ?freguesia |
| ?freguesia | a | E53.1_Freguesia | P87_is_identified_by | ?nomeFreguesia |
| - | - | - | P89_falls_within | ?concelho |
| ?nomeFreguesia | a | E44_Place_Apellation | P3_has_note | ?p___5 |
| ?concelho | a | E53.2_Concelho | P87_is_identified_by | ?nomeConcelho |
| - | - | - | P89_falls_within | ?distrito |
| ?nomeConcelho | a | E44_Place_Apellation | P3_has_note | ?p___6 |
| ?distrito | a | E53.3_Distrito | P87_is_identified_by | ?nomeDistrito |
| ?nomeDistrito | a | E44_Place_Apellation | P3_has_note | ?p___7 |
| ?Couple | a | E74.1_Couple | P107_has_current_or _former_member | ?Conjuge |
| - | - | - | P2_has_type | "Casal" |
| ?Conjuge | a | E21.8_Conjuge | P131_is_identified_by | ?nomeConjuge |
| ?nomeConjuge | a | E82_Actor_Apellation | P3_has_note | ?p___8 |

Notice that, from the first group of rules (four initial lines of Table 12.2), each "var to be derived" value (excluding the literal values denoted by the SELECT clause variables ?p___X) became a new "initial var to be derived". This configures a top-down derivation of the mapping axioms, expanding all of them until achieving the literals (final values). Seeking for not having an infinite loop in the expand task, the rule in red is not executed. This rule is related to the inverse object property.

After expanding and storing the mapping axioms, the fourth step is to get the variables of the SELECT and WHERE clauses. Two listeners handle this situation. Listing 12.14 presents the method `enterPlaceholder()` to get the variables of the SELECT clause and Listing 12.15 shows the listener `enterUri()` aiming at getting the variables of the WHERE clause.

Listing 12.14: Listener that performs the SPARQL *SELECT* clause variables generation

```
1  @Override public void enterPlaceholder(...PlaceholderContext ctx) {
2      mappings.get(i).add("?p___"+selectVarsCounter);
3      mappings.get(i).add("selectable");
4      //selectable is set just for control
5  }
```

Listing 12.15: Listener that performs the SPARQL *WHERE* clause variables generation

```
1  @Override public void enterUri(CavaSPARQLParser.UriContext ctx) {
2      mappings.get(i).add(ctx.getText());
3      mappings.get(i).add("?"+ctx.VALUE().get(0).getText());
4  }
```

As the listeners access the input file token by token based on the *cavaSPARQL* grammar (Listing 12.13), the methods in Listing 12.14 and 12.15 only care about getting the text of a specific production (in this case, URI and Placeholder) of the grammar. The index "0" (line 3) in Listing 12.15 refers to the text between ":" and "#" in the mapping file (*mef.obda*).

After having the two SPARQL clauses defined, it is time to concatenate them, forming the correct statements (syntax) of SPARQL queries. The same is done with the prefixes of the query. They can be found in the [prefixDeclaration] section of the mappings file and they are placed at the beginning of the query (.rq) file.

To finish this process, CaVa$^{queries}$ creates and writes the ".rq" file with all the content of the strings (prefixes and clauses (*SELECT* and *WHERE*)). The name of each file is given by the "query" string plus the name of the CaVa$^{DSL}$ operator (in this case *all()*) and the number of the exhibition room where the operator was called. As there is only one declared exhibition, the name of the query file is `queryAll1.rq`. Listing 12.16 presents the generated SPARQL query by CaVa$^{queries}$.

Listing 12.16: Generated query file fragment – "*queryAll1.rq*"

```
PREFIX    :  <http://semanticweb.org/rgm/2018/ontoME/>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>

SELECT ?p___0,?p___1,?p___2,?p___3,?p___4,?p___5,?p___6,?p___7,?p___8
WHERE {
    ?Emigrante a :E21.1_Emigrante ;
    :P3_has_note ?p___0 ; :P131_is_identified_by ?nomeEmigrante ;
    :P98i_was_born ?nascimentoEmigrante ;
    :P107i_is_current_or_former_member_of ?Couple .

    ?nomeEmigrante a :E82_Actor_Appellation ;
    :P3_has_note ?p___1 ; :P2_has_type "Emigrante" .

    ?nascimentoEmigrante a :E67_Birth ;
    :P4_has_time-span ?dataNascimentoEmigrante ; :P96_by_mother ?mae ;
    :P97_from_father ?pai ; :P7_took_place_at ?Naturalidade .

    ?Couple a :E74.1_Couple ;
    :P107_has_current_or_former_member ?Conjuge ;
    :P2_has_type "Casal" .

    ?dataNascimentoEmigrante a :E52_Time-Span ;
    :P3_has_note ?p___2 .

    ?mae a :E82_Actor_Appellation ;
    :P3_has_note ?p___3 ; :P2_has_type "Mae" .

    //other graph pattern declarations (triples)...
}
```

Using the abstract CaVa$^{\text{run}}$ processor pointed out in Figure 10.5, in this case study it was transformed into a concrete implementation through the following map:

- mappings Specification $\mapsto$ *mef.obda* file;
- Repository $\mapsto$ bdME;
- Ontology (TBox) $\mapsto$ OntoME;
- Queries $\mapsto$ SPARQL (*.rq*);
- Query Results $\mapsto$ JSON (*.json*);
- CaVa$^{\text{run}}$ Processor $\mapsto$ Quest reasoner;

Figure 12.14 depicts the CaVa$^{\text{run}}$ schema with the filetypes and technologies used.

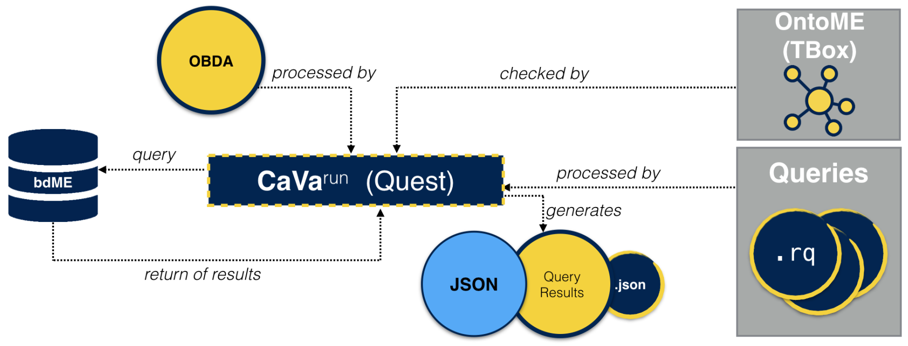


Figure 12.14: Concrete instance of the CaVa$^{run}$ processor schema

Assuming the code of Listing 12.6 as the OBDA model (mapping axioms), the schema of Figure 12.10 as the ontology, and the code of Listing 12.16 as the query to be executed by the CaVa$^{run}$ processor, its execution is explained in six steps:

1. Receives as input the mapping axioms (loading an external *.obda* file – in this case, "*mef.obda*"). Listing 12.17 shows the snippet for this specific action.

   Listing 12.17: Code fragment to load an external *.obda* file

   ```
   OBDADataFactory fac = OBDADataFactoryImpl.getInstance();
   OBDAModel obdaModel = fac.getOBDAModel();
   ModelIOManager ioManager = new ModelIOManager(obdaModel);
   ioManager.load("mef.obda");
   ```

2. Receives as input the ontology external IRI (International Resource Identifier). Listing 12.18 presents the code fragment for this specific action.

   Listing 12.18: Snippet to load an ontology from IRI

   ```
   OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
   OWLOntology ontology = manager.loadOntology(IRI.create("http://semanticweb.org/rgm/2018/ontoME/"));
   ```

3. Prepares the configuration for the Quest instance. The excerpt code of Listing 12.19 shows the setup for "virtual ABox" mode[15] of Quest.

[15] Read more at: https://github.com/ontop/ontop/wiki/ObdalibQuestIntro

Listing 12.19: Preparing the configuration for the Quest Instance

```java
QuestPreferences preference = new QuestPreferences();
preference.setCurrentValueOf(QuestPreferences.ABOX MODE, QuestConstants.VIRTUAL);
```

4. Creates the instance of Quest reasoner. Listing 12.20 shows up the code to create the instance of Quest reasoner based on the OBDA model of Listing 12.17, the configuration for the Quest Instance of Listing 12.19, and the ontology of Listing 12.18. As this case study uses the MySQL database, note that the first line of the code sets the Java Database Connectivity (JDBC) driver.

   Listing 12.20: Code to create the instance of Quest reasoner

```java
1   Class.forName("com.mysql.jdbc.Driver").newInstance();
2   QuestOWLFactory factory = new QuestOWLFactory();
3   factory.setOBDAController(obdaModel);
4   factory.setPreferenceHolder(preference);
5   QuestOWL reasoner = (QuestOWL) factory.createReasoner(ontology, new SimpleConfiguration());
```

5. Prepares the data connection for querying and executes the query. Listing 12.21 shows the code for these two particular actions. Note the "queryString" parameter in the fourth line, which for this case study is the "*queryAll1.rq*" file content.

   Listing 12.21: Code to prepare the data connection for querying and query execution

```java
1   QuestOWLConnection conn = reasoner.getConnection();
2   QuestOWLStatement st = conn.createStatement();
3   try {
4       QuestOWLResultSet rs = st.executeTuple(queryString);
5       //queryString is the string related to the content of
6       //the generated query by CaVa system
7
8       //other statements .  .  .
9   }
```

6. Writes the SPARQL query execution result set into a JSON file to be read by the specific exhibition room that called the operator. Listing 12.22 lays out the code fragment to write a JSON file with the query results. Notice that the *fileName* parameter (JSON file to store the results of the query) in line 2 of Listing 12.22 has the same pattern name of the query name, i.e., "*queryAll1.json*".

Listing 12.22: Code to write the query result into a JSON file

```java
try{
    FileWriter writer = new FileWriter(fileName);
    JsonGenerator gen = Json.createGenerator(writer);
    gen.writeStartObject();
    int indexFirstJSONObject = 0;
    while (rs.nextRow()) {
        gen.writeStartObject(Integer.toString(indexFirstJSONObject));
        for (int idx = 1; idx <= columnSize; idx++) {
            OWLObject binding = rs.getOWLObject(idx);
            String val = binding.toString();
            val = val.replace("\"", "");
            gen.write(Integer.toString(idx-1), val);
        }
        gen.writeEnd();
        indexFirstJSONObject++;
    }
    gen.writeEnd();
    gen.close();
    rs.close();
} finally {
    //close connections and resources
}
```

After presenting these six tasks, the exhibition room (Listing 12.25) gets the JSON result file content (excerpt shown in Listing 12.23) to display the instances in the virtual LS exhibition room.

Listing 12.23: Excerpt of the JSON result file generated by CaVa$^{run}$– “*queryAll1.json*”

```json
{
    //other JSON objects ...
    "2227": {
        "0": "713204",
        "1": "Aníbal de Castro",
        "2": "1926-10-30",
        "3": "Rosa de Castro",
        "4": "",
        "5": "Fornelos",
        "6": "FAFE",
        "7": "BRAGA",
        "8": "Ana de Almeida"
    },
    //other JSON objects ...
    "3005": {
        "0": "2828624",
        "1": "José Carlos Magalhães",
        "2": "1944-09-27",
        "3": "Felismina de Freitas",
        "4": "Serafim de Magalhães",
        "5": "Fornelos",
        "6": "FAFE",
        "7": "BRAGA",
        "8": ""
    },
    //other JSON objects ...
    "3315": {
        "0": "9747541",
        "1": "Laurinda Costa",
        "2": "1942-09-22",
        "3": "Felismina Gonçalves",
        "4": "José da Costa",
        "5": "Fornelos",
        "6": "FAFE",
        "7": "BRAGA",
        "8": "Américo Fernandes"
    },
    //other JSON objects ...
}
```

To get the result presented in Listing 12.23, the exhibition room calls the command *shell_exec()* passing all the necessary parameters. At this point, CaVa$^{\text{structure}}$ proceeds with the execution and generates the exhibition room client and server-side files (as the "exhibition1.tpl" file of Listing 12.24 and "exhibition1.php" of Listing 12.25).

Listing 12.24: Automatically generated template code for an exhibition room – "*exhibition1.tpl*"

```
{assign var=accordionID value=1}
{assign var=accordionName value=accordion}
{assign var=countID value=1}
{assign var=indexLabels value=0}
<div class="row">
    <div class="container">
        <div class="col-lg-12">
            <div class="ibox float-e-margins {if $data.collapsed eq 'collapsed'}collapsed{/if}">
                <div class="ibox-title">
                    <h5><i class="fa fa-file-o"></i> Listagem de Emigrantes</h5>
                    <div class="ibox-tools">
                        <a class="collapse-link">
                            <i class="fa fa-chevron-up"></i>
                        </a>
                    </div>
                </div>
                <div class="ibox-content">
                    <div class="panel-body">
                        <div class="panel-group" id="accordion">
                            {foreach from=$data item=i key=k}
                            {if $k neq "collapsed" and $k neq "labels"}
                            <div class="panel panel-default">
                                <div class="panel-heading">
                                    <h5 class="panel-title">
                                        <a data-toggle="collapse" data-parent="#accordion"
                                           href="#{$accordionName}{$accordionID}">{$i['0']}
                                        </a>
                                    </h5>
                                </div>
                                <div id="{$accordionName}{$accordionID}" class="panel-collapse collapse">
                                    {assign var=accordionID value=$accordionID+1}
                                    <div class="panel-body">
                                        <div class="col-md-12">
                                            <div class="row">
                                                <div class="col-md-9">
                                                    <h2 class="font-bold m-b-xs">{$i['0']}</h2>
                                                </div>
                                            </div>
                                            {foreach from=$i item=j}
                                                {if $j eq ""}
```

```
                                                    {assign var=indexLabels value=$indexLabels+1}
                                                {elseif $j neq ""}
                                                    <div class="row col-md-12">
                                                        <hr>
                                                    </div>
                                                    <div class="row col-md-9">
                                                        <div class="label">{$data.labels[$indexLabels]}:</div>
                                                        {assign var=indexLabels value=$indexLabels+1}
                                                        <dl class="small m-t-md">
                                                            <dt class="label navy-bg">{$j}</dt>
                                                        </dl>
                                                    </div>
                                                {/if}
                                            {/foreach}
                                            {assign var=indexLabels value=0}
                                        </div>
                                    </div>
                                </div>
                            </div>
                            {/if}
                            {/foreach}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

Listing 12.25: Automatically generated PHP code for an exhibition room – "*exhibition1.php*"

```
<?php
    $sparqlQuery = "queryAll1.rq";
    $sparqlResult = "queryAll1.json";
    $mappingOrTriplesFile = "mef.obda";
    $IRIOntology = "http://semanticweb.org/rgm/2018/OntoME/";
    $jarFilePath = ".../mappingOnto2Database.jar ";
    shell_exec($jarFilePath . " " . $sparqlQuery . " " . $sparqlResult
                    . " " . $mappingOrTriplesFile . " " . $IRIOntology);
    $json = file_get_contents($sparqlResult);
    $data = json_decode($json, TRUE);
    $data ['labels'] = array(
        0 => "Identificação do Emigrante",
        1 => "Nome do Emigrante",
        2 => "Data de Nascimento",
        3 => "Mãe",
        4 => "Pai",
        5 => "Freguesia",
        6 => "Concelho",
        7 => "Distrito",
        8 => "Nome do Cônjuge",
    );
    $data ['collapsed'] = "expanded";
    $tpl = new SMTemplate();
    $tpl->render('exhibition1', $data);
```

The generated code ("*exhibition1.php*") file performs six tasks:

1. receives as input:

   (a) "*queryAll1.rq*" file as the SPARQL query;
   (b) "*queryAll1.json*" file to save the query's result;
   (c) "*mef.obda*" file as the OBDA mapping collection;
   (d) OntoME IRI ("*http://semanticweb.org/rgm/2018/OntoME/*");
   (e) the program ("*mappingOnto2Database.jar*") that in fact executes the mappings and the "*queryAll1.rq*" file.

2. calls the *shell_exec()* command with the received inputs;

3. gets the content of "*queryAll1.json*", filled by the application "*mappingOnto2Database.jar*";

4. stores in *$data* the "*queryAll1.json*" decoded content;

5. stores the labels for the operator *all()* (Listing 12.9) and declares a flag which defines the UI component behavior (expanded);

6. renders the exhibition room from the *$data* variable content. The last two lines of Listing 12.25 mean that the variable *$tpl* (an instance of the *SMTemplate* class) calls the method render (already defined in the CaVa system), passing the array *$data* to render the Smarty template *exhibition1.tpl* (see Listing 12.24). The placeholders of the *exhibition1* template are filled with the *$data* array information, resulting in the rendered *exhibition1* (Figure 12.19) in Section 12.8. Every generated exhibition room "*.php*" file, in CaVa, instantiates the *SMTemplate* class, looking for rendering the template ("*.tpl*") file associated.

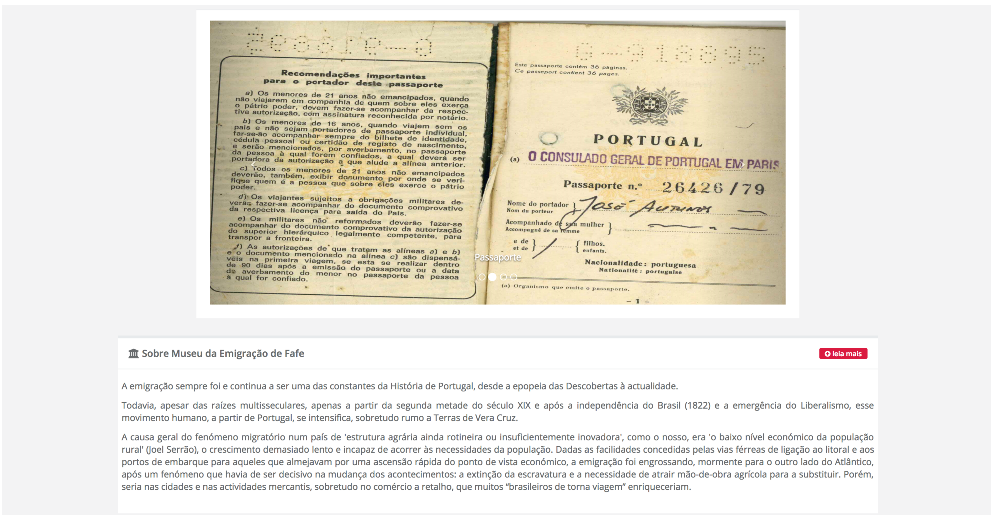
As the generated code to create the exhibition room calls *shell_exec()* every time the page is requested, at the moment the performance criteria is not taken into account, but in the future, it should be, because it can be a problem, since it consumes time trying to get the result of a query, when indeed the database was not changed. Maybe a kind of cache system shall be used to overcome that flaw.

The following section presents the rendered views of the "Museu da Emigração de Fafe" virtual Learning Space.

## 12.8 Rendering the final Emigration virtual LS with CaVa$^{\text{render}}$

As described in Chapter 11, the LS Scripts were generated by the CaVa$^{\text{gen}}$ set of processors. From these LS Scripts, CaVa$^{\text{render}}$ is the responsible for recognizing and rendering them through a web browser. So, this section displays screenshots that illustrate the outcome of rendering the generated LS Scripts, comprising the virtual Learning Space entitled "Museu da Emigração de Fafe". The order of the images is presented in accordance with the organization of Section 12.6, respecting the four main blocks of CaVa$^{\text{DSL}}$.

The *mainconfig* element, rendered, is shown in Figure 12.15.



A emigração sempre foi e continua a ser uma das constantes da História de Portugal, desde a epopeia das Descobertas à actualidade.

Todavia, apesar das raízes multisseculares, apenas a partir da segunda metade do século XIX e após a independência do Brasil (1822) e a emergência do Liberalismo, esse movimento humano, a partir de Portugal, se intensifica, sobretudo rumo a Terras de Vera Cruz.

A causa geral do fenómeno migratório num país de 'estrutura agrária ainda rotineira ou insuficientemente inovadora', como o nosso, era 'o baixo nível económico da população rural' (Joel Serrão), o crescimento demasiado lento e incapaz de acorrer às necessidades da população. Dadas as facilidades concedidas pelas vias férreas de ligação ao litoral e aos portos de embarque para aqueles que almejavam por uma ascensão rápida do ponto de vista económico, a emigração foi engrossando, mormente para o outro lado do Atlântico, após um fenómeno que havia de ser decisivo na mudança dos acontecimentos: a extinção da escravatura e a necessidade de atrair mão-de-obra agrícola para a substituir. Porém, seria nas cidades e nas actividades mercantis, sobretudo no comércio a retalho, que muitos "brasileiros de torna viagem" enriqueceriam.

Figure 12.15: *mainconfig* element rendered (the initial LS page)

Figure 12.15 illustrates the page resulted from the specification of Listing 12.7 processed by the CaVa platform. It presents the introdutory text defined in the *about* element and the images of the *carousel* with the second picture ("Passaporte") active.

The header of the virtual LS is described by the *menu* element of the CaVa$^{\text{DSL}}$ specification. It is depicted in Figure 12.16, following the description of

Listing 12.8.

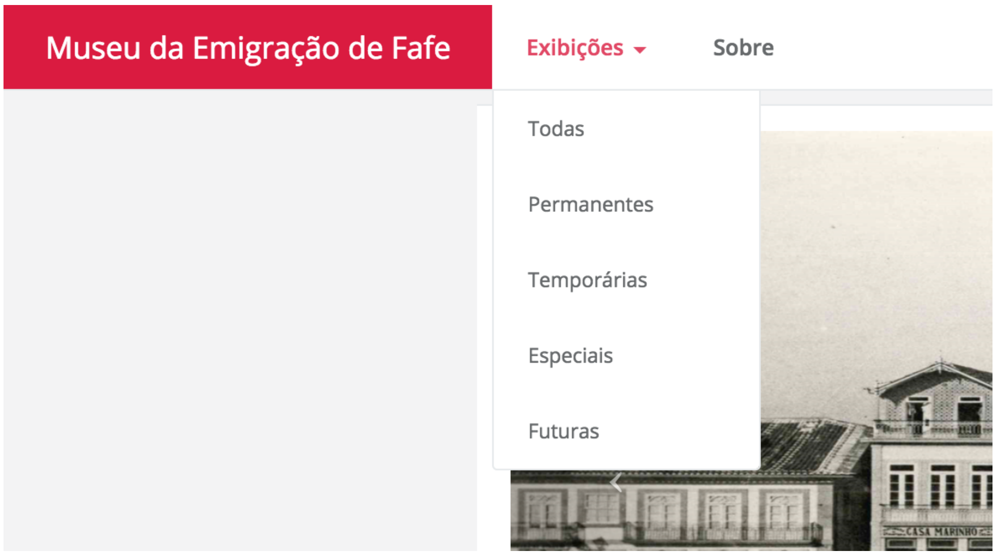
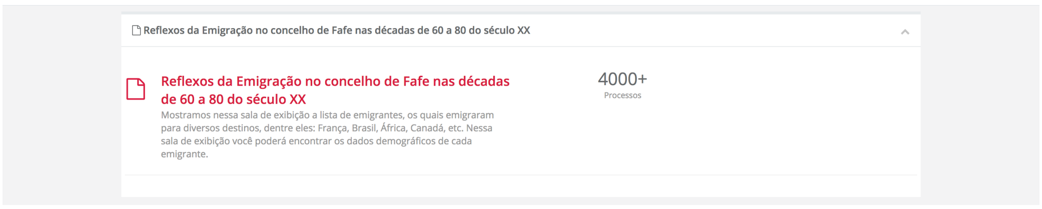


Figure 12.16: *menu* element rendered

The header of the virtual LS is composed of the *menu* element describing the *brand* ("Museu da Emigração de Fafe"), the *background color* ("crimson"), the *foreground color* ("white"), the *behavior* ("fixed") and a menu bar composed of two elements: one "Exibições" with five dropdown menus ("Todas", "Permanentes", "Temporárias", "Especiais", and "Futuras"); and one simple menu ("Sobre").

The *exhibitions* list element, in the CaVa system, is divided by type, so in this case study there is two ways to access it. The first one is through the *menu* (Figure 12.16) in the option "Exibições". The rendered list accessed via the menu option is presented in Figure 12.17.


Reflexos da Emigração no concelho de Fafe nas décadas de 60 a 80 do século XX

Reflexos da Emigração no concelho de Fafe nas décadas de 60 a 80 do século XX

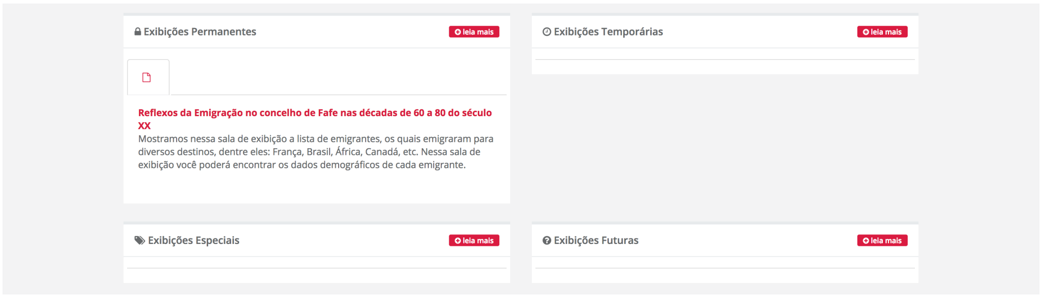
Mostramos nessa sala de exibição a lista de emigrantes, os quais emigraram para diversos destinos, dentre eles: França, Brasil, África, Canadá, etc. Nessa sala de exibição você poderá encontrar os dados demográficos de cada emigrante.

4000+
Processos

Figure 12.17: Exhibitions list accessed through the *menu* element

According to the specification of Listing 12.9, in Figure 12.17 it is possible to see the attributes *title*, *short description*, *icon* (“file-o”), *additional info*, and *behavior* (“expanded”).

The other way to access the list of exhibitions in CaVa can be seen in Figure 12.18, which presents all exhibitions in the initial page.


Exibições Permanentes
leia mais

Reflexos da Emigração no concelho de Fafe nas décadas de 60 a 80 do século XX

Mostramos nessa sala de exibição a lista de emigrantes, os quais emigraram para diversos destinos, dentre eles: França, Brasil, África, Canadá, etc. Nessa sala de exibição você poderá encontrar os dados demográficos de cada emigrante.

Exibições Temporárias
leia mais

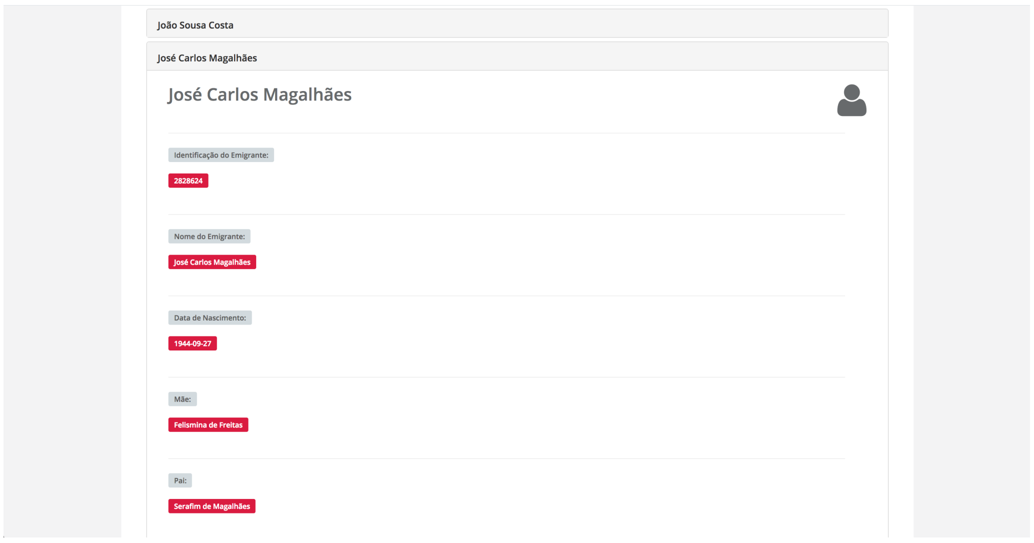
Exibições Especiais
leia mais

Exibições Futuras
leia mais

Figure 12.18: Exhibitions list accessed through the initial page

Based on the description of Listing 12.9, in Figure 12.18, the attributes *title*, *short description*, and *icon* are shown. Through the button labeled “leia mais” on the top right corner it is possible to access the list of exhibitions (rendered as in Figure 12.17) of the respective kind (e.g., permanent, etc.).

To see the content of the exhibition room (query result), it is necessary to access, through one of the two ways presented, the desired exhibition room. Figure 12.19 presents the *exhibition1.php* rendered.


João Sousa Costa

José Carlos Magalhães

José Carlos Magalhães

Identificação do Emigrante:

2828624

Nome do Emigrante:

José Carlos Magalhães

Data de Nascimento:

1944-09-27

Mãe:

Felismina de Freitas

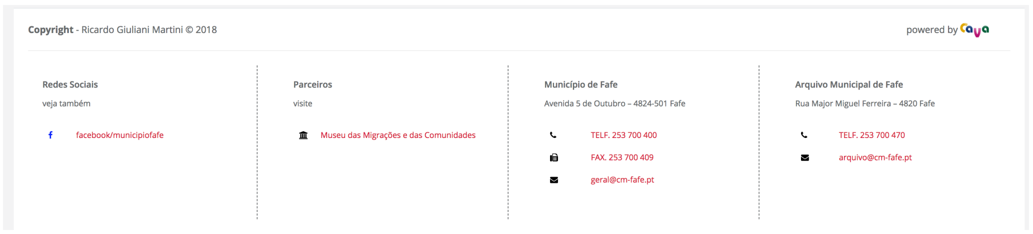
Pai:

Serafim de Magalhães

Figure 12.19: Exhibition1 content

The *exhibition1.php* file gives raise to an exhibition room that displays a list of all the instances of emigrant found in the database, according to the operator *all()* specified in Listing 12.9.

To finish the description of the entire virtual Learning Space created to display the emigration phenomena, the *footer* element described according to the Listing 12.10 is shown in Figure 12.20.


**Copyright** - Ricardo Giuliani Martini © 2018

powered by CaVa

Redes Sociais
veja também
facebook/municipiofafe

Parceiros
visite
Museu das Migrações e das Comunidades

Município de Fafe
Avenida 5 de Outubro – 4824-501 Fafe
TELF. 253 700 400
FAX. 253 700 409
geral@cm-fafe.pt

Arquivo Municipal de Fafe
Rua Major Miguel Ferreira – 4820 Fafe
TELF. 253 700 470
arquivo@cm-fafe.pt

Figure 12.20: *footer* element rendered

The *footer* contains *images*, *date*, *developer name*, *alignment* of the elements, and *style* ("extended"), describing links for "social networks", "partners", "addresses", and "contacts" related to the virtual LS.

Thus, putting up the four blocks of CaVa$^{\text{DSL}}$ specification (*mef.cava*), described in Section 12.6, together, the final Emigration virtual Learning Space is finalized as presented in this section.

## 12.9 Summary

The main goal of this case study was to resort to the digital repository of documents concerned with the emigration phenomena to show, step-by-step, all the specifications written and all the processing performed to use CaVa system to create the Emigration virtual Learning Space.

This chapter presented the application of the Emigration phenomena in the CaVa framework, aiming at automatically generating the virtual LS.

In this case study, the development of a relational database (bdME) for the domain in question was outlined. To populate the bdME repository, a web-based application was developed as an ingestion function called SGPE.

To achieve the repository instances through the abstract concepts and a controlled vocabulary, an ontology was designed (OntoME), which describes the instances of bdME in an abstract way, and a mapping developed between the bdME repository and the OntoME ontology, allowing the Curator to query the relational database via the ontology vocabulary.

The specification of the Emigration virtual LS was presented. The archivist of the Fafe's Archive, Mónica Guimarães, specified the LS according to her needs in CaVa$^{\text{DSL}}$ and the LS generation process guided by CaVa$^{\text{gen}}$ was detailed. As a result, the final virtual Learning Space was conceived. This case study showed some screenshots of the generated web-based LS.

So, to conclude, this case study was applied in the CaVa architecture and the objective of automatically generating a virtual LS based on a formal description using CaVa$^{\text{DSL}}$ and an ontology was successfully achieved.

However, the development of this project is much more than a mere academic exercise to be used as a proof of concept. Analyzing this archive collection, and extracting data to populate a database goes beyond this. Several opportunities were created, like:

- Potentiate the demand for cultural issues;
- Make available the documentary archive about the problematic of the emigrations in Portugal and in the world;
- Research, in collaboration with scientific and academic institutions, to share the knowledge about the problematic of the emigrations;
- Promote seminars, conferences, debates and other activities related to the phenomena associated with population movements;
- Promote the research about the emigrants in the host territories and in their return to Portugal, treating questions about architecture, industry, commerce, philanthropy, journalism and arts, among others.

In the context of this work, following the route of the emigrants is important, because this allows the recognition of the cultural trades, the difficulties in the adaptation, the prejudices that they suffered, among other relevant subjects. To sum it up, the study of the emigrants potentiated by the information system reported is essential to understand the development of the destination areas as well as the influences that the emigration causes in the society in the return of the emigrant to his country.

As future work, this project can be extended to bring together the information not only about the passport application form, but other sources like biographies, letters, ships' routes, etc. Thus, the final virtual Emigration Museum can be enriched with more knowledge about the emigration phenomena.

In order to fortify the use of relational databases as the sources for the CaVa system, Chapter 13 presents another case study, showing all the process from the sources until reaching the final virtual LS about the prosopographical repository of the *Fasti Ecclesiae Portugaliae* project.

# Chapter 13

# Case Study 2 – The prosopographical repository of the *Fasti Ecclesiae Portugaliae* project (Clero Catedralicio)

To illustrate the use of the CaVa system applied to different scenarios in order to demonstrate its adaptability, the case study about the *Fasti Ecclesiae Portugaliae* project is discussed in this chapter. A relational database will be used again to store the digital project repository; so a mapping will be specified between it and an ontology, which describes the database instances. This chapter presents only particular aspects of this case study, because it is very similar to the case study outlined in Chapter 12.

The *Fasti Ecclesiae Portugaliae* project had as objective the design of a relational database for the study of the Portuguese Cathedral Clergy in the middle age. The main idea behind this project was to make the assets available to the scientific community [Jorge et al., 2004].

The date of the documents included in the database (bdFasti) is between 1071, year of restoration of the first Portuguese diocese (Braga) after the

Muslim occupation, and 1325, the end of the reign of D. Dinis. The project holds more than 7000 documents of nine dioceses (Braga, Porto, Coimbra, Lamego, Guarda, Viseu, Lisboa, Évora e Silves) [Jorge et al., 2004].

The study about the clergy is based on the analysis of bishops (or archbishops in the case of Braga), as well as dignities, canons, and constitutional portions of the cathedrals, excluding the remaining auxiliary clergy, such as chaplains, vicars and clergymen, among others [Jorge et al., 2004].

The case study here reported is based on the publication entitled "Automatic Generation of Virtual Learning Spaces Driven by CaVa$^{DSL}$: An Experience Report" [Martini and Henriques, 2017a]. Next, the structure of the database is presented, showing examples of the data.

## 13.1 bdFasti, the *Fasti Ecclesiae Portugaliae* Project Database

This section presents the bdFasti schema and examples of the data contained in the database. Figure D.1 displays the physical model of bdFasti.

This database contains 17 tables. The main tables are "clerocatedralicio" (CodClerigo PK) and "documentos" (CotaDocumento PK), which are related by the "clerodocumentos" association table.

To exemplify the data included in the bdFasti database, Table 13.1 shows the table "clerocatedralicio" and Table 13.2 presents the table "documentos".

Table 13.1: "clerocatedralicio" table

| **clerocatedralicio** | | | |
|---|---|---|---|
| **CodClerigo** | **historiador** | **NomeProprio** | **Patronimico** |
| 1 | justiniana | Martinus | Iohannis |
| 2 | justiniana | Joham | Siluestre |
| ... | ... | ... | ... |
| 1220 | andre | Gundisalbus | |
| ... | ... | ... | ... |
| 3989 | mariofarel | Franciscus | Simeonis |
| 3990 | mariofarel | Petros | Martini |

Table 13.2: “documentos” table

| documentos | | | | | |
|---|---|---|---|---|---|
| **CD**[1] | **Arquivo** | **DataCron** | **Dioceses** | **TipoDoc** | **Sumario** |
| 1 | ADB[2] | 1318-Março-18 | Braga | Carta de Compra | Carta de compra da Quinta ... |
| 2 | ADB | 1318-Março-23 | Braga | Carta de Doação | Doação do herdamento de Subcunha ... |
| ... | ... | ... | ... | ... | ... |
| 6543 | TT[3] | 1248/05/21 | Coimbra | | O bispo eleito D. Egas, o cabido da Sé ... |

Notice that just a subset of all the fields and data rows are shown, not comprising all the columns and rows of the presented tables.

These two tables will be used to illustrate the mapping phase between bdFasti and OntoFasti in Section 13.3.

## 13.2 OntoFasti, an ontology for the *Fasti Ecclesiae Portugaliae* Project

OntoFasti is an ontology designed for the prosopographical repository of the *Fasti Ecclesiae Portugaliae* project domain. The ontology (Figure 13.1) contains:

- concepts like: “Archive”, “Belongings”, “Clergy”, “Date”, “Document”, “Person”, and “Place” (“Diocese” as a subclass of “Place”);
- relations[4] like: “belongs to”, “contains”, “has current location”, “has relative/has as dependent/historian”, “is cited in”, and “takes place at”;

[1] CotaDocumento
[2] Arquivo Distrital de Braga
[3] Instituto dos Arquivos Nacionais/Torre do Tombo
[4] The inverse relations were not taken into account for this section to keep the schema simple

- properties like:
  - "is identified by", "has name", and "has patronymic", describing the attributes in the small circles (cc), (n), and (p), respectively for the concept "Clergy";
  - "has name" (n) for the entity "Person";
  - "has description" (d) for "Archive";
  - "is located in" (p) for "Place" and as inheritance, for "Diocese";
  - "has date" (d) for the "Date" entity;
  - "is identified by", "has summary", and "has type", for the attributes (cd), (s), and (t), respectively for the "Document" concept.

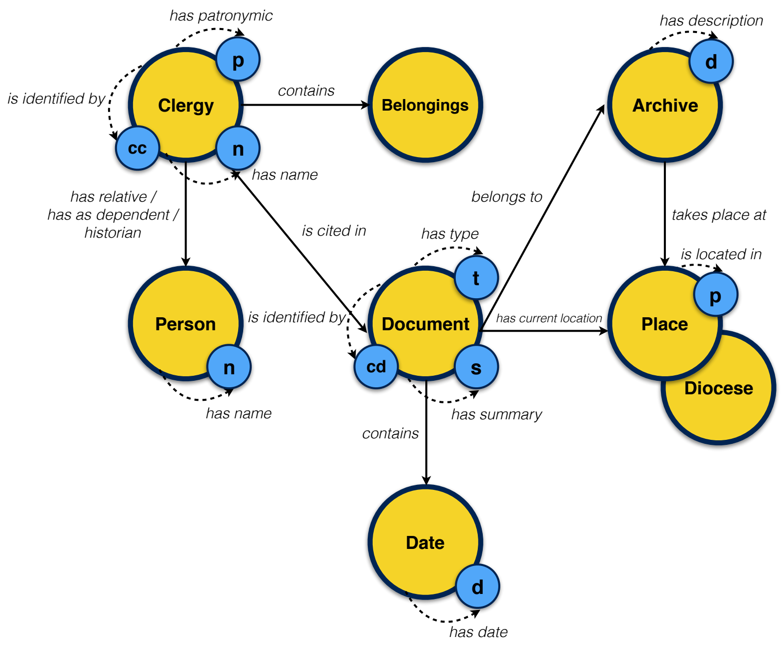


Figure 13.1: OntoFasti, an ontology for the *Fasti Ecclesiae Portugaliae* domain

So in order to create a pavement between bdFasti and OntoFasti for retrieving the database instances to put together in the final virtual LS about the *Fasti Ecclesiae Portugaliae* domain, Section 13.3 describes the mapping for this specific task.

## 13.3 Mapping between OntoFasti and bdFasti

To create the mapping axioms for this case study, the Ontop framework was used. So, based on the OntoFasti schema of Figure 13.1 and the bdFasti database (Figure D.1), Listing 13.1 presents the mapping rules for this case study.

Listing 13.1: Mapping of bdFasti "clerocatedralicio", and "documentos" tables – "clero.obda"

```
[PrefixDeclaration]
...
[SourceDeclaration]
...
[MappingDeclaration] @Collection [[

mappingId Document
target :URI/Document#{CotaDocumento} a :Document ;
       :is_identified_by {CotaDocumento} ;
       :has_type {TipoDoc} ;
       :has_summary {Sumario} ;
       :contains :URI/Date#{DataCron} ;
       :has_current_location :URI/Diocese#{Dioceses} ;
       :belongs_to :URI/Archive#{Arquivo} .
source SELECT CotaDocumento, TipoDoc, Sumario, DataCron, Dioceses, Arquivo FROM documentos

mappingId Clergy
target :URI/Clergy#{CodClerigo} a :Clergy ;
       :is_identified_by {CodClerigo} ;
       :has_name {NomeProprio} ;
       :has_patronymic {Patronimico} ;
       :historian :URI/Person#{historiador} ;
       :is_cited_in :URI/Document#{CotaDocumento} .
source SELECT cc.CodClerigo, cc.NomeProprio, cc.Patronimico, cc.historiador, d.CotaDocumento
       FROM clerocatedralicio as cc, documentos as d, clerodocumentos as cd
       WHERE d.CotaDocumento = cd.CotaDocumento
       AND cc.CodClerigo = cd.CodClerigo

//other mapping rules ...
]]
```

The mapping axioms of Listing 13.1 describe two ontological concepts of OntoFasti. The "Document" concept, which according to the target declaration, has three relations (namely "has_current_location", "contains", and "belongs_to") and three properties (namely "is_identified_by", "has_type", and "has_summary"). In *source*, it is specified the SQL instruction for fetching the instances of "CotaDocumento", "TipoDoc", "Sumario", "DataCron", "Dioceses" and "Arquivo" fields from "documentos" table.

The second concept is the "Clergy", which consists of two object properties (namely "historian", which refers to a "Person", and "is_cited_in", that refers to the "Document" entity) and three attributes ("CodClerigo" corresponding to the property "is_identified_by", "NomeProprio" related to the datatype property "has_name", and "Patronimico" associated to "has_patronymic"). The declaration in *source* is related to the SQL query to get the instances for the "CodClerigo", "NomeProprio", "Patronimico", "historiador", and "CotaDocumento" fields from three database tables ("clerocatedralicio", "documentos", and the association table "clerodocumentos").

As explained in Chapter 12, the main objective is to relate a database and an ontology, in order to query the sources via an abstract layer, putting together the database instances in the virtual LS.

So, having all the prerequisites for the specification of the described virtual LS, Section 13.4 outlines a CaVa$^{\text{DSL}}$ description for the *Fasti Ecclesiae Portugaliae* virtual Learning Space based on the OntoFasti vocabulary.

## 13.4 A CaVa$^{\text{DSL}}$ specification for the *Fasti Ecclesiae Portugaliae* virtual LS

Based on CaVa$^{\text{grammar}}$ presented in Appendix A.1 and the explanations in the previous chapters, Listing 13.2 presents a CaVa$^{\text{DSL}}$ specification based on the prosopographical repository of the *Fasti Ecclesiae Portugaliae* project, to illustrate how to define a virtual Learning Space with the same title of the project – "Fasti Ecclesiae Portugaliae".

Listing 13.2: A CaVa$^{\text{DSL}}$ specification for the *Fasti Ecclesiae Portugaliae* virtual LS – "fep.cava"

```
mainconfig [
    LS title:  "Fasti Ecclesiae Portugaliae",
    about [
        p:  "O projeto Fasti Ecclesiae Portugaliae compreende o estudo da
            Prosopografia do Clero Português.",
        p:  "O universo social em análise inclui bispos ou arcebispos, no caso de Braga,
            bem como dignidades, cónegos e porcionários constituitivos dos cabidos das catedrais.",
    ]
]

menu [
    brand:  "FEP",
    background color:  crimson,
    foreground color:  white,
    behavior:  fixed,
    options [
        label:  "Exibições", dropdown [
            dropdown label:  "Permanentes", url:  "permanentes",
            dropdown label:  "Temporárias", url:  "temporarias",
        ]
        label:  "Sobre", url:  "sobre_fep",
    ]
]

exhibitions [
    exhibition [
        title:  "Documentos FEP: 1071-1325",
        short description:  "Conjunto de 7792 documentos
                            da clerezia das catedrais portuguesas, desde a
                            restauração da primeira diocese do futuro reino,
                            Braga, em 1071 ...",
        icon:  "icon-archive",
        additional info [
            title:  "7792",
            description:  "Documentos",
        ]
        behavior:  expanded,
        type:  permanent,
        Document->all("Documentos FEP", "clero.obda",
        "http://semanticweb.org/rgm/2018/OntoFasti/")
    ]
    # other exhibitions .  .  .
]

footer [
    images [
        image:  "cava_logo.png",
        alignment:  right,
    ]
    format date:  "Y",
    developer [
        name:  "CaVa®",
        alignment:  left,
    ]
    behavior:  fixed,
    style:  condensed,
]
```

Listing 13.2 specifies a virtual Learning Space with a menu containing background color ("crimson"), foreground color ("white"), behavior ("fixed" – which means the menu stays on top of the screen and two sub-menus (one of type dropdown with two choices (1) Permanentes – that lists all permanent exhibitions; and (2) Temporárias – that lists all temporary exhibitions; and another, a simple menu with label “Sobre” (About))).

This specification exposes only one exhibition entitled “Documentos FEP: 1071-1325”, but on the “exhibitions” component, the curator can add as many exhibitions as he wants (note the “# other exhibitions . . .” comment). The exhibition specified contains a short description that tells what the exhibition is about, an icon (just for presentation) and an additional info (title “7792” and description “Documentos”).

Moreover, it is characterized with an “expanded” behavior – which means the exhibition component on the web page will appear opened –, a type (“permanent” exhibition), and the query operator:

*concept->all(“params”)[“list of attributes”]*

The query operator in the example is defining a query on the concept “Document”. This concept must be described on OntoFasti ontology (the third parameter of operator *all* — *http://semanticweb.org/rgm/2018/OntoFasti/*). The first parameter corresponds to the list name that is presented in the exhibition room (just for presentation). The second parameter is related to the OBDA mapping file (“clero.obda”). The list of attributes, in this case, was not defined, so the CaVa system will get the annotation properties as the labels for rendering the final exhibition room content.

To conclude the description of the virtual Learning Space for the *Fasti Ecclesiae Portugaliae* domain, the footer component is specified with some attributes like images, format date, developer name, behavior (“fixed” – similar to the menu’s behavior) and style (“condensed” – simple footer, the opposite of “extended”). The curator can also choose the alignment (images and developer’s information).

Next, in Section 13.5, the machinery for the generation of the virtual LS for this particular case study is explained.

## 13.5 CaVa$^{gen}$ applied to the automatic generation of the *Fasti Ecclesiae Portugaliae* virtual LS

Based on Chapter 10, this section deals with the processing of the CaVa$^{DSL}$ specification (Listing 13.2) to generate the virtual *Fasti Ecclesiae Portugaliae* Learning Space. As it is analogous to case study 1 (Chapter 12), only the specific characteristics of CaVa$^{gen}$ will be presented in detail.

To recognize the *fep.cava* specification file, the CaVa$^{structure}$ schema (Figure 10.3) is used with the subsequent implementation:

- CaVa$^{DSL}$ Specification $\mapsto$ *fep.cava* file;
- CaVa$^{grammar}$ Processor $\mapsto$ ANTLR;
- CaVa State file $\mapsto$ plain text (*.txt*);

Notice again that CaVa$^{grammar}$ (Appendix A.1) is not instantiated, because it is always the same for this project.

To deal with the *fep.cava* input file, ANTLR processes CaVa$^{grammar}$ and generates the CaVa$^{structure}$ application skeleton. The generated skeleton is the same of that explained in Section 12.7. So, all the listeners implementation based on the skeleton is the same. The only difference is in the code produced, which depends on the context (*ctx*) caught by the listeners. So, for instance, the code of Listing 12.11 (lines 6 to 9) defines the statement to generate the "LS title" in PHP code. For case study 1, the title caught by the listener context was "Museu da Emigração de Fafe". For this case study, the context gets a different value, which is "Fasti Ecclesiae Portugaliae", as specified in Listing 13.2. For the rest of the generation process, the same occurs.

To exemplify the generated code for this case study, Listing 13.3 lays out the result of the SPARQL query assembly by CaVa$^{\text{queries}}$ for the CaVa$^{\text{DSL}}$ specification of Listing 13.2.

Listing 13.3: Generated SPARQL query file excerpt – "*queryAll1.rq*"

```
PREFIX    :  <http://semanticweb.org/rgm/2018/OntoFasti/>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>

SELECT ?p___0,?p___1,?p___2,?p___3,?p___4,?p___5,?p___6
WHERE {
    ?Document a :Document ;
    :is_identified_by ?p___0 ;
    :has_type ?p___1 ;
    :has_summary ?p___2 ;
    :contains ?Date ;
    :has_current_location ?Diocese ;
    :belongs_to ?Archive .

    ?Date a :Date ;
    :has_date ?p___3 .

    ?Diocese a :Diocese ;
    :is_located_in ?p___4 .

    ?Archive a :Archive ;
    :has_description ?p___5  ;
    :takes_place_at ?Place .

    ?Place a :Place ;
    :is_located_in ?p___6 .
}
```

The biggest difference between case study 1 and this one is that in the execution of the generated SPARQL query, as the "list of attributes" is unknown, the CaVa$^{\text{structure}}$ processor gets back the execution flow and calls an extra processor that uses the ontology property names (`rdfs:label`) as the labels for the final exhibition room rendering. So, at this point, this processor searches for the property name, achieved from the mapping file "clero.obda" (Listing 13.1), in the ontology file description and returns the annotation text.

To reach the annotation property text from the ontology file, through the mapping rules, two actions need to be performed based on the initial rule to be derived in the assembly of the SPARQL query (like in Table 12.2):

1. when the placeholder is related to a literal (the one that make up the SPARQL variable "?p___X"), the property prior to it is picked up;

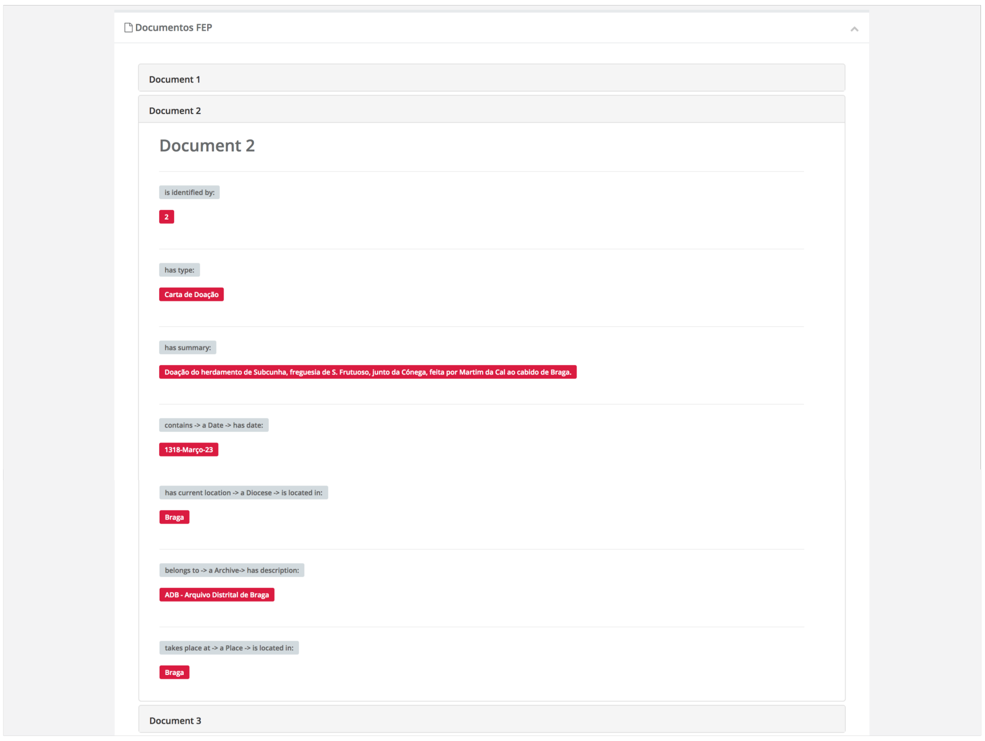
2. when the placeholder is associated to a new mapping rule, the object property and all the relations until reaching the literal placeholder (as described in step 1) are taken.

Listing 13.4 presents a small part of the ontology file, exemplifying the excerpt from where the `rdfs:label` is extracted.

Listing 13.4: Excerpt showing the `rdfs:label` tag of *is_located_in* datatype property

```xml
<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/rgm/2018/ontoFasti#is_located_in">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/rgm/2018/ontoFasti#Place"/>
    <rdfs:label xml:lang="en">is located in</rdfs:label>
</owl:DatatypeProperty>
```

So, after the execution of the generated query (the code of Listing 13.3), the labels (taken from the OntoFasti file, like in Listing 13.4) to be shown in the exhibition room entitled “Documentos FEP: 1071-1325” can be seen in Figure 13.2. Notice that without the “list of attributes”, the `headerOfEachElement` becomes the name of the concept called in the query operator of the CaVa$^{DSL}$ specification plus the first fetched value for each subject (in this case “Document” plus the value of “CotaDocumento”).


Documentos FEP

Document 1

Document 2

Document 2

is identified by:

2

has type:

Carta de Doação

has summary:

Doação do herdamento de Subcunha, freguesia de S. Frutuoso, junto da Cónega, feita por Martim da Cal ao cabido de Braga.

contains -> a Date -> has date:

1318-Março-23

has current location -> a Diocese -> is located in:

Braga

belongs to -> a Archive-> has description:

ADB - Arquivo Distrital de Braga

takes place at -> a Place -> is located in:

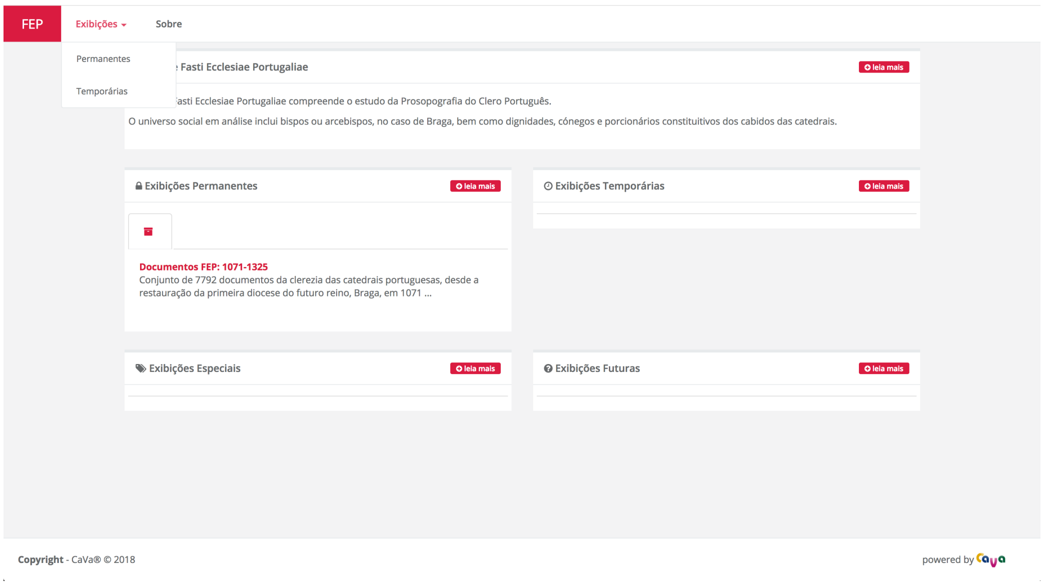
Braga

Document 3

Figure 13.2: Exhibition Room rendered with labels from `rdfs:label`

Thus, getting all the objects of the JSON file generated by the execution of the SPARQL query of Listing 13.3, the exhibition room is rendered, as shown in Figure 13.2.

To illustrate the generated static content, Figures 13.3 and 13.4 display the initial page of the virtual LS and the list of all permanent exhibitions, respectively.


FEP Exibições Sobre

Permanentes

Temporárias

Fasti Ecclesiae Portugaliae leia mais

Fasti Ecclesiae Portugaliae compreende o estudo da Prosopografia do Clero Português.

O universo social em análise inclui bispos ou arcebispos, no caso de Braga, bem como dignidades, cónegos e porcionários constituitivos dos cabidos das catedrais.

Exibições Permanentes leia mais

Documentos FEP: 1071-1325

Conjunto de 7792 documentos da clerezia das catedrais portuguesas, desde a restauração da primeira diocese do futuro reino, Braga, em 1071 ...

Exibições Temporárias leia mais

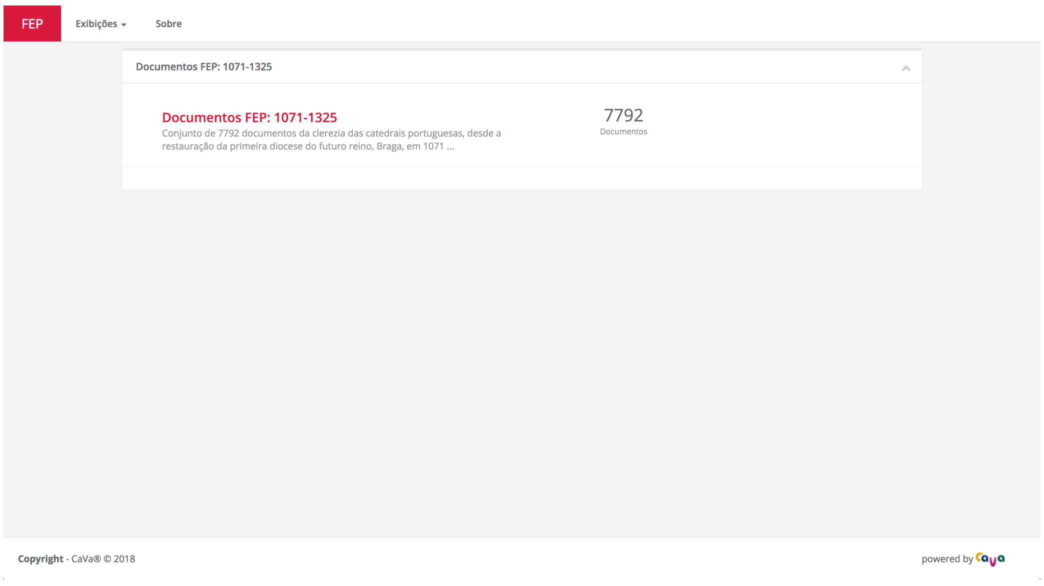
Exibições Especiais leia mais

Exibições Futuras leia mais

Copyright - CaVa® © 2018 powered by CaVa

Figure 13.3: Initial web page of FEP virtual LS

Figure 13.3 depicts the initial web page of the FEP LS, showing the menu with the “Exibições” and “Sobre” labels, the “about” text and the four types of exhibitions, listing one entitled “Documentos FEP: 1071-1325” in the permanent exhibition room section.


FEP Exibições Sobre

Documentos FEP: 1071-1325

Documentos FEP: 1071-1325

Conjunto de 7792 documentos da clerezia das catedrais portuguesas, desde a restauração da primeira diocese do futuro reino, Braga, em 1071 ...

7792 Documentos

Copyright - CaVa® © 2018 powered by CaVa

Figure 13.4: Permanent Exhibition Room list of FEP virtual LS

Figure 13.4 illustrates the list of the permanent exhibition rooms. As in this case only one exhibition was specified, just one element appears in the rendered list. This element contains a title "Documentos FEP: 1071-1325", additional info with title "7792" and description "Documentos". The short description of the exhibition room is also shown, as the behavior "expanded", which displays the UI component opened.

## 13.6 Summary

This case study was developed to reinforce the approach used in case study 1, demonstrating that CaVa architecture fits adequately to different domains. The relational database of the *Fasti Ecclesiae Portugaliae* project was used as the source for this work.

An ontology (OntoFasti) was developed, which serves to describe the bdFasti instances through abstract concepts.

The use of the mapping between the ontology and the relational database was the focus of this study, because it evidences that the CaVa system fulfills its goals based on the tested approach.

So, to automatically generate the virtual LS of the Fasti project, a specification written in CaVa$^{DSL}$ was presented and used by CaVa$^{gen}$ to create the LS scripts that together formed the final LS. As a result, some screenshots of the generated web-based LS were depicted.

Notice that this case study is smaller than the first one (Chapter 12), but it performed the same steps and achieved the desired output, i.e., the final virtual *Fasti Ecclesiae Portugaliae* Learning Space.

Finally, as it happened with case study 1, because it deals with interesting matters in the cultural and social domain, social studies similar to the ones listed in Section 12.9 can also be elaborated.

As future work, other sources, as well as other epochs related to the cathedral clergy can be studied, thus creating new exhibition rooms for the generated virtual learning environment.

In order to demonstrate the application of the CaVa architecture using an RDF database (triple store), Chapter 14 presents a virtual Learning Space for the Museum of the Person.

# Chapter 14

# Case Study 3 – Collection of life stories of the Museum of the Person

In order to build an international network of life stories, the Museum of the Person (MP) was founded in 1991 in São Paulo, Brazil[1]. The main purpose of MP is to record, preserve and disseminate the life stories of anyone, be it famous or anonymous. Thus, from storing stories, it is possible to study and contribute to different aspects of society [Almeida et al., 2001], [Simões and Almeida, 2003], [Araújo, 2016].

The original Museum of the Person is still an active and accessible project. The Brazilian version of MP can be accessed at: `http://www.museudapessoa.net`. From the original MP, the Portuguese version was conceived, focusing on knowing and reproducing life stories of Portuguese citizens. MP is located in Brazil, Portugal, USA, and Canada.

This case study is concerned with the creation of a virtual LS that exhibits life stories of common citizens. So, it is a reconstruction of the Portuguese version of the global Museum of the Person that connects individuals and

[1] Accessible at: `http://www.museudapessoa.net`

groups through sharing their life stories.

Regarding the duties of Module A – CaVa$^{settler}$ (Sections 14.1 to 14.3), this case study was done in collaboration with Cristiana Araujo in her master thesis and some publications in this area. So, those sections will present the first part of CaVa, aiming at populating the Database Repository (triple store) and describing the OntoMP ontology.

Section 14.1 presents the collection of the Museum of the Person, as well as the way it was stored in its initial format.

## 14.1 The Museum of the Person Assets

This section presents the assets of the Museum of the Person in this case study and it is based on the publication [Martini et al., 2016a] and the master thesis [Araújo, 2016].

This museum deals with common people, human beings, not with physical objects usually composing the traditional museum assets. Its "art collection" is made up of intangible or immaterial things.

According to [Almeida et al., 2001], life stories are evidences in support of facts or statements attested by common people carrying a social and historical character, which must be preserved and processed to become an immeasurable human heritage.

The alive objects are used as informers, reporting the events and emotions they experienced [Almeida et al., 2001]. Actually, the narrators, to report their life stories during a predefined structured interview, remember events and particular situations they have participated in. These memories will act as a basic element for social research, because the set of life stories allows to reconstruct a social universe [Almeida et al., 2001].

So, aiming at getting and processing the life story of each participant, the following approach is used by the Museum of the Person technicians:

- The report of a participant is recorded (audio or video) by an interviewer. Although every interview is a unique thing, interviewers guide to some predefined topics in order to cover the entire life story;
- Interviews are transcribed;
- Transcriptions are annotated in XML, marking events, self contained stories, etc;

The MP's collection is built up from interviews and it consists of XML documents related to each participant separated by projects (e.g., "Projecto da Afurada", "Memórias do Trabalho - testemunhos do Porto laboral no séc. XX", etc.). These XML interviews can be used to produce several outputs (e.g., the final virtual Museum of the Person Learning Space, etc.). An interview is divided into three parts [Simões and Almeida, 2003]:

- **mini-biography and personal data**: such as name, birthdate, place of birth, and job. A separated file (named `BI`) contains such information;
- **two versions of the interview**: the interview original text, and the edited document.
  - The `interview` file refers to the raw interview; it contains all the questions asked and the narrator's answers;
  - The `edited` file is a plain text, structured by themes that define small portions of a person's life story. In this format, a life history may give rise to thematic stories (e.g., dating, childhood, craft, among others).

  Both `interview` and `edited` files contain metadata, tagging to define important testimony zones. Examples of these marks are given, like institution names, jobs, places, etc;

- **photographs and their subtitles**: the subtitle document contains a section for each photo or scanned document as a file name and a caption. This caption includes a description of the image and the date, and when possible, the name of the stakeholder.

In addition to the interviews, there is a *thesaurus*, which includes key concepts mentioned in the stories.

The edited interview is in XML format, according to a Document Type Definition (DTD) defined specially for this purpose. This DTD consists of: identification of the deponent, episode, ancestry, descent, childhood, house, education, tradition, religion, quotidian, migration, place, dating, marriage, office, life's philosophy, event and photograph, among others. To exemplify, the event (evento) element is shown in Listing 14.1.

Listing 14.1: DTD schema to describe an Event (Evento)

```
<!ELEMENT evento (%texto ;)>
<!ATTLIST evento
ano CDATA #REQUIRED
mes CDATA #IMPLIED
dia CDATA #IMPLIED
local CDATA #IMPLIED
eixo CDATA #IMPLIED
titulo CDATA #IMPLIED
descricao CDATA #IMPLIED
relevancia (Alta | Media | Baixa) ''Media''
>
```

The element `evento` is defined as `%texto`, an XML Entity. The attributes of the `evento` element are: year `(ano)`, month `(mes)`, day `(dia)`, place `(local)`, type `(eixo)`, title `(titulo)`, description `(descricao)`, and relevance `(relevancia)`.

To clarify how the assets of the Museum of the Person are stored, an example based on the master thesis [Araújo, 2016] is given. The example presented in Listing 14.2 is the `BI.xml` file about the Maria Cacheira's life story, which contains her mini-biography (personal data, birthplace, birthdate, job, etc).

Listing 14.2: Maria Cacheira's BI

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<bi>
    <projecto>Projecto da Afurada</projecto>
    <depoente>Maria Alice Rodrigues Cacheira</depoente>
    <entrevistador>Ana Pereira; Guilherme Soares e Maria Jose da Cunha</entrevistador>
    <biografia>Maria Alice Rodrigues Cacheira nasceu na Afurada, a 8 de Outubro de 1946. Andou na escola ate a
        <habilitacoes quem="Maria Cacheira" nivel="EB">
            <escolaridade>quarta classe</escolaridade>
        </habilitacoes>.
        Perdeu a mae aos 11 anos e abandonou os estudos. Como era a filha mais velha, passou a infancia a cuidar
        dos tres irmaos e a ajudar o pai, que era pescador, a vender o peixe. Passados 18 meses o pai casou com
        uma tia e dessa relacao nasceram sete filhos. Casou aos 21 anos com um ajudante de motorista de barco.
        Apos seis anos de casamento o marido morreu com leucemia deixando tres filhas ainda pequenas. Comecou
        a trabalhar num matadouro e, posteriormente, a vender peixe. Tem seis netos e trabalha na
        <local de="Local de trabalho" quem="Maria Cacheira">junta de freguesia de Sao Pedro da Afurada.</local>
        <local de="Local de trabalho" quem="Maria Cacheira">Ao sabado ainda vende peixe no Arrabida.</local>
    </biografia>
    <data ano="2000" mes="12" dia="12"/>
    <profissao>peixeira; empregada de limpeza</profissao>
    <nascimento onde="Afurada" ano="1946" mes="10" dia="08"/>
</bi>
```

The BI element can be composed of: the project and the deponent names, the interviewer(s), photo, date, job, address, biography and birth, among others.

Thus, supported by the thesaurus and some extra concepts, the OntoMP ontology was designed based on three well-defined ontologies (CIDOC-CRM, FOAF, and DBpedia), aiming at describing the MP assets. Section 14.2 presents the final ontology for this case study domain.

## 14.2 OntoMP, an ontology for the Museum of the Person

This section presents the final OntoMP ontology designed with the purpose of describing the MP assets, as already outlined in Section 14.1. This work is based on [Martini et al., 2016a] and the master thesis [Araújo, 2016].

The OntoMP ontology for the Museum of the Person was built following the steps below:

- The first step was an exhaustive extraction of the concepts present in the life stories. After a long analysis phase the following list came up:

people (pessoa), ancestry (ascendência), offspring(descendência), house (casa), job (profissão), education (educação), episode (episódio), dating (namoro), accident (acidente), migration (migração), festivity (festividade), political event (evento político), catastrophic event (evento catastrófico), marriage (casamento), birth (nascimento), dream (sonho), childhood (infância), uses (costumes), quotidian (quotidiano), leisure (lazer), religion (religião), life's philosophy (filosofia de vida).

- In this phase relations were also identified:

  performs (exerce), depicted (éRetratada), visits (visita), lives (vive), receives (recebe), tells (narra), has (tem), enrolls (participa), has-type (tipo), occurs (ocorre), refers to (dizRespeito).

- After the extraction phase it was built an ontology for the MP using the concepts and relations above. Then it was realized that some more concepts could still be added to make a more complete ontology. The newly added concepts were:

  marital status (estadoCívil), sex (sexo), literacy (habilitações literárias), political party (partido político), first communion (primeira comunhão), death (morte), baptism (batismo), and photos (fotos).

The first sketch of the OntoMP ontology is shown in Figure 14.1, presenting an instance of OntoMP for Maria Cacheira life story.

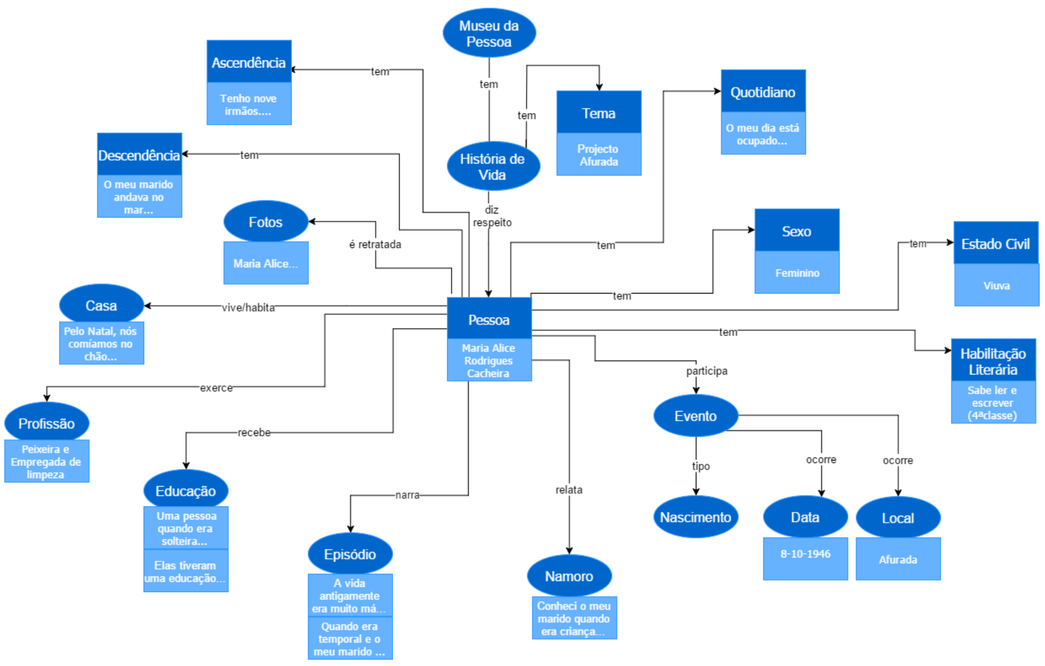


Figure 14.1: An instance of OntoMP for Maria Cacheira life story (excerpt) [Martini et al., 2016a] [Araújo, 2016]

As depicted in Figure 14.1, OntoMP has the ability to break down the raw story into logically related elements. In this way, the LS visitor can conceptually navigate over the collection [Martini et al., 2016a] [Araújo, 2016].

After the OntoMP first sketch, a new version based on the CIDOC-CRM ontology came out, but this version had some lacks in the personal relationships. Then in [Araújo, 2016], OntoMP was improved, handling with the properties relating people in a more natural form than using only CIDOC-CRM, which is not appropriated to deal with this kind of situation. So, since FOAF and DBpedia contain a vocabulary specific to describe personal relations and personal activities, they were used to extend the CIDOC-CRM version of OntoMP [Allemang and Hendler, 2011].

Concerning DBpedia, the properties religion, education, profession, spouse, and party were used. The applied properties related to FOAF were gender, name, givenName, familyName, nick, depicts, and depiction [Araújo, 2016].

The final version of OntoMP with CIDOC-CRM, FOAF, and DBpedia was instantiated with real data obtained from the document life stories. Once again, the example for the Maria Cacheira's interview is shown. Figure 14.2 presents the same instantiation excerpt of Figure 14.1, but now with the final version of OntoMP. For the sake of simplicity, notice that the examples were extracted from [Araújo, 2016].

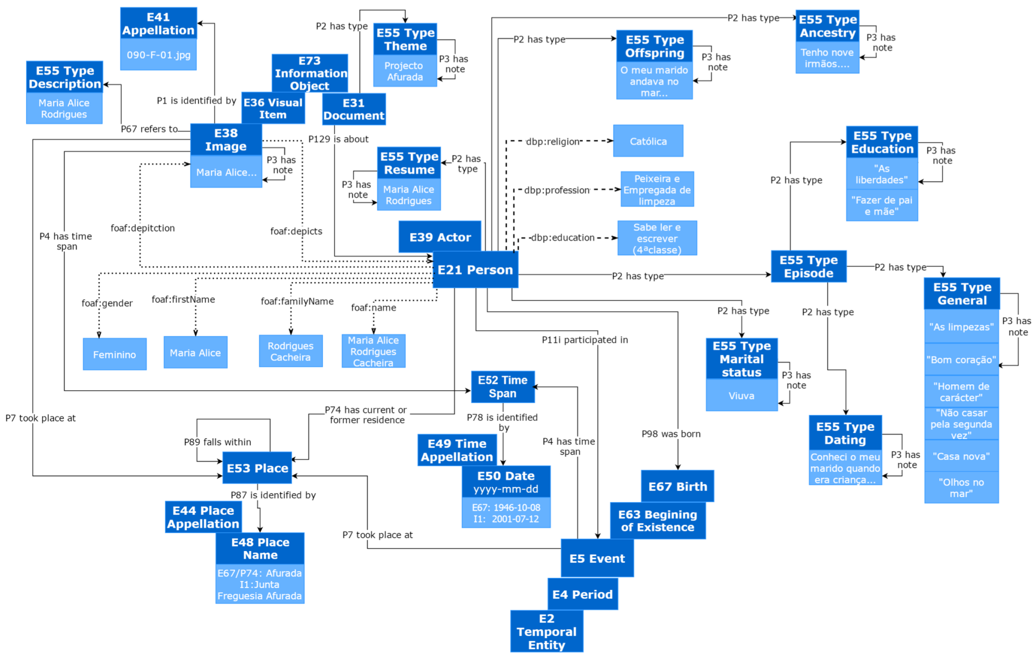


Figure 14.2: An instance of OntoMP for Maria Cacheira life story (excerpt) based on CIDOC-CRM / FOAF / DBpedia [Araújo, 2016]

The fragment of Figure 14.2 describes a CIDOC-CRM *E21 Person* concept, who has some FOAF datatype properties (dotted lines) like *gender* "Feminino", *firstName* "Maria Alice", *familyName* "Rodrigues Cacheira", and *name* "Maria Alice Rodrigues Cacheira". As object properties, this fragment shows *depicts*, having as domain, the CIDOC-CRM *E38 Image* concept and range, the CIDOC-CRM *E21 Person* (the inverse object property is *depiction*).

In addition to the FOAF properties, the DBpedia datatype properties (dashed lines) are *education* "Sabe ler e escrever ($4^a$ classe)", *profession* "Peixeira e Empregada de limpeza", and *religion* "Católica".

The enhancement of the OntoMP ontology with CIDOC-CRM, FOAF, and DBpedia properties describe the MP knowledge repository in a proper way. The full diagrams for OntoMP (abstract ontology and instances) and its CIDOC-CRM version are available at `www.di.uminho.pt/~gepl/OntoMP`. As result of the work of Cristiana Araújo in her master thesis, the virtual Museum of the Person and the OntoMP full versions can be accessed at `http://npmp.epl.di.uminho.pt/`.

After having the OntoMP schema defined, Section 14.3 lays out the XML2RDF translator, in order to populate the OntoMP ontology with the Museum of the Person assets.

## 14.3 Data Extraction and Ontology Population (XML2RDF)

To deal with the population of the MP Database Repository, a translator called XML2RDF was designed and implemented. This section will be described based on the publication [Araújo et al., 2017].

The XML2RDF translator plays the role of a DIS in the CaVa system, with the goal of populating the Database Repository, that in this case is Apache Jena TDB[2], a component of Jena for RDF storage and query via datasets. To allow persistent storage and use of SPARQL protocols for querying and updating the datasets, the Apache Jena Fuseki[3] server was used, which is tightly integrated with TDB.

So, the main objective of XML2RDF is to get the input, that is a structured collection of XML documents (in the edited format) and generate an output, that is a sequence of <subject, predicate and object> RDF triples to be stored in a TDB dataset (named "MPD", which stands for Museu da Pessoa Dataset). The concepts are related to the data items that are the value

[2] Accessible at: `http://jena.apache.org/documentation/tdb/`
[3] Accessible at: `http://jena.apache.org/documentation/fuseki2/`

of the attributes of an XML tag or the tag content. The predicate that links the concepts can be taken from the XML tags and their structure. So, XML2RDF needs to identify those data items in the given input, to extract their values and print them in the RDF output file. A similar task is needed to extract and print out the relations [Araújo, 2016] [Araújo et al., 2017].

To keep this section simple, since this work was already published and can be accessed in the publications referred above, an example of the input and the generated output for a catastrophic event is shown in Listings 14.3 and 14.4, respectively.

Listing 14.3: An XML input document – Catastrophic Event

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<eventoCatastrofico tipo="catastrofico" subtipo="cheias" dequem="Douro" data="1962-01-02" onde="Afurada">
    <p><rel tipo="where">rio Douro</rel>O ponto mais alto atingido pela agua foi precisamente no dia
        2 de Janeiro de 1962. Nos so começamos a trabalhar aqui no dia 22, mas a casa passou para nos
        no dia 2. No dia do aniversario foi quando as cheias atingiram o ponto mais alto. Isto tambem
        era antiquado, era um tasco antiquado...
    </p>
</eventoCatastrofico>
```

Listing 14.4: An RDF output document (excerpt) – Catastrophic Event – *mp.rdf*

```
<rdf:Description rdf:about="&ecrm;E9">
    <rdf:type rdf:resource="&ecrm;E5_Event"/>
    <P2_has_type rdf:resource="&ecrm;Catastrophic"/>
    <P4_has_time-span rdf:resource="&ecrm;TS29"/>
    <P7_took_place_at rdf:resource="&ecrm;PL4"/>
    <P3_has_note rdf:datatype="&xsd;string">O ponto mais alto atingido pela agua foi precisamente no dia
        2 de Janeiro de 1962. Nos so começamos a trabalhar aqui no dia 22, mas a casa passou para nos
        no dia 2. No dia do aniversario foi quando as cheias atingiram o ponto mais alto. Isto tambem
        era antiquado, era um tasco antiquado...
    </P3_has_note>
</rdf:Description>

<rdf:Description rdf:about="&ecrm;TS29">
    <rdf:type rdf:resource="&ecrm;E52_Time-Span"/>
    <P78_is_identified_by rdf:resource="&ecrm;1962-01-02"/>
</rdf:Description>

<rdf:Description rdf:about="&ecrm;PL4">
    <rdf:type rdf:resource="&ecrm;E48_Place_Name"/>
    <P3_has_note rdf:datatype="&xsd;string">Afurada</P3_has_note>
</rdf:Description>
```

So, getting the XML code of Listing 14.3 as input for the XML2RDF translator, the generated output (*mp.rdf*) is a set of RDF triples, like those shown in Listing 14.4. Notice that for this example, only CIDOC-CRM (denoted by the “&ecrm” prefix) properties were presented, but FOAF and DBpedia properties are also generated and described in this document.

Translating all the Museum of the Person XML collection, the outcome will be several RDF triples, configuring the complete Database Repository (triple store). The populated triple store database is then accessed to extract the information about the MP to be displayed in the final virtual Learning Space generated by CaVa$^{gen}$. Section 14.4 presents a specification of a virtual Learning Space about the Museum of the Person in CaVa$^{DSL}$.

## 14.4 A CaVa$^{DSL}$ specification for the Museum of the Person

To specify a virtual Learning Space in CaVa, it is necessary to have all requirements fulfilled. So, after defining Module A – CaVa$^{settler}$ (the triple store database) and the OntoMP ontology, it is time to define the Museum of the Person virtual LS according to CaVa$^{DSL}$.

Thus, this section presents a CaVa$^{DSL}$ specification named *mp.cava*. Once again, for reasons of organization and simplicity of the specification, it will be presented divided into four blocks.

**The main configuration (*mainconfig* element):**

Listing 14.5: *mainconfig* element

```
mainconfig [
    LS title:  “Museu da Pessoa”,
    about [
        p:  “O Museu da Pessoa nasceu em São Paulo, Brasil, em 1991,
            criado por um grupo de historiadores que decidiram construir a história
            do país usando depoimentos de pessoas comuns.  Este ainda é um projeto
            existente e acessível em http://www.museudapessoa.net.
            O Museu da Pessoa objetiva reunir os depoimentos de cada ser humano,
            famoso ou anônimo, para perpetuar sua história.  ”,
        p:  “A partir da história de vida dos indivíduos, o objetivo
            é escrever as histórias de família, comunidades ou até instituições.  Este museu
            lida com pessoas comuns, seres humanos, e não com objetos físicos, os quais
            normalmente compõem os espólios de museus.  A ‘‘coleção de artes’’ é feita de
            coisas imateriais ou intangíveis.  ”,
        p:  “Nesta versão portuguesa do Museu da Pessoa,
            bem como nas outras versões, os ‘‘objetos’’ da coleção (pessoas) são usadas
            como informantes, reportando os eventos e emoções que eles tiveram como experiência.
            Atualmente, os narradores relatam suas histórias de vida durante uma
            entrevista estruturada (gravada digitalmente).  Nessa entrevista, os entrevistados
            devem lembrar de eventos e outras situações particulares que participaram.
            Essas memórias atuarão como um elemento básico para a pesquisa social,
            pois o conjunto de histórias de vida permitem reconstruir um universo social.  ”,
    ]
    carousel [
        interval:  5,
        images [
            caption:  “Museu da Pessoa”, src:  “mp-capa.png”, active,
            caption:  “Projecto do Comércio Tradicional da Ribeira, Porto”, src:  “001-F-06.jpg”,
        ]
    ]
]
```

The specification of Listing 14.5 defines the title “Museu da Pessoa” and three paragraphs about the Museum of the Person virtual LS, which will be presented in the website homepage. Also, a carousel of images was specified with a interval of 5 seconds for the transition between the images. The “active” value determines the initial image when the page is loaded.

The next block of this specification is related to the header of the virtual LS (*the menu element*).

### The header (*menu* element):

Listing 14.6: *menu* element

```
menu [
    brand:  “Museu da Pessoa”,
    background color:  purple,
    foreground color:  white,
    behavior:  static,
    options [
        label:  “Exibições”, dropdown [
            dropdown label:  “Todas”, url:  “exhibitions”,
            dropdown label:  “Permanentes”, url:  “permanent_exhibitions”,
            dropdown label:  “Temporárias”, url:  “temporary_exhibitions”,
            dropdown label:  “Especiais”, url:  “especial_exhibitions”,
            dropdown label:  “Futuras”, url:  “future_exhibitions”,
        ]
        label:  “Sobre”, url:  “about”, extension:  php,
    ]
]
```

The *menu* specification presented in Listing 14.6 defines the header of the whole virtual LS. This element specifies the brand “Museu da Pessoa”, the background color “purple” and the foreground color “white”, the behavior "static", which means that the menu will remain at the top of the screen and not scrolling with it, and the menus (five dropdown and one simple labeled "Sobre").

Next, the content block is specified, showing the list of exhibitions containing two exhibition rooms related to the Museum of the Person virtual LS.

### The content (*exhibitions* element):

Listing 14.7: *exhibitions* element

```
exhibitions [
    exhibition [
        title:  "Projectos e Histórias de Vida",
        short description:  "Todas as pessoas entrevistadas e suas interessantes histórias de vida.",
        icon:  "users",
        additional info [
            title:  "13",
            description:  "Pessoas",
        ]
        behavior:  expanded,
        type:  permanent,
        SPARQL [
                PREFIX : <http://erlangen-crm.org/150929/>
                PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
                PREFIX dbp:  <http://dbpedia.org/ontology/>
                PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>

                SELECT DISTINCT ?name ?projecto ?gender ?job ?religion ?party ?education
                ?resFREGUESIA ?resCONCELHO ?resDISTRITO ?birthFREGUESIA
                ?birthCONCELHO ?birthDISTRITO ?birthTime ?nameConj
                WHERE {
                    ?pessoa a :E21_Person;
                    :P129_is_subject_of ?doc .

                    ?doc :P2_has_type ?theme .

                    ?theme :P3_has_note ?projecto .

                    ?pessoa  foaf:name ?name ;
                    foaf:gender ?gender ;
                    dbp:profession ?job ;
                    dbp:religion ?religion ;
                    dbp:party ?party ;
                    dbp:education ?education ;
                    :P74_has_current_or_former_residence ?res1 .

                    ?res1 :P87_is_identified_by ?residencia1 .
                    ?residencia1 :P3_has_note ?resFREGUESIA .
                    ?res1 :P89_falls_within ?res2 .
                    ?res2 :P87_is_identified_by ?residencia2 .
                    ?residencia2 :P3_has_note ?resCONCELHO .
                    ?res2 :P89_falls_within ?res3 .
                    ?res3 :P87_is_identified_by ?residencia3 .
                    ?residencia3 :P3_has_note ?resDISTRITO .

                    ?pessoa :P98i_was_born ?birth .
                    ?birth :P7_took_place_at ?place1 .
                    ?place1 :P87_is_identified_by ?freguesia .
                    ?freguesia :P3_has_note ?birthFREGUESIA .
                    ?place1 :P89_falls_within ?place2 .
                    ?place2 :P87_is_identified_by ?concelho .
                    ?concelho :P3_has_note ?birthCONCELHO .
                    ?place2 :P89_falls_within ?place3 .
                    ?place3 :P87_is_identified_by ?distrito .
                    ?distrito :P3_has_note ?birthDISTRITO .
                    ?birth :P4_has_time-span ?ts .
                    ?ts :P78_is_identified_by ?birthTime .

                    ?pessoa dbp:spouse ?pess2 .
                    ?pess2 foaf:firstName ?nameConj .
                }
        ][headerOfEachElement: “Entrevistado”, “Projecto”, “Sexo”, “Trabalho”, “Religião”,
        “Partido Político”, “Educação”, “Residência - Freguesia”, “Residência - Concelho”,
        “Residência - Distrito”, “Nascimento - Freguesia”, “Nascimento - Concelho”,
        “Nascimento - Distrito”, “Data de Nascimento”, “Nome do Cônjuge”]
    ]

    exhibition [
        title: “Pessoas e seus Eventos”,
        short description: “Esta sala de exibição apresenta os entrevistados
                            e suas histórias relacionadas a eventos em que
                            participaram, como casamento, eventos catastróficos, etc.”,

        icon: “calendar”,
        additional info [
            title: “5”,
            description: “Tipos de Eventos”,
        ]
        behavior: expanded,
        type: temporary,
        available until: “2018-09-01”,
        SPARQL [
                PREFIX : <http://erlangen-crm.org/150929/>
                PREFIX foaf: <http://xmlns.com/foaf/0.1/>
                PREFIX dbp: <http://dbpedia.org/ontology/>
                PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

                SELECT DISTINCT ?name ?tipoEvent ?descricao
                WHERE {
                    ?pessoa a :E21_Person;
                    :P129_is_subject_of ?doc;
                    foaf:name ?name;
                    :P11_participated_in ?ev .

                    ?ev :P2_has_type ?event ;
                    :P3_has_note ?descricao .

                    ?event :P3_has_note ?tipoEvent .
                } ORDER BY ?name
        ][headerOfEachElement: “Entrevistado”, “Tipo de Evento”, “Descrição do Evento”]
    ]
    # other exhibitions . . .
]
```

The *exhibitions* list specified in Listing 14.7 shows the definition of two exhibition rooms. The first one is about the Projects and life stories of the respondents. The exhibition “Projectos e Histórias de Vida” contains, besides the title, a short description, an icon “users”, additional info composed of a title “13” and description “Pessoas”. This first exhibition still encloses a behavior “expanded” and a type “permanent”. In addition to these static information, the exhibition “Projectos e Histórias de Vida” includes a SPARQL query, which is selecting the interviewed name, gender, job, religion, party, education, residence, birthplace and birthdate, spouse name, and the project in which he/she is allocated. A list of labels is also specified, as seen from line 63 to 66.

The second specified exhibition room is entitled “Pessoas e seus Eventos” (People and their events). This exhibition contains a short description, an icon “calendar”, additional info consisting of a title “5” and a description “Tipos de Eventos”. In addition to these attributes, a behavior “expanded” and a type “temporary” were set. As the exhibition room has type “temporary” defined, it is needed to set the date by which the exhibition will be available, that in this case is “2018-09-01”. Aiming at reaching the instances about the interviewed and their events, a SPARQL query was written. This query selects the interviewed name and his/her events (type and description). For presentation, a list of labels is also specified, as seen at line 102.

Notice that the prefixes for both SPARQL statements in the exhibition rooms are about the CIDOC-CRM (erlangen[4] implementation), FOAF, and DBpedia ontologies, as well as RDF and XML Schema Definition (XSD).

### The footer (*footer* element):

Listing 14.8: *footer* element

```
footer [
    images [
        image:  "cava_logo.png",
        alignment:  right,
    ]
    format date:  "Y",
    developer [
        name:  "Ricardo G. Martini",
        alignment:  left,
    ]
    behavior:  fixed,
    style:  extended [
        title:  "Redes Sociais", subtitle:  "visite também"[
```

[4] Available at: http://erlangen-crm.org/

```
            label:  “facebook/museudapessoa”,
            link:  “https://www.facebook.com/museudapessoa/”,
            icon:  “facebook”, icon color:  blue,
        ]
        title:  “Parceiros”, subtitle:  “visite”[
            label:  “Núcleo Português do Museu da Pessoa”,
            link:  “http://npmp.epl.di.uminho.pt/”,
            icon:  “institution”, icon color:  purple,
        ]
        title:  “Museu da Pessoa”,
            subtitle:  “versões de outros países”[
            label:  “Museu da Pessoa (Brasil)”,
            link:  “http://www.museudapessoa.net”,
            icon:  “institution”, icon color:  purple,
        ]
    ]
]
```

The *footer* specification presented in Listing 14.8 specifies the footer of the entire virtual LS. It contains images (as the CaVa logo), alignments for the images and the developer attribute, a behavior “fixed” (which means the footer will scroll with the screen), and the style as “extended”, meaning that the footer can have some extra options consisting of title, subtitle, label, link, icon, and icon color. This is useful to provide additional information about partners, social networks, etc.

The processing of the *mp.cava* specification (all four blocks in the same *.cava* file) is a duty of the CaVa$^{\text{gen}}$ processor. So, Section 14.5 details the steps from the recognition of *mp.cava* to the generation of the scripts that shall be interpreted by the web browser in the CaVa$^{\text{render}}$ module.

## 14.5 CaVa$^{\text{gen}}$ applied to the automatic generation of the Museum of the Person virtual LS

Regarding the processing of the *mp.cava* specification file to generate the final virtual “Museu da Pessoa” Learning Space, CaVa$^{\text{gen}}$ is applied, generating the static and dynamic content.

The process of generating static content is a task of the CaVa$^{\text{structure}}$ processor and it is very similar to the other two case studies (Chapters 12 and 13) already presented. The only difference is the context, i.e., the values to be generated, but the structure is the same. Because of that, this case study focuses on the processing and generation of the dynamic content, describing the work done about the CaVa$^{run}$ processor, which deals with the SPARQL queries written in Listing 14.7 to reach the right instances of the "MPD" triple store dataset. Note that the CaVa$^{queriesTriple}$ processor is not triggered for the assembly of SPARQL queries, but it is called to verify if the query is well formed according to the SPARQL grammar rules.

The CaVa$^{structure}$ blueprint of Figure 10.3 was realized in this case study by the subsequent implementation:

- CaVa$^{DSL}$ Specification $\mapsto$ *mp.cava* file;
- CaVa$^{grammar}$ Processor $\mapsto$ ANTLR;
- CaVa State file $\mapsto$ plain text (*.txt*);

Note that CaVa$^{grammar}$, the CFG presented in Appendix A.1 is always the same and is not instantiated in the project.

To exemplify the static content generation, based on the same listeners as shown in Listing 12.11, the static LS script (*.php*) for the *mainconfig* element of CaVa$^{DSL}$ for this case study is generated and shown in Listing 14.9.

Listing 14.9: Generated PHP code for creating the mainconfig element according to *mp.cava*

```php
<?php
    $data = array(
        'lsTitle'=>"Museu da Pessoa",
        'about'=>array(
            array('p'=>"O Museu da Pessoa nasceu em São Paulo ..."),
            array('p'=>"A partir da história de vida dos indivíduos ..."),
            array('p'=>"Nesta versão portuguesa do Museu da Pessoa ..."),
        ),
        'carousel'=>array(
            'active'=>"true",
            'interval'=>5000,
            'images'=>array(
                array('caption'=>"Museu da Pessoa", 'src'=>"mp-capa.png", 'active'=>"true"),
                array('caption'=>"Projecto do Comércio Tradicional da Ribeira, Porto",
                      'src'=>"001-F-06.jpg", 'active'=>"false"),
            ),
        ),
    //continues with the exhibitions list element PHP code ...
```

Notice that the only differences from Listing 12.12 to the code of Listing 14.9 are the values on the right side of the 'key-value' pairs, i.e., the value of each attribute.

At this point, $\mathsf{CaVa}^{\text{structure}}$ recognizes the token "SPARQL" in the exhibition room specification (*mp.cava*) and generates the exhibition rooms (client- and server-side files). To exemplify the generated files, the specified exhibition 2 is taken. The client- and server-side generated files can be seen in Listings 14.10 and 14.11, respectively.

Listing 14.10: Automatically generated template code for an exhibition room – "*exhibition2.tpl*"

```
{assign var=accordionID value=1}
{assign var=accordionName value=accordion}
{assign var=countID value=1}
{assign var=indexLabels value=0}
<div class="row">
    <div class="container">
        <div class="col-lg-12">
            <div class="ibox float-e-margins {if $data.collapsed eq 'collapsed'}collapsed{/if}">
                <div class="ibox-title">
                    <h5><i class="fa fa-file-o"></i> Pessoas e seus Eventos</h5>
                    <div class="ibox-tools">
                        <a class="collapse-link">
                            <i class="fa fa-chevron-up"></i>
                        </a>
                    </div>
                </div>
                <div class="ibox-content">
                    <div class="panel-body">
                        <div class="panel-group" id="accordion">
                            {foreach from=$data item=i key=k}
                            {if $k neq "collapsed" and $k neq "labels"}
                            <div class="panel panel-default">
                                <div class="panel-heading">
                                    <h5 class="panel-title">
                                        <a data-toggle="collapse" data-parent="#accordion"
                                           href="#{$accordionName}{$accordionID}">{$i['0']}
                                        </a>
                                    </h5>
                                </div>
                                <div id="{$accordionName}{$accordionID}" class="panel-collapse collapse">
                                    {assign var=accordionID value=$accordionID+1}
                                    <div class="panel-body">
                                        <div class="col-md-12">
                                            <div class="row">
                                                <div class="col-md-9">
                                                    <h2 class="font-bold m-b-xs">{$i['0']}</h2>
                                                </div>
                                            </div>
                                            {foreach from=$i item=j}
                                                {if $j eq ""}
                                                    {assign var=indexLabels value=$indexLabels+1}
                                                {elseif $j neq ""}
                                                    <div class="row col-md-12">
                                                        <hr>
                                                    </div>
                                                    <div class="row col-md-9">
```

```
                                                        <div class="label">{$data.labels[$indexLabels]}:</div>
                                                        {assign var=indexLabels value=$indexLabels+1}
                                                        <dl class="small m-t-md">
                                                            <dt class="label navy-bg">{$j}</dt>
                                                        </dl>
                                                    </div>
                                                {/if}
                                            {/foreach}
                                            {assign var=indexLabels value=0}
                                        </div>
                                    </div>
                                </div>
                            </div>
                            {/if}
                            {/foreach}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

Listing 14.11: Automatically generated PHP code for an exhibition room – "*exhibition2.php*"

```
1   <?php
2       $sparqlQuery = "query2.rq";
3       $sparqlResult = "query2.json";
4       $mappingOrTriplesFile = "mp.rdf";
5       $IRIOntology = "http://semanticweb.org/rgm/2018/ontoMP/";
6       $jarFilePath = ".../rdf2jsonJena.jar ";
7       shell_exec($jarFilePath . " " . $sparqlQuery . " " . $sparqlResult
8                       . " " . $mappingOrTriplesFile . " " . $IRIOntology);
9       $json = file_get_contents($sparqlResult);
10      $data = json_decode($json, TRUE);
11      $data ['labels'] = array(
12          0 => "Entrevistado",
13          1 => "Tipo de Evento",
14          2 => "Descrição do Evento",
15      );
16      $data ['collapsed'] = "expanded";
17      $tpl = new SMTemplate();
18      $tpl->render('exhibition2', $data);
```

When an exhibition room (Listing 14.11) is accessed, the CaVa$^{run}$ processor receives the content of "SPARQL" as a parameter (stored in a *.rq* file, as "query2.rq" at line 2). CaVa$^{run}$ (in this case the "rdf2jsonJena.jar" at line 6) will then execute (*shell_exec()* at line 7) and generate the JSON result file (as "query2.json" at line 3) with the instances found in the "MPD" dataset.

Summing it up, the generated "*exhibition2.php*" file performs six tasks:

1. receives as input:

   (a) "*query2.rq*" file as the SPARQL query;
   (b) "*query2.json*" file to save the query's result;
   (c) "*mp.rdf*" file as the RDF triples file generated by the XML2RDF translator;
   (d) OntoMP IRI ("*http://semanticweb.org/rgm/2018/OntoMP/*");
   (e) the program ("*rdf2jsonJena.jar*") that in fact realizes the execution of the "*query2.rq*" file.

2. calls the *shell_exec()* command with the received inputs;
3. gets the content of "*query2.json*", filled by "*rdf2jsonJena.jar*";
4. stores in *$data* the "*query2.json*" decoded content;
5. stores the labels for the SPARQL statement (Listing 14.7) and declares a flag which defines the UI component behavior (expanded);
6. renders the exhibition room from the *$data* variable content. The last two lines of Listing 14.11 mean that the variable *$tpl* (an instance of the *SMTemplate* class) calls the method render (already defined in the CaVa system), passing the array *$data* to render the Smarty template *exhibition2.tpl* (see Listing 14.10). The placeholders of the *exhibition2* template are filled with the *$data* array information, resulting in the rendered *exhibition2* (Figure 14.8) in Section 14.6. Every generated exhibition room "*.php*" file, in CaVa, instantiates the *SMTemplate* class, looking to render the associated template ("*.tpl*") file.

In order to execute the SPARQL query of the specified exhibition rooms with "*rdf2jsonJena.jar*", CaVa$^{\text{run}}$, sketched in Figure 10.5 for this case study, is instantiated based on the following map:

- triples Specification $\mapsto$ *mp.rdf* file;
- Repository $\mapsto$ "MPD" dataset;

- Ontology (TBox) $\mapsto$ OntoMP;
- Queries $\mapsto$ SPARQL (*.rq*);
- Query Results $\mapsto$ JSON (*.json*);
- CaVa$^{run}$ Processor $\mapsto$ Jena Fuseki - ARQ reasoner;

Figure 14.3 displays the CaVa$^{run}$ schema instantiated.

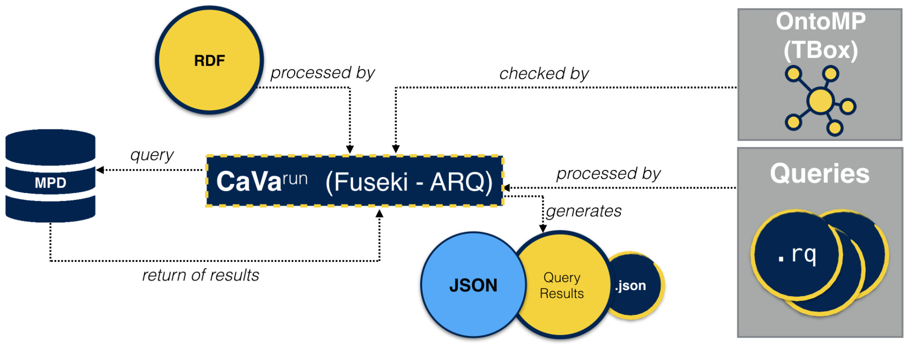


Figure 14.3: Concrete instance of the CaVa$^{run}$ processor schema for the MP virtual LS

Assuming the code of Listing 14.4 as the RDF triples (*mp.rdf*) already stored in the "MPD" dataset through the TDB command-line utilities[5] using the *tdbloader2* command, the schema of Figure 14.1 as the ontology (implemented with CIDOC-CRM, FOAF, and DBpedia, as illustrated in Figure 14.2), and the SPARQL queries of the exhibition rooms (to exemplify, the SPARQL statement of exhibition 2, as seen between the lines 83 and 102 of Listing 14.7) to be executed by the CaVa$^{run}$ processor, its execution is carried out in three steps:

1. Defines the SPARQL endpoint ("http://localhost:40000/MPD/sparql") and dataset to be queried (in this case, "MPD"), passing the query to be executed. Listing 14.12 outlines the snippet for this specific task.

[5] Available at: `https://jena.apache.org/documentation/tdb/commands.html`

Listing 14.12: Code snippet to define the SPARQL endpoint and the dataset to be queried

```java
import org.apache.jena.query.* ;
//...all the import statements necessary...

//read the content of the SPARQL query
queryString = new String(Files.readAllBytes(Paths.get("./query2.rq")));

//create the query object
Query query = QueryFactory.create(queryString);

//initialize the QueryExecutionFactory with remote service
QueryExecution queryExec = QueryExecutionFactory.sparqlService("http://localhost:40000/MPD/sparql", query);
```

2. Executes the query over the SPARQL endpoint and the "MPD" dataset, getting the result set at the "res" variable. Listing 14.13 presents the code to perform this action.

   Listing 14.13: Code snippet to execute the SPARQL query over the "MPD" dataset

```java
ResultSet res = queryExec.execSelect();
```

3. Writes and formats the result of *query2.rq* as JSON, storing it in a file (*query2.json*). Listing 14.14 shows the code snippet to execute this task.

   Listing 14.14: Code fragment to write the query results as JSON

```java
//write to a ByteArrayOutputStream
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

//format the result as JSON
ResultSetFormatter.outputAsJSON(outputStream, res);

//turn the result (res) into a String
String sJson = new String(outputStream.toByteArray());

//write the String sJson to query2.json file to be read by exhibition2.php
try {
    File jsonFile = new File("query2.json");
    FileWriter wr = new FileWriter(jsonFile, false); // false to overwrite.
    wr.write(sJson);
    wr.close();
} catch (Exception e) {e.printStackTrace();}
```

After these three tasks are performed, the exhibition room (Listing 14.11) handles the JSON result file content (fragment presented in Listing 14.15) to display the instances in the virtual Learning Space exhibition room.

Listing 14.15: Fragment of the JSON result file generated by CaVa$^{\text{run}}$– "*query2.json*"

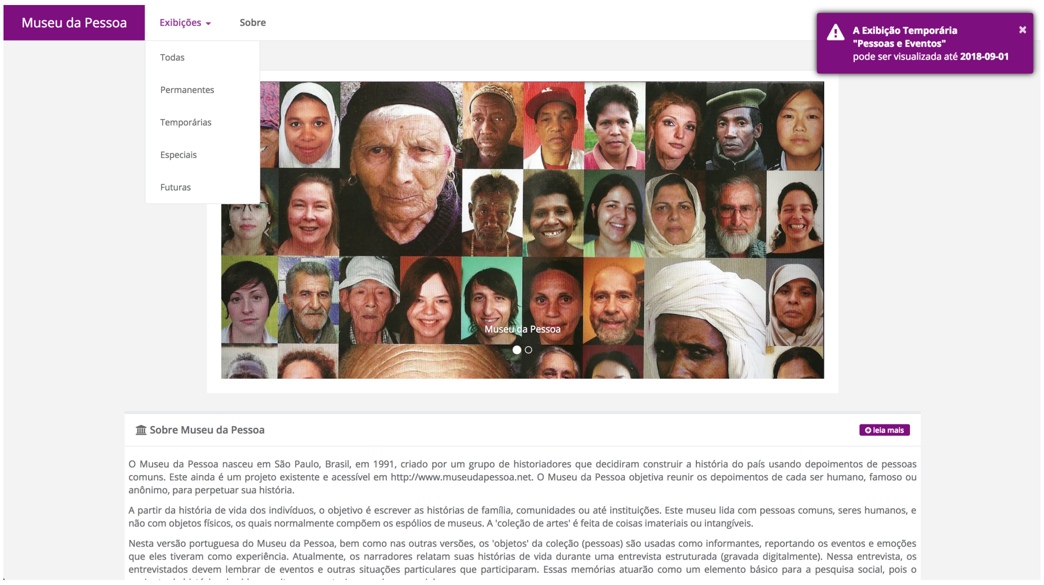
```
{
    "0": {
        "0": "Albertina da Conceição Soares",
        "1": "Wedding",
        "2": "Casei com 24 anos."
    },
    "1": {
        "0": "António Oliveira Machado",
        "1": "Catastrophic",
        "2": "O ponto mais alto atingido pela água foi precisamente
             no dia 2 de Janeiro de 1962. Nós só começamos a trabalhar
             aqui no dia 22, mas a casa passou para nós no dia 2.
             No dia do aniversário foi quando as cheias atingiram
             o ponto mais alto. Isto também era antiquado, era um tasco
             antiquado. A água partiu as portas, entre outras coisas.
             A gente dali de fora a ver as portas abrirem-se e a fecharem
             e as coisas a irem pela porta fora para o rio. Eu vim aqui
             dentro buscar uma máquina de pesar e a água dáva-me pelo pescoço.
             Mais de um metro e meio, quase dois metros."
    },
    //other JSON objects ...
    "3": {
        "0": "Caridade de Oliveira Matos",
        "1": "Widowhood",
        "2": "Fiquei viúva há 47 anos. O marido morreu tuberculoso.
             Tive 5 filhos: três meninas e dois meninos. Morreram todos.
             Com certeza o problema já vinha no sangue. Como o meu marido
             morreu tuberculoso. O meu sogro já morreu tuberculoso.
             Já morreram dois cunhados também. E isto já vinha no sangue.
             Só uma menina é que morreu com menigite, aos oito meses.
             Os outros nasciam muito gordos, muito bonitos, duravam ali
             até aos dois anos, 20 meses. Depois, lá iam embora."
    },
    //other JSON objects ...
    "6": {
        "0": "Maria Alice Rodrigues Cacheira",
        "1": "Childs Birth",
        "2": "Depois passado dois meses de casada nasceu a minha primeira
             filha, que eu já ia de bebé. Foi serviço adiantado."
    },
    //other JSON objects ...
}
```

The index "0" for each JSON Object is related to the respondent's name. The index "1" is about the event type, and the index "3" refers to the event description. So, using the generated code and results, Section 14.6 depicts some screenshots of the automatically generated Museum of the Person virtual Learning Space.

## 14.6 Rendering the final Museum of the Person virtual LS with CaVa$^{render}$

From the LS Scripts presented in Section 14.5 and generated by CaVa$^{gen}$, CaVa$^{render}$ is the module in charge for recognizing and rendering them via a web browser. This section depicts some screenshots of the rendered LS Scripts, comprising the virtual Learning Space entitled "Museu da Pessoa".

Figure 14.4 presents the initial web page when the visitor accesses the virtual LS.



A Exibição Temporária "Pessoas e Eventos" pode ser visualizada até **2018-09-01**

O Museu da Pessoa nasceu em São Paulo, Brasil, em 1991, criado por um grupo de historiadores que decidiram construir a história do país usando depoimentos de pessoas comuns. Este ainda é um projeto existente e acessível em http://www.museudapessoa.net. O Museu da Pessoa objetiva reunir os depoimentos de cada ser humano, famoso ou anônimo, para perpetuar sua história.

A partir da história de vida dos indivíduos, o objetivo é escrever as histórias de família, comunidades ou até instituições. Este museu lida com pessoas comuns, seres humanos, e não com objetos físicos, os quais normalmente compõem os espólios de museus. A 'coleção de artes' é feita de coisas imateriais ou intangíveis.

Nesta versão portuguesa do Museu da Pessoa, bem como nas outras versões, os 'objetos' da coleção (pessoas) são usadas como informantes, reportando os eventos e emoções que eles tiveram como experiência. Atualmente, os narradores relatam suas histórias de vida durante uma entrevista estruturada (gravada digitalmente). Nessa entrevista, os entrevistados devem lembrar de eventos e outras situações particulares que participaram. Essas memórias atuarão como um elemento básico para a pesquisa social, pois o

Figure 14.4: Initial web page of the Museum of the Person Learning Space

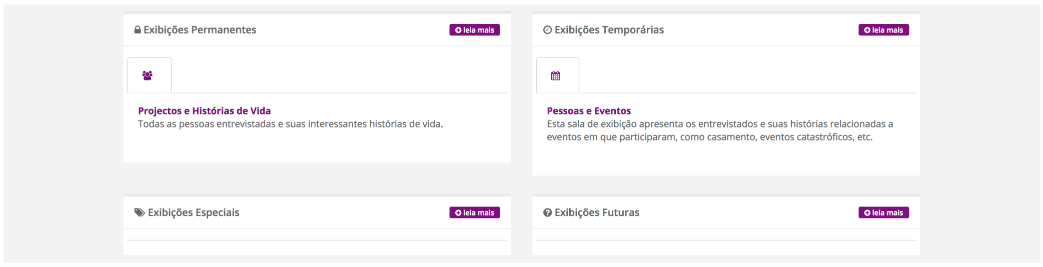
The initial page depicted in Figure 14.4 contains:

- the *mainconfig* element rendered: it includes a carousel with two images setting the first one with caption "Museu da Pessoa" as active (it means that when the visitor opens the referred web page, the active image will appear before the others) and a time (5 seconds) for the images transition. In addition, the text about the museum is presented;

- the *menu* element rendered: it contains the brand "Museu da Pessoa", one dropdown menu "Exibições" including five sub-menus ("Todas", "Permanentes", "Temporárias", "Especiais", and "Futuras"), which refer to the exhibition room types, and a simple menu "Sobre" (about the museum). In addition, the background color "purple" and the foreground color "white" can be seen;

- the *available until* attribute rendered: a brief message advertising that the temporary exhibition "Pessoas e Eventos" can be visualized until the date "2018-09-01". This date was specified in Listing 14.7, line 82. This is a resource of temporary exhibition rooms. The message appears for 7 seconds, every time that the initial web page is visited.

If the visitor of the "Museu da Pessoa" virtual Learning Space scrolls the initial page (Figure 14.4) down, he/she will see four boxes. Each box is related to an exhibition type (permanent, temporary, special, and future). Figure 14.5 depicts the exhibition boxes.


Exibições Permanentes

leia mais

Projectos e Histórias de Vida
Todas as pessoas entrevistadas e suas interessantes histórias de vida.

Exibições Temporárias

leia mais

Pessoas e Eventos
Esta sala de exibição apresenta os entrevistados e suas histórias relacionadas a eventos em que participaram, como casamento, eventos catastróficos, etc.

Exibições Especiais

leia mais

Exibições Futuras

leia mais

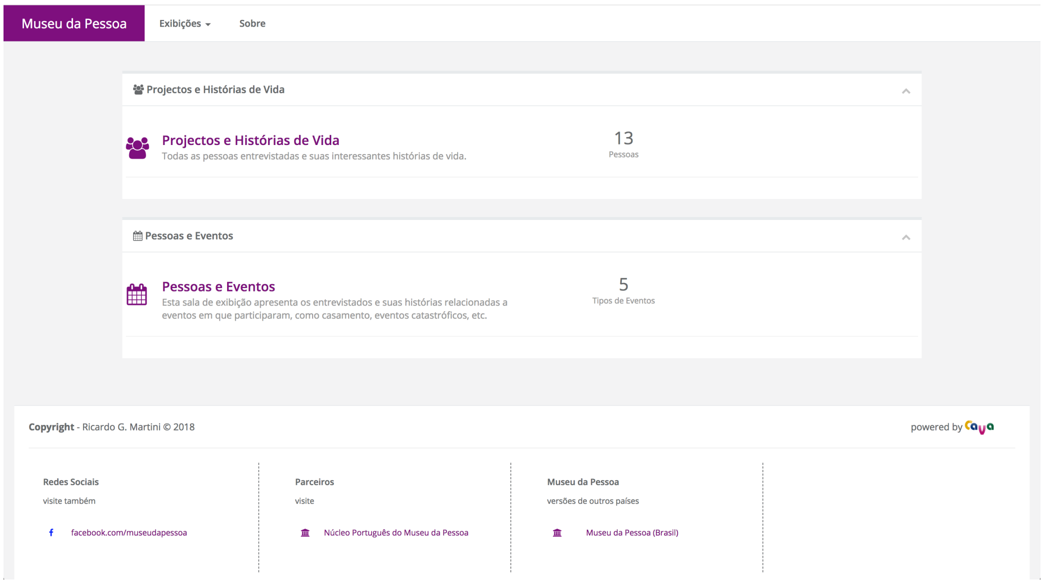
Figure 14.5: All the exhibitions in the initial web page

In this case, as only two exhibition rooms (one permanent and one temporary) were specified, the two boxes below appear empty. The box "Exibições Permanentes" shows the exhibition room "Projectos e Histórias de Vida", containing a short description, an icon "users", and a button labeled "leia mais"[6] (read more), which leads to the page that lists all the permanent exhibitions. The same occurs with the box "Exibições Temporárias" (Temporary

6 Each box contains a button to access the list of exhibitions of that specific type.

Exhibitions), that contains an exhibition room “Pessoas e Eventos”. This box has the “leia mais” button, an icon “calendar”, the exhibition title, and the exhibition short description.

Figure 14.6 shows a list of all exhibition rooms, which can be accessed through the menu “Exibições->Todas”.



Projectos e Histórias de Vida

Projectos e Histórias de Vida
Todas as pessoas entrevistadas e suas interessantes histórias de vida.
13
Pessoas

Pessoas e Eventos

Pessoas e Eventos
Esta sala de exibição apresenta os entrevistados e suas histórias relacionadas a eventos em que participaram, como casamento, eventos catastróficos, etc.
5
Tipos de Eventos

**Copyright** - Ricardo G. Martini © 2018
powered by Caya

Redes Sociais
visite também
facebook.com/museudapessoa

Parceiros
visite
Núcleo Português do Museu da Pessoa

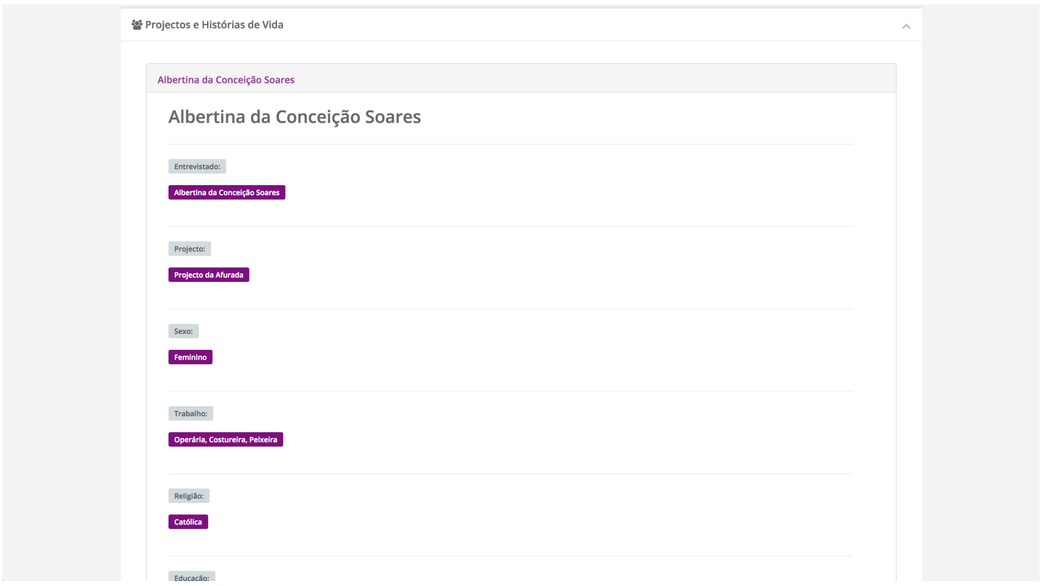
Museu da Pessoa
versões de outros países
Museu da Pessoa (Brasil)

Figure 14.6: Exhibitions list accessed via the *menu* element

Additionally to the *menu* and *footer* elements, Figure 14.6 depicts a list of the two specified exhibition rooms, containing a box for each one. The first box is related to the “Projectos e Histórias de Vida” permanent exhibition room. It contains a title, an icon “users”, a short description, and an additional info (title “13” and description “Pessoas”). In addition, the box appears “Expanded” (denoted by the chevron-up icon in the top right corner). The same occurs with the temporary exhibition room “Pessoas e Eventos” of the next box. This exhibition consists of a title, an icon “calendar”, a short description, an additional info (title “5” and description “Tipos de Eventos”), and a behavior “Expanded”. Notice that this list can be accessed via the *menu* options or through the initial page (not for all exhibitions, but for a specific type of), as already explained.

The *footer* element of Figure 14.6 is composed of images (like the CaVa logo), format date “2018”, developer name “Ricardo G. Martini”, behavior “fixed”, which means the footer does not follow the scrolling of the page. In addition, the links for social networks “Redes Sociais”, partners “Parceiros”, and for the other version of the Museum of the Person “Museu da Pessoa” are shown.

To illustrate the exhibition rooms, Figures 14.7 and 14.8 presents the content of the permanent *exhibition1* and the temporary *exhibition2*, respectively.


Projectos e Histórias de Vida

Albertina da Conceição Soares

Albertina da Conceição Soares

Entrevistado:
Albertina da Conceição Soares

Projecto:
Projecto da Afurada

Sexo:
Feminino

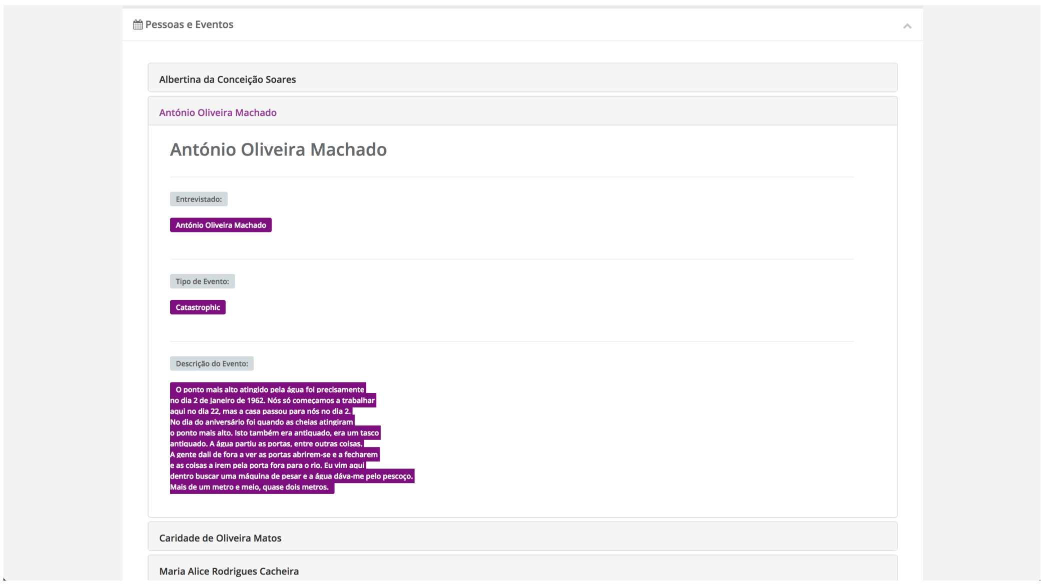
Trabalho:
Operária, Costureira, Peixeira

Religião:
Católica

Educacão:

Figure 14.7: Permanent exhibition room “Projectos e Histórias de Vida”

The exhibition room depicted in Figure 14.7 shows the instances taken from the *query1.json* file (similar to the JSON file of Listing 14.15).



Pessoas e Eventos

Albertina da Conceição Soares

António Oliveira Machado

António Oliveira Machado

Entrevistado:

António Oliveira Machado

Tipo de Evento:

Catastrophic

Descrição do Evento:

O ponto mais alto atingido pela água foi precisamente
no dia 2 de janeiro de 1962. Nós só começamos a trabalhar
aqui no dia 22, mas a casa passou para nós no dia 2.
No dia do aniversário foi quando as cheias atingiram
o ponto mais alto. Isto também era antiquado, era um tasco
antiquado. A água partiu as portas, entre outras coisas.
A gente dali de fora a ver as portas abrirem-se e a fecharem
e as coisas a irem pela porta fora para o rio. Eu vim aqui
dentro buscar uma máquina de pesar e a água dáva-me pelo pescoço.
Mais de um metro e meio, quase dois metros.

Caridade de Oliveira Matos

Maria Alice Rodrigues Cacheira

Figure 14.8: Temporary exhibition room "Pessoas e Eventos"

The exhibition room illustrated in Figure 14.8 presentes the instances taken from the *query2.json* file (see Listing 14.15).

Notice that for the two exhibition rooms, the *headerOfEachElement*, specified in Listing 14.7 lines 63 and 102, is used in the exhibitions template (denoted by "$i['0']" at line 26 of Listing 14.10) to render, in this case, the respondent name (for *exhibition1*, the name "Albertina da Conceição Soares"; for *exhibition2*, the names "Albertina da Conceição Soares", "António Oliveira Machado", "Caridade de Oliveira Matos", and "Maria Alice Rodrigues Cacheira").

Figures 14.7 and 14.8 present the result set instances achieved from the triple store database and the labels written in the CaVa$^{\text{DSL}}$ specification of Listing 14.7.

## 14.7 Summary

The main objective of this case study was to apply a different approach of storing the sources described in RDF triples in a triple store to automatically generate the Museum of the Person virtual Learning Space based on a specification written in CaVa$^{\text{DSL}}$ and ontologies.

This case study presented the development and use of the OntoMP ontology, which was conceived in the master thesis of Cristiana Araújo [Araújo, 2016].

To deal with the Museum of the Person XML assets, a translator (XML2RDF) was explained and used for this case study. This translator was also conceived in [Araújo, 2016]. The objective of XML2RDF was to transform the XML input files, which tell life stories of ordinary people, to RDF triples. To store the generated RDF triples, an Apache Jena TDB dataset (MPD) was created.

Having all the prerequisites, a CaVa$^{\text{DSL}}$ specification was written, with the purpose of describing the Museum of the Person virtual LS to be generated by the CaVa$^{\text{gen}}$ set of processors. In order to automatically generate the LS Scripts for the desired final virtual LS, the generation process lead by CaVa$^{\text{gen}}$ was explained.

Once again, as this case study deals with important documents, which tell life stories of people, aiming at writing up the stories of families, communities, or institutions (interesting social and cultural matters), social studies can also be developed.

As future work, other interviews can be recorded and transcribed to XML, generating more assets to the Museum of the Person, and consecutively, more exhibition rooms with different and attractive stories of life might be generated.

Chapter 15 concludes this thesis, shows the contributions, and gives the directions for future work. In addition, the case studies general analysis is done.

# Chapter 15

# Conclusion

The main goal of this PhD work was, through the collected information of cultural institutions, to allow their curators to describe virtual Learning Spaces, which are automatically generated.

People who are from other areas than computer science normally do not know about programming logic. In the museological area it is no different. So, it is a hard task to the Curator to build a virtual museum as a Learning Space with exhibition rooms. Because of that, the museums' person in charge commonly hire a third party service to perform this task. This is an expensive approach (it costs time and money).

So, in this work, the CaVa architecture was designed and developed to deal with this kind of problems, letting the Curator organize and exhibit the collections of the museum in exhibition rooms through a high-level specification written in CaVa$^{\text{DSL}}$, without knowledge about data persistence or programming logic. The Curator only needs to know about his duties, regarding object exhibitions.

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. This means that learning the CaVa$^{\text{DSL}}$ language is enough to specify and generate the desired virtual Learning Space.

To cope with the CaVa$^{\text{DSL}}$ specifications to finally generate the virtual Learning Spaces, a set of processors (CaVa$^{\text{gen}}$) was designed and developed.

All the necessary subjects related to the scope of this thesis were studied and presented in Part I, while the proposal and the CaVa platform blocks were introduced in Part II.

## 15.1 Revisiting objectives and results

Regarding the main goal of this PhD research, which was automating the generation of virtual Learning Spaces based on ontologies and a Domain-Specific Language, two specific objectives were formulated in Chapter 1. These objectives are revisited here and the main achieved results are highlighted.

1. **Create a formalism (DSL) to rigorously describe virtual Learning Spaces taking into account the domain ontology**

   The achievement of this objective, addressed in Chapter 9, resulted in the design and development of an external Domain-Specific Language, CaVa$^{\text{DSL}}$, which allows the specification of a virtual Learning Space based on a controlled domain ontology vocabulary. The development of CaVa$^{\text{DSL}}$ followed the design guidelines presented in Section 4.3. This CaVa$^{\text{DSL}}$ formalism was built with the purpose of fostering the CH Curator to specify a virtual Learning Space from his/her perspective.

   The use of CaVa$^{\text{DSL}}$ in real case studies allowed a comprehensive evaluation of the application of the developed language. The CaVa$^{\text{DSL}}$ specification outcome for each case study was presented in Part III (Chapters

12, 13, and 14) dealing with two different approaches concerning the method to access the instances stored in the digital repository. After applying CaVa$^{DSL}$ for three different scenarios, the analysis of the development of the language, following the criteria of [Visser, 2008], shows that it comprises:

- *Expressivity*: specifying virtual LS in the context of this work, in CaVa$^{DSL}$, is light and requires less effort compared with programming in several general-purpose languages. To measure the issue of less effort in practice, a good approach is to compare the lines of code written and generated[1]. For the developed case studies, the number of written lines were 76, 57, and 182, respectively. The number of generated lines only for the related parts of the virtual LS (main configuration, header, content, and footer) were 1100+ without counting the stylesheets and JavaScript code, which are required for the system to work properly;
- *Coverage*: CaVa$^{DSL}$ provides elements and structure to create virtual LS independent of its domain;
- *Completeness*: the generator (CaVa$^{gen}$) that handles CaVa$^{DSL}$, generates complete code. This means that there is no need to complement the generated code;
- *Portability*: CaVa$^{DSL}$ is an abstract language, not tightly related to the generated code (only with the parts that a web application must have), so the portability to other implementation platforms such as Java, Ruby on Rails, Python, etc. is guaranteed;
- *Extensibility*: CaVa$^{DSL}$ was designed to be extensible, in order to hold the conception of future elements. Extending CaVa$^{grammar}$ makes it possible to create new elements and sub-languages for CaVa$^{DSL}$.

2. **Develop a mechanism to automatically generate the virtual LS from its formal specification**

[1] Probably the generated code is not necessarily as compact as if written manually.

The fulfillment of this specific objective, discussed in Chapter 10, resulted in the development of CaVa$^{\text{gen}}$, a set consisting of four processors, which deal with the CaVa$^{\text{DSL}}$ specifications, generating the static and dynamic content, and the structure of virtual Learning Spaces. CaVa$^{\text{gen}}$ was applied for three different case studies presented in Chapters 12, 13, and 14, producing the desired outcomes.

## 15.2 Main Contributions and Thesis

Based on the initial objectives and the achieved results, this section lists the main contributions of this work:

- *Proposal of an architecture (CaVa) to automatically generate virtual Learning Spaces based on ontologies and a Domain-Specific Language* [Martini et al., 2016c];
- *Proposal of an external DSL (CaVa$^{DSL}$) to specify virtual Learning Spaces* [Martini and Henriques, 2017b];
- *Proposal of a set of processors (CaVa$^{gen}$) to deal with the CaVa$^{DSL}$ specifications* [Martini and Henriques, 2017b];
- *Design of three ontologies for the real proposed case studies domains*, respectively [Martini et al., 2016b], [Martini and Henriques, 2017b], and [Martini et al., 2016a];
- *Development of an international project in cooperation with Fafe's Municipal Archive, funded by the ADAI program of Iberarchivos, in which bdME and SGPE are included* [Martini et al., 2015] and the project website[2];
- *Development of the three real case studies scenarios to validate CaVa*, respectively [Martini et al., 2017], [Martini and Henriques, 2017b], and [Martini et al., 2018].

[2] Available at: `http://bit.ly/2p3tC2S`

These contributions support the statement that the Research Hypothesis has been proved: *it is feasible to automatically create virtual Learning Spaces, as web pages, based on an ontology — that describes an institutional information repository — and on a DSL specification — that defines which concepts should be exhibited and how they should be placed in the final virtual Learning Space.*

## 15.3 Topics of Future Work

Several topics were identified as interesting directions for future work. Some of the topics listed below are technical issues that may improve particular aspects of CaVa, the platform developed in this doctoral project. But also some ideas, at a more conceptual/abstract level, will be pointed out below.

### Technical work

- *Improvements in the UI*: creation of filters and sorting tools to prepare the returned data (query results) to display in the exhibition rooms; creation of new templates; creation of a different way to navigate over the concepts of the ontology. Maybe include a graphical picture of the ontology in the virtual LS;
- *CaVa$^{DSL}$ extension*: creation of new language elements (e.g., operators to define tables to display the queried data, maps, charts, etc); definition of new templates through CaVa$^{\text{DSL}}$; specification of more than one query by exhibition room; development of tools to aid the Curator in the specification of a virtual LS (e.g., intelligent code completion tool for the concepts and properties of the ontology, graphical tools, etc.);
- *CaVa$^{gen}$ portability*: development of new case studies targeting different languages and based on different digital repositories and new ontologies. This implies more research for new approaches for the mapping issues;

- *CaVa performance*: improve query performance with approaches like cache system, etc.

### Research work

Requiring a higher level of research, some other proposals can be listed:

- *Regarding the ontology*:
  - verify whether the CIDOC-CRM standard is enough and appropriate to describe the ontologies associated with other kind of cultural institutions, like archives and libraries, or if new description approaches need to be considered to cope with those new assets;
  - study new options to implement the system storage, conducting tests on different digital repositories. For example, NoSQL, as graph databases, Document-Oriented databases, Object-Oriented databases, etc. Those tests shall be designed to check whether those alternatives are a good solution for storing large amounts of data;
  - look for other approaches regarding the connection between the digital repository and the ontology when different storage systems are used. The mapping step is a crucial issue in this proposal as it allows to query over the repository instances, so other alternatives shall be explored, namely R2RML must be considered.
- *Regarding the virtual Learning Space specification*:
  - draw and conduct experiments to appraise the usability and the efficiency of CaVa$^{\text{DSL}}$, as well as attest whether the Curator is capable of actually describing virtual Learning Spaces.
- *Regarding the final virtual Learning Space*:

– Conduct tests with end-users (visitors) to identify if they really can learn about the domain in which the virtual Learning Environment is embedded, as well as identify which are the best ways to learn. Is the navigation over concepts really the best technique?

# Appendices

# Appendix A

# Generated and used grammars

## A.1 CaVa$^{\text{grammar}}$

This appendix presents the developed ANTLR version of CaVa$^{\text{grammar}}$ (see Listing A.1).

Listing A.1: ANTLR version of CaVa$^{\text{grammar}}$

```
grammar cava;

cava: mainConfig header content footer ;

mainConfig: 'mainconfig' leftBracket learningSpaceTitle learningSpaceAbout? learningSpaceCarousel? rightBracket;

learningSpaceTitle: 'LS' 'title' colon TEXT comma ;
learningSpaceAbout: 'about' leftBracket (learningSpaceAboutParagraphs)+ rightBracket ;
learningSpaceAboutParagraphs: 'p' colon TEXT comma ;
learningSpaceCarousel: 'carousel' leftBracket (optionLearningSpaceCarousel)+ rightBracket ;
optionLearningSpaceCarousel: images | interval ;
images: 'images' leftBracket (optionLearningSpaceCarouselConfig)+ rightBracket ;
optionLearningSpaceCarouselConfig: 'caption' colon TEXT comma 'src' colon TEXT comma imageActive ;
imageActive: (active comma)? ;
active: 'active' ;
interval: 'interval' colon INT comma ;

header: 'menu' leftBracket (optionHeader)+ rightBracket ;

optionHeader: brand | backgroundColor | fontColor | behaviourStat | items ;
brand : 'brand' colon TEXT comma ;
backgroundColor: 'background' 'color' colon colorAlternative comma ;
fontColor: 'foreground' 'color' colon colorAlternative comma ;
behaviourStat: 'behavior' colon headerBehaviour comma ;
items: 'options' leftBracket (label)+ rightBracket ;
label : labelSimples | labelDropdown ;
labelSimples: 'label' colon TEXT comma 'url' colon TEXT comma 'extension' colon EXTENSION comma ;
labelDropdown: 'label' colon TEXT comma dropdown ;
dropdown: 'dropdown' leftBracket (dropdownList)+ rightBracket ;
```

```
dropdownList : 'dropdown' 'label' colon TEXT comma 'url' colon TEXT comma ;
headerBehaviour : 'fixed' | 'static' ;

content: exhibitions ;

exhibitions: 'exhibitions' leftBracket (exhibition)+ rightBracket ;
exhibition: 'exhibition' leftBracket (optionExhibition)+ rightBracket ;
optionExhibition: exhibitionTitle
                | exhibitionShortDescription
                | exhibitionIcon
                | exhibitionBehaviour
                | exhibitionAdditionalInfo
                | exhibitionType
                | exhibitionNotification
                | (queryOperators | sparql)
                ;
queryOperators  : all | one ;
all             : CONCEPT separator 'all' leftParenthesis parametersAll rightParenthesis labelsOptions ;
one             : CONCEPT separator 'one' leftParenthesis parametersOne rightParenthesis labelsOptions ;
sparql          :  'SPARQL' leftBracket sparqlStatement rightBracket labelsOptions ;
sparqlStatement: TEXT ;
separator       : '->' ;
leftParenthesis : '(' ;
rightParenthesis: ')' ;
parametersAll   : listName comma mappingOrTriplesFileName comma ontologyFileName ;
parametersOne   : listName comma mappingOrTriplesFileName comma ontologyFileName ;
listName        : TEXT ;
mappingOrTriplesFileName : TEXT ;
ontologyFileName : TEXT ;
labelsOptions    : leftBracket (labelsExhibitionRoom)+ rightBracket ;
labelsExhibitionRoom : elem (comma elem)* ;
elem            : TEXT | headerOfEachElement ;
headerOfEachElement : 'headerOfEachElement' colon TEXT ;
exhibitionTitle: 'title' colon TEXT comma ;
exhibitionShortDescription: 'short' 'description' colon TEXT comma ;
exhibitionIcon: 'icon' colon TEXT comma ;
exhibitionBehaviour: 'behavior' colon behaviourOptionExhibition comma ;
behaviourOptionExhibition: 'expanded' | 'collapsed' ;
exhibitionAdditionalInfo: 'additional' 'info' leftBracket (additionalInfoOptionExhibition)+ rightBracket ;
additionalInfoOptionExhibition: additionalInfoTitle | additionalInfoDescription ;
additionalInfoTitle: 'title' colon TEXT comma ;
additionalInfoDescription: 'description' colon TEXT comma ;
exhibitionType: 'type' colon typeOptionExhibition comma ;
typeOptionExhibition: 'permanent'
                    | 'temporary'
                    | 'special'
                    | 'future'
                    ;
exhibitionNotification: 'available' 'until' colon TEXT comma ;

footer: 'footer' leftBracket (optionFooter)+ rightBracket ;

optionFooter: footerImage | footerFormatDate | footerDeveloper | footerBehavior | footerStyle ;
footerImage: 'images' leftBracket (imageOptions)+ rightBracket ;
imageOptions: image | alignment ;
image: 'image' colon TEXT comma ;
footerFormatDate: 'format date' colon DATE comma ;
footerDeveloper: 'developer' leftBracket (developerOptions)+ rightBracket ;
developerOptions: developer | alignment ;
developer: 'name' colon TEXT comma ;
alignment: 'alignment' colon alignmentOption comma ;
alignmentOption: 'left' | 'right' ;
footerBehavior: 'behavior' colon footerBehaviorOption comma ;
footerBehaviorOption: 'fixed' | 'static' ;
footerStyle: 'style' colon footerStyleOption ;
footerStyleOption: extended | 'condensed' comma ;
extended: 'extended' leftBracket (footerExtendedOptions)+ rightBracket ;
```

```
footerExtendedOptions: 'title' colon TEXT comma 'subtitle' colon
                       TEXT leftBracket (footerExtendedOptionsExtended)+ rightBracket ;
footerExtendedOptionsExtended: 'label' colon TEXT comma 'link' colon TEXT comma
                               'icon' colon TEXT comma 'icon color' colon colorAlternative comma ;
leftBracket : '[' ;
rightBracket : ']' ;
colon : ':' ;
comma : ',' ;

colorAlternative: COLORNAME | HEXCODE ;
COLORNAME : 'aqua'
          | 'black'
          | 'blue'
          | 'crimson'
          | 'fuchsia'
          | 'gray'
          | 'green'
          | 'lime'
          | 'maroon'
          | 'navy'
          | 'olive'
          | 'orange'
          | 'purple'
          | 'red'
          | 'silver'
          | 'teal'
          | 'white'
          | 'yellow'
          ;

HEXCODE : '#' HEX_DIGIT+;
DATE    : DATE_DIGIT+;
TEXT: STRING ;
EXTENSION   : [a-zA-Z]+ ;

fragment
DATE_DIGIT  : '"'('Y' | 'y' | 'M' | 'm' | 'D' | 'd')+'"' ;

fragment
STRING
   :  '"' ( ESC_SEQ | ~('"') )* '"' ;

fragment
ESC_SEQ
   :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
   |   UNICODE_ESC
   |   OCTAL_ESC
   ;

fragment
OCTAL_ESC
   :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
   |   '\\' ('0'..'7') ('0'..'7')
   |   '\\' ('0'..'7')
   ;

fragment
UNICODE_ESC
   :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT ;

fragment
HEX_DIGIT   : ('0'..'9'|'a'..'f'|'A'..'F') ;

INT         : [0-9_.]+ ;
CONCEPT         : [a-zA-Z0-9_.]+ ;
LINECOMMENT : '#' ~[\r\n]* -> channel(HIDDEN) ;
WS : [ \t\r\n\f]+ -> channel(HIDDEN) ;
```

## A.2 OBDA Ontop grammar (*cavaSPARQL*)

This appendix presents the developed ANTLR version of the *cavaSPARQL* grammar (see Listing A.2).

Listing A.2: ANTLR version of *CaVaSPARQL* grammar

```
grammar cavaSPARQL;

cavaSPARQL  : (mappingid comma target)+ ;

target      : uri relation object (relations)* period ;
mappingid   : VALUE ;
concept     : prefix colon VALUE ;
comma       : ',' ;
uri         : prefix colon uriprefix VALUE hashtag leftBracket VALUE rightBracket ;
relations   : semicolon relation object ;
uriprefix   : 'URI/' ;
hashtag     : '#' ;
leftBracket : '{' ;
rightBracket: '}' ;
relation    : rdftype | predicate ;
rdftype     : 'a' ;
predicate   : prefix colon VALUE ;
prefix      : VALUE ;
colon       : ':' ;
object      : uri | concept | placeholder | string ;
string      : STRING ;
placeholder : leftBracket VALUE rightBracket ;
semicolon   : ';' ;
period      : '.' ;

VALUE       : [a-zA-Z][a-zA-Z0-9_.-]* ;
STRING      : '"' [ a-zA-Z0-9_.]+ '"' ;
WS          : [ \t\r\n]+ -> skip ;
```

# Appendix B

# Passport Application Form Data

The following list shows the data needed to get a passport by an emigrant.

- **General data about the emigrants documents**: Emigrants document number in City Council; Emigrants document Number in Board of Emigration; and Year.

  **Meaning**: data that identifies the emigration's documents.

- **General data about the emigrant**: Name; Nationality; Date of Birth; Filiation; Civil Status; Name of Spouse; Number of Identity; and Residence.

  **Meaning**: data that identifies the emigrant, their parents, the spouse and address (residence).

- **Data of the applicant**: Name; Residence (place and town (parish)); and, Country and location.

  **Meaning**: data that identifies the applicant and location that he intends to emigrate.

- **Dispatching**[1]: Date that the emigration document was sent; Number of the craft; Date that was granted; Number of the license of emigration; Passport number; Ship, class and company; and Boarding date.

  **Meaning**: data that identifies the dispatch of the documents of the emigrant.

- **Attached documents**: Several documents such as: medical certificate, birth certificate, marriage certificate, work contract, etc.

  **Meaning**: documents provided to get a visa. There is a list of documents to be marked as checked when the applicant deliver them.

- **Family members accompanying the emigrant**: Name; Date of Birth; Age; kinship with the emigrant; and literary qualifications.

  **Meaning**: data that identifies the members of family that accompanying the emigrant.

- **Desired type of transportation**: Desired type of transportation; Approximate date of shipment; If the pass on the company is paid; Port of Boarding; and Port of landing.

  **Meaning**: data that identifies the desired type of transportation by the emigrant. These data are grouped as "Boarding" in the emigration documents.

- **Qualifications (literary and professional) and criminal records**: Current occupation; Literary qualifications; Entity where worked; If the applicant has already been judged by the courts; If the applicant or his relatives have any case pending.

  **Meaning**: data that identifies the literary and professional qualifications of the applicant, as well as data about the emigrant's criminal records.

[1] Portuguese: expediente

- **Family members in charge of the emigrant who remain in the country**: Name; Age; kinship with the emigrant; and Residence of each person of the family.

  **Meaning**: data that identifies the family members in charge of the emigrant who remain in the country.

- **Previous travel abroad**: Whether or not it is the first time that the applicant is traveling; Date on which returned; Passport number and date; Entity that issued the passport; Whether or not the applicant came repatriated; Reasons for repatriation; and if the applicant has already submitted a request to emigrate.

  **Meaning**: data that identifies if is the first time that the applicant is traveling to another country.

- **Details of the person calling the emigrant**: Name; Residence; kinship of the emigrant with the caller; How long is resident in the country that the emigrant goes; and Data of the passport that the caller did leave the country.

  **Meaning**: data that identifies the caller who is calling the emigrant to the destination country.

- **Employment contract**: Name and Residence of the contractor; If the applicant knows the contractor and since when; If the applicant does not know the contractor, how it obtained the contract; Name, residence, occupation and filiation of the intermediary; Passport that the intermediary did leave the country; kinship of the applicant with the intermediary and how much was paid or has to pay for the employment contract.

  **Meaning**: data that identifies the contractor and the intermediary, if exists.

- **Details about women and minors employed**: Whether or not already worked to the contractor or to the family of him/her; Duration of the work; Occupation.

**Meaning**: data that identifies women and minor employed.

- **Details of the married men who leave the family in the country of origin**: Knowledge of wife about the target location of her husband; and if the wife considers that her maintenance is assured by the husband in the country that she stays.

  **Meaning**: data about the married man who leaves the family in the country of the origin. This data (statement) should be provided by the wife of the married man.

- **Aid in the destination country**: Whether or not the applicant has family in the destination country; Name of the people who will give the aids; Degree of kinship; Residence; Occupation; How long he/she lives in that country; What are the possibilities to provide assistance; If the person who will give assistance has already traveled to Portugal. If yes, when and the length of stay; If has maintained correspondence with these relatives; If the relatives know the applicant's claim and vowed to assist it; and how it will help the applicant in the country of destination.

  **Meaning**: data about the people who will aid the emigrant in the destination country.

# Appendix C

# Logical and Physical Model of bdME

Figure C.1 presents the logical model of bdME.

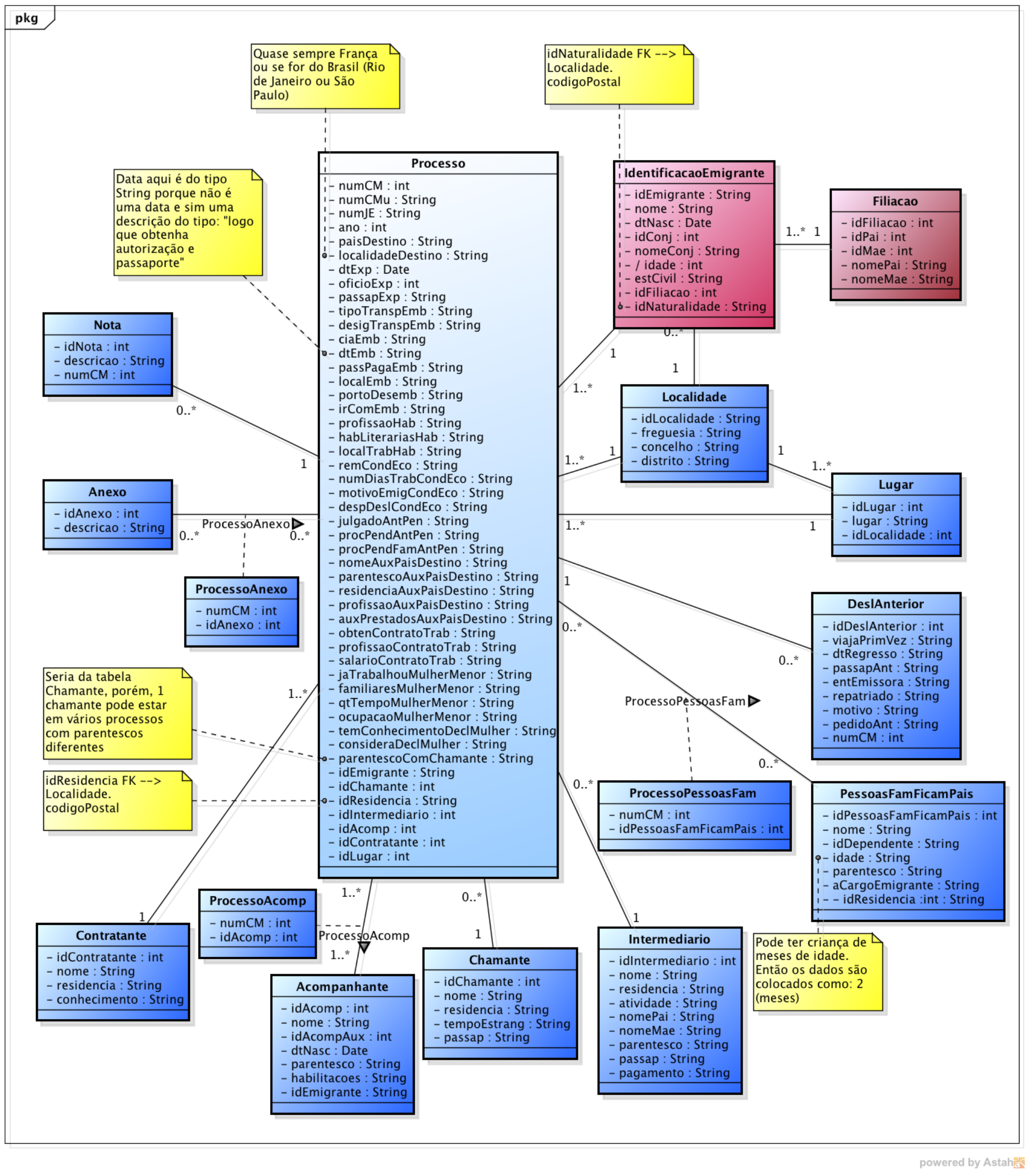


Figure C.1: Logical model of the database

Figure C.2 illustrates the physical model of bdME.

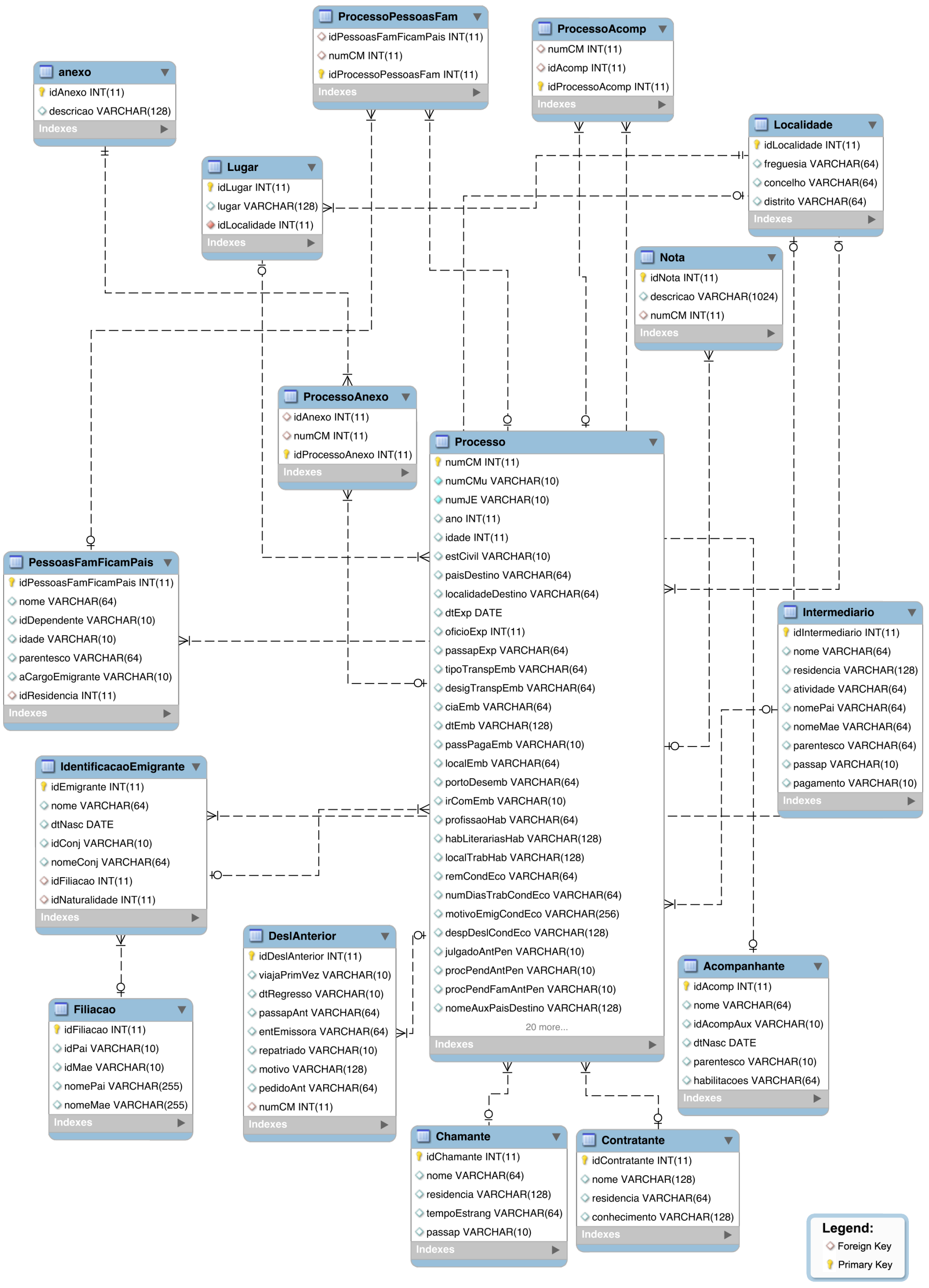


Figure C.2: Physical model of the bdME

# Appendix D

# Physical Model of bdFasti

Figure D.1 illustrates the physical model of bdFasti.

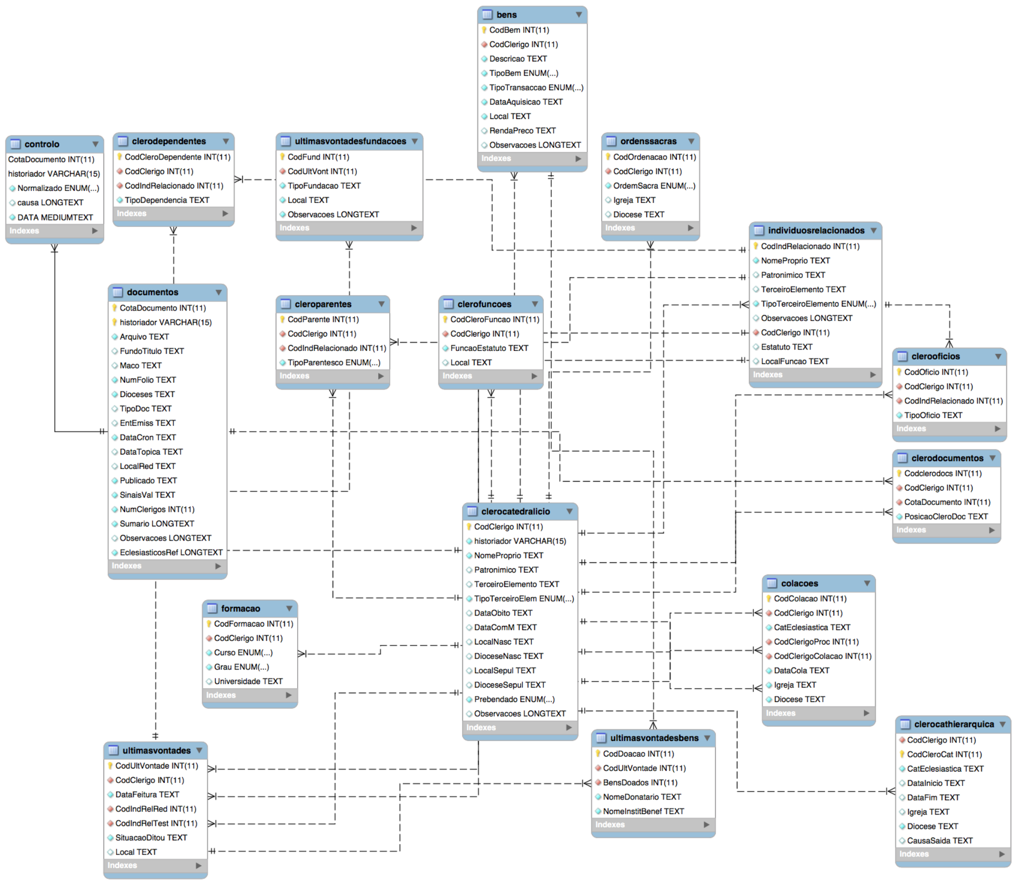


Figure D.1: Physical model of bdFasti

# Bibliography

[ACCU, 1998] ACCU (1998). *Preservation and Promotion of the Intangible Cultural Heritage.* Asia-Pacific Cultural Centre for UNESCO (ACCU) 6, Fukuromachi, Shinjuku-ku Tokyo.

[Allemang and Hendler, 2011] Allemang, D. and Hendler, J. (2011). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition.

[Almeida et al., 2001] Almeida, J. J., Rocha, J. G., Henriques, P. R., Moreira, S., and Simões, A. (2001). Museu da Pessoa – arquitectura. In *Encontro Nacional da Associação de Bibliotecários, Arquivista e Documentalistas, ABAD'01.* BAD.

[Araújo et al., 2017] Araújo, C., Henriques, P. R., and Martini, R. G. (2017). Automatizing ontology population to drive the navigation on virtual learning spaces. In *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6.

[Araújo, 2016] Araújo, C. E. (2016). Building the Museum of the Person based on a combined CIDOC-CRM - FOAF - DBpedia Ontology. Master's thesis, Universidade do Minho, Braga, Portugal.

[Bartalos and Bieliková, 2007] Bartalos, P. and Bieliková, M. (2007). An approach to object-ontology mapping. In *Slovak University of Technology*, pages 9–16.

[Brown, 2005] Brown, M. (2005). Learning spaces. *EDUCAUSE e-Books*, pages 12.2–12.22.

[Cadavid et al., 2009] Cadavid, J. J., Quintero, J. B., Lopez, D. E., Hincapié, J. A., Brogi, A., João, A., and Anaya, R. (2009). A domain specific language to generate web applications. In *CIbSE*, pages 139–144.

[Callaghan et al., 2009] Callaghan, M., McCusker, K., Losada, J., Harkin, J., and Wilson, S. (2009). Integrating virtual worlds & virtual learning environments for online education. In *Games Innovations Conference, 2009. ICE-GIC 2009. International IEEE Consumer Electronics Society's*, pages 54–63.

[Carvalho, 2014] Carvalho, N. (2014). *Conclave: writing programs to understand programs.* PhD dissertation, Universidade do Minho, Braga, Portugal.

[Ceh et al., 2011] Ceh, I., Crepinsek, M., Kosar, T., and Mernik, M. (2011). Ontology driven development of domain-specific languages. *Comput. Sci. Inf. Syst.*, 8(2):317–342.

[Centre, 2005] Centre, W. H. (2005). *World Heritage Information Kit.* Unesco World Heritage Centre.

[da Purificação and Silva, 2009] da Purificação, C. E. P. and Silva, P. C. (2009). A domain-specific language for modeling web user interactions with a model driven approach. In Matti Rossi, Jonathan Sprinkle, J. G. and Tolvanen, J.-P., editors, *Proceedings of the 9th OOPSLA Workshop on Domain Specific Modelling*, DSM'09, Orlando, USA. Helsinki School of Economics.

[da Silva Nascimento, 2009] da Silva Nascimento, J. (2009). Emigração madeirense para a venezuela (1940-1974). Master's thesis, Dissertação submetida à Universidade da Madeira para obtenção do Grau de Mestre em Estudos Interculturais – Estudos Luso-Brasileiros. Universidade da Madeira.

[Das et al., 2012] Das, S., Sundara, S., and Cyganiak, R. (2012). R2RML: RDB to RDF mapping language. Technical report, W3C.

[Davallon, 1986] Davallon, J. (1986). *Claquemurer, pour ainsi dire, tout l'univers: La mise en exposition.* Alors (Paris. 1983). Centre Georges Pompidou, Centre de création industrielle.

[Desvallées, 2010] Desvallées, A. (2010). *Key Concepts of Museology.* Armand Colin.

[Deursen et al., 2000] Deursen, A. V., Klint, P., and Visser, J. (2000). Domain-specific languages: An annotated bibliography. *ACM SIGPLAN NOTICES*, 35:26–36.

[Doerr, 2009] Doerr, M. (2009). *Ontologies for Cultural Heritage.* Springer Publishing Company, Incorporated.

[Doerr et al., 2003] Doerr, M., Hunter, J., and Lagoze, C. (2003). Towards a core ontology for information integration. *J. Digit. Inf.*, 4(1).

[Ekwelem et al., 2011] Ekwelem, V. O., Okafor, V. N., and Ukwoma, S. C. (2011). Preservation of cultural heritage: The strategic role of the library and information science professionals in south east nigeria. *Library Philosophy and Practice.*

[Feather, 2006] Feather, J. (2006). Managing the documentary heritage: issues fro the present and future.

[Fensel, 2000] Fensel, D. (2000). Relating ontology languages and web standards.

[Fonseca, 2014] Fonseca, J. (2014). Converting ontologies into dsls. Master's thesis, Universidade do Minho, Campus Gualtar, Braga, Portugal.

[Fowler, 2010] Fowler, M. (2010). *Domain-Specific Languages.* Addison-Wesley Signature Series (Fowler). Pearson Education.

[Franconi, 2008] Franconi, E. (2008). Ontologies and databases: Myths and challenges. *Proc. VLDB Endow.*, 1(2):1518–1519.

[Gali et al., 2004] Gali, A., Chen, C. X., Claypool, K. T., and Uceda-Sosa, R. (2004). *Conceptual Modeling for Advanced Application Domains: ER 2004 Workshops CoMoGIS, CoMWIM, ECDM, CoMoA, DGOV, and eCOMO, Shanghai, China, November 8-12, 2004. Proceedings*, chapter From Ontology to Relational Databases, pages 278–289. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Gava and Menezes, 2003] Gava, T. B. S. and Menezes, C. S. D. (2003). Uma ontologia de domínio para a aprendizagem cooperativa. *XIV Simpósio Brasileiro de Informática na Edução NCE IMUFRJ*, pages 336–345.

[Ghiselli et al., 2005] Ghiselli, C., Trombetta, A., Bozzato, L., and Binaghi, E. (2005). Semantic web meets virtual museums: The domus naturae project.

[Ghosh, 2011] Ghosh, D. (2011). *DSLs in Action*. Manning Pubs Co Series. Manning.

[Goos, 2006] Goos, M. (2006). Creating learning spaces. *The Annual Clements/Foyster Lecture*.

[Gorman and Shep, 2006] Gorman, G. and Shep, S. (2006). *Preservation Management for Libraries, Archives and Museums*. Henry Ford Estate collection. Facet.

[Gruber, 1993] Gruber, T. R. (1993). Toward principles for the design of ontologies used for knowledge sharing. In *International Journal of Human-Computer Studies*, pages 907–928. Kluwer Academic Publishers.

[Guarino, 1997] Guarino, N. (1997). Understanding, building and using ontologies: A commentary to using explicit ontologies in kbs development. *International Journal of Human and Computer Studies*, pages 293–310.

[Hess et al., 2015] Hess, M., Robson, S., Serpico, M., Amati, G., Pridden, I., and Nelson, T. (2015). Developing 3d imaging programmes–workflow and quality control. *J. Comput. Cult. Herit.*, 9(1):1:1–1:11.

[Hudak, 1998] Hudak, P. (1998). Modular domain specific languages and tools. In *Proceedings of the 5th International Conference on Software Reuse*, ICSR '98, pages 134–, Washington, DC, USA. IEEE Computer Society.

[Hürst et al., 2016] Hürst, W., Tan, X. J., and de Coninck, F. (2016). Using digital extensions to create new vr museum experiences. In *Proceedings of the 13th International Conference on Advances in Computer Entertainment Technology*, ACE '16, pages 45:1–45:6, New York, NY, USA. ACM.

[Hyvonen, 2009] Hyvonen, E. (2009). Semantic portals for cultural heritage. In *Handbook on Ontologies*, pages 757–778. Springer Publishing Company, Incorporated.

[Hyvönen et al., 2005] Hyvönen, E., Mäkelä, E., Salminen, M., Valo, A., Viljanen, K., Saarela, S., Junnila, M., and Kettula, S. (2005). Museumfinland-finnish museums on the semantic web. *Web Semant.*, 3(2-3):224–241.

[ICOM/CIDOC, 2015] ICOM/CIDOC (2015). Definition of the CIDOC Conceptual Reference Model. Technical report, ICOM/CIDOC.

[Inukai et al., 2007] Inukai, Y., Gehrmann, A., Nagai, Y., and Ishizu, S. (2007). Rough set theory using similarity of objects described by ontology. In *Proceedings of the 51st Annual Meeting of the ISSS - 2007, Tokyo, Japan*.

[Ivey, 2004] Ivey, B. (2004). Issues in intangible cultural heritage. Technical report, Council on Library and Information Resources.

[Jaligama and Liarokapis, 2011] Jaligama, V. and Liarokapis, F. (2011). An online virtual learning environment for higher education. In *Games and Virtual Worlds for Serious Applications (VS-GAMES), 2011 Third International Conference on*, pages 207–214.

[Jaén et al., 2005] Jaén, J., Mocholí, J. A., Esteve, J. M., Bosch, V., and Canós, J. H. (2005). Momo: Enabling social multimedia experiences in hybrid museums.

[Jorge et al., 2004] Jorge, A. M., Rodrigues, A. M., Vilar, H., Henriques, P. R., and Lopes, S. (2004). Construção e exploração de uma base de dados prosopográfica normalizada do clero catedralício português. In *Sistemas Informáticos para Análise de Dados Demográficos (SIA2D'04)*, Cadernos NEPS, pages 49–67, Guimarães, Portugal.

[Karsai et al., 2009] Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schneider, M., and Völkel, S. (2009). Design guidelines for domain specific languages. In Rossi, M., Sprinkle, J., Gray, J., and Tolvanen, J.-P., editors, *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*, pages 7–13.

[Kejriwal and Bedekar, 2015] Kejriwal, A. A. and Bedekar, M. (2015). Mobidsl - a domain specific langauge for mobile web applications: developing applications for mobile platform without web programming. In Steffen Fries, Siemens AG, C. T. and Freire, M., editors, *The Tenth International Conference on Internet and Web Applications and Services*, ICIW 2015, Brussels, Belgium. IARIA.

[Kersten et al., 2017] Kersten, T. P., Tschirschwitz, F., and Deggim, S. (2017). Development of a Virtual Museum Including a 4d Presentation of Building History in Virtual Reality. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 361–367.

[Kogalovsky, 2012] Kogalovsky, M. R. (2012). Ontology-based data access systems. *Programming and Computer Software*, 38(4):167–182.

[Kontchakov et al., 2014] Kontchakov, R., Rezk, M., Rodríguez-Muro, M., Xiao, G., and Zakharyaschev, M. (2014). Answering sparql queries over databases under owl 2 ql entailment regime. In *Proceedings of the 13th International Semantic Web Conference - Part I*, ISWC '14, pages 552–567, New York, NY, USA. Springer-Verlag New York, Inc.

[Kosar et al., 2008] Kosar, T., López, P. E. M., Barrientos, P. A., and Mernik, M. (2008). A preliminary study on various implementation approaches of domain-specific language. *Information and Software Technology*, 50(5):390 – 405.

[Krstićev et al., 2016] Krstićev, D. B., Tesendić, D., Jović, M., and Bajić, Z. (2016). Dsl for web application developement. In *Proceedings of the 6th International Conference on Information Society and Technology ICIST 2016*, pages 174–178, Belgrade, Serbia. Society for Information Systems and Computer Networks.

[Lagoze and Hunter, 2001] Lagoze, C. and Hunter, J. (2001). The abc ontology and model. In *Proc. of the Int. Conf. on Dublin Core and Metadata Applications*, pages 160–176. National Institute of Informatics, Tokyo, Japan.

[Lenzerini, 2011] Lenzerini, M. (2011). Ontology-based data management. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 5–6, New York, NY, USA. ACM.

[Librelotto et al., 2008a] Librelotto, G. R., Freitas, L. O., Gasse, J. B., Copetti, M., Turchetti, R. C., da Silva, F. L., and Augustin, I. (2008a). Uma ferramenta para o processamento da representação do domínio de atividades médicas. *Hífen*, pages 25–32.

[Librelotto et al., 2008b] Librelotto, G. R., Gassen, J. B., Freitas, L. O., Silveira, M. C., Silva, F. L., Augustin, I., and Henriques, P. (2008b). Uma ontologia aplicada a um ambiente pervasivo hospitalar. *In: 8 Conferência da Associação Portuguesa de Sistemas de Informação*, pages 34–34.

[Lomas and Oblinger, 2006] Lomas, C. and Oblinger, D. G. (2006). *Students Practices and Their Impact on Learning Spaces*. EDUCAUSE. Editor - Diana G. Oblinger.

[Madsen and Madsen, 2015] Madsen, J. B. and Madsen, C. B. (2015). Handheld visual representation of a castle chapel ruin. *J. Comput. Cult. Herit.*, 9(1):6:1–6:18.

[Martinez-Cruz et al., 2012] Martinez-Cruz, C., Blanco, I. J., and Vila, M. A. (2012). Ontologies versus Relational Databases: Are they so different? A comparison. *Artif. Intell. Rev.*, 38(4):271–290.

[Martini et al., 2015] Martini, R., Guimarães, M., Librelotto, G., and Henriques, P. (2015). Storing archival emigration documents to create virtual exhibition rooms. In Rocha, A., Correia, A. M., Costanzo, S., and Reis, L. P., editors, *New Contributions in Information Systems and Technologies*, volume 353 of *Advances in Intelligent Systems and Computing*, pages 403–409. Springer International Publishing.

[Martini et al., 2016a] Martini, R. G., Araújo, C., Almeida, J. J., and Henriques, P. R. (2016a). OntoMP, An Ontology to Build the Museum of the Person. In Rocha, Á., Correia, A. M., Adeli, H., Reis, L. P., and Mendonça Teixeira, M., editors, *New Advances in Information Systems and Technologies*, pages 653–661, Cham. Springer International Publishing.

[Martini et al., 2016b] Martini, R. G., Araújo, C., Librelotto, G. R., and Henriques, P. R. (2016b). A Reduced CRM-Compatible Form Ontology for the Virtual Emigration Museum. In Rocha, Á., Correia, M. A., Adeli, H., Reis, P. L., and Teixeira, M. M., editors, *New Advances in Information Systems and Technologies*, pages 401–410. Springer International Publishing, Cham.

[Martini et al., 2018] Martini, R. G., Araújo, C., Henriques, P. R., and Pereira, M. J. V. (2018). *Trends and Advances in Information Systems and Technologies*, chapter CaVa: An Example of the Automatic Generation of Virtual Learning Spaces. Springer International Publishing, Switzerland.

[Martini et al., 2017] Martini, R. G., Guimarães, M., Librelotto, G. R., and Henriques, P. R. (2017). Creating Virtual Exhibition Rooms from Emigration Digital Archives. *Univers. Access Inf. Soc.*, 16(4):823–833.

[Martini and Henriques, 2017a] Martini, R. G. and Henriques, P. R. (2017a). Automatic Generation of Virtual Learning Spaces Driven by CaVa$^{DSL}$: An Experience Report. *SIGPLAN Not.*, 52(12):233–245.

[Martini and Henriques, 2017b] Martini, R. G. and Henriques, P. R. (2017b). Automatic Generation of Virtual Learning Spaces Driven by CaVa$^{DSL}$: An Experience Report. In *Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, GPCE 2017, pages 233–245, New York, NY, USA. ACM.

[Martini et al., 2016c] Martini, R. G., Librelotto, G. R., and Henriques, P. R. (2016c). Formal Description and Automatic Generation of Learning Spaces Based on Ontologies. *Procedia Computer Science*, 96:235 – 244. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 20th International Conference KES-2016.

[Masolo et al., 2003] Masolo, C., Borgo, S., Gangemi, A., Guarino, N., and Oltramari, A. (2003). WonderWeb deliverable D18 ontology library (final). Technical report, IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web.

[Matsuzono, 2004] Matsuzono, M. (2004). Museums, intangible cultural heritage and the spirit of humanity. *ICOM NEWS*, 4:13–14.

[MCEECDYA, 2008] MCEECDYA (2008). *Melbourne Declaration on Educational Goals for Young Australians*. Ministerial Council for Education, Early Childhood Development and Youth Affairs.

[McGuinness and Harmelen, 2004] McGuinness, D. L. and Harmelen, F. V. (2004). Owl web ontology language overview.

[Mernik et al., 2005] Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344.

[Moore et al., 2011] Moore, J. L., Dickson-Deane, C., and Galyen, K. (2011). e-learning, online learning, and distance learning environments: Are they the same? *The Internet and Higher Education*, 14(2):129–135.

[Nguyen et al., 2010] Nguyen, P. H., Kaneiwa, K., and Nguyen, M.-Q. (2010). Ontology inferencing rules and operations in conceptual structure theory. In *Proceedings of Australasian Ontology Workshop (AOW 2010), Australia*, pages 61–70.

[Nisheva-Pavlova et al., 2008] Nisheva-Pavlova, M. M., Pavlov, P. I., and Devreni-Koutsouki, A. S. (2008). Ontology-based access to digitized cultural heritage and archival collections.

[Noy and Mcguinness, 2001] Noy, N. F. and Mcguinness, D. L. (2001). Ontology development 101: A guide to creating your first ontology. Technical report.

[Oldman and Labs, 2014] Oldman, D. and Labs, C. (2014). The cidoc conceptual reference model (cidoc-crm): Primer. *International Council of Museums (ICOM)*, 1.

[Oliveira, 2009] Oliveira, N. (2009). Improving program comprehension tools for domain specific languages. Master's thesis, Universidade do Minho, Braga, Portugal.

[Oliveira et al., 2009] Oliveira, N., Pereira, M. J. V., Henriques, P. R., and da Cruz, D. (2009). Domain-Specific Languages - A Theoretical Survey. In *Proceedings of the 3rd Compilers, Programming Languages, Related Technologies and Applications (CoRTA'2009)*, pages 35–46.

[Parr, 2013] Parr, T. (2013). *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition.

[Piccoli et al., 2001] Piccoli, G., Ahmad, R., and Ives, B. (2001). Web-based virtual learning environments: A research framekwork and a preliminary assessment of effectiveness in basic it skills training. *MIS Q.*, 25(4):401–426.

[Poggi et al., 2008] Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., and Rosati, R. (2008). Linking data to ontologies. In Spaccapietra, S., editor, *Journal on Data Semantics X*, pages 133–173. Springer-Verlag, Berlin, Heidelberg.

[Rainie et al., 2010] Rainie, H., Anderson, J., and Fox, S. (2010). *Challenges and Opportunities: The Future of the Internet*. The Future of the Internet. Cambria Press.

[Ravanello, 2018] Ravanello, R. B. (2018). *Narrativa para bens culturais: tecnologias e aplicabilidades da fotografia digital expandida em museus virtuais*. PhD thesis, Universidade do Minho, Campus Gualtar, Braga, Portugal.

[Ribeiro, 1986] Ribeiro, F. (1986). *Emigração portuguesa: algumas características dominantes dos movimentos no período de 1950 a 1984*. Série Migrações: Sociologia. Secretaria de Estado das Comunidades Portuguesas, Centro de Estudos.

[Rodríguez-Muro et al., 2012] Rodríguez-Muro, M., Hardi, J., and Calvanese, D. (2012). Quest: Efficient sparql-to-sql for rdf and owl. In *Proceedings of the 2012th International Conference on Posters & Demonstrations Track - Volume 914*, ISWC-PD'12, pages 53–56, Aachen, Germany, Germany. CEUR-WS.org.

[Rodríguez-Muro et al., 2013] Rodríguez-Muro, M., Kontchakov, R., and Zakharyaschev, M. (2013). *Ontology-Based Data Access: Ontop of Databases*, pages 558–573. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Ruge, 2008] Ruge, A. (2008). Museum professions - a european frame of reference. *ICOM NEWS*, pages 1–39.

[Sacher et al., 2013] Sacher, D., Biella, D., and Luther, W. (2013). Towards a versatile metadata exchange format for digital museum collections. In *Digital Heritage International Congress (DigitalHeritage), 2013*, volume 2, pages 129–136.

[Serra and Girardi, 2011] Serra, I. and Girardi, R. (2011). A process for extracting non-taxonomic relationships of ontologies from text. *Intelligent Information Management*, 3(4):119–124.

[Sharma et al., 2015] Sharma, S., Stigall, J., and Rajeev, S. (2015). Game-theme based instructional module for teaching object oriented programming. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 252–257.

[Simões and Almeida, 2003] Simões, A. and Almeida, J. J. (2003). Histórias de Vida + Processamento Estrutural = Museu da Pessoa. In *XATA 2003 — XML: Aplicações e Tecnologias Associadas*, page 16, Braga, Portugal. UM.

[Smith et al., 2004] Smith, M. K., Welty, C., and McGuinness, D. M. (2004). Owl web ontology language guide.

[Stefano Ceri and Matera, 2001] Stefano Ceri, P. F. and Matera, M. (2001). Webml application frameworks: a conceptual tool for enhancing design reuse. In *WWW10 Workshop Web Engineering, Hong Kong.*

[Stenou and Unesco, 2002] Stenou, K. and Unesco (2002). *Universal declaration on cultural diversity : a vision ; a conceptual platform ; a pool of ideas for implementation ; a new paradigm ; a document of the World Summit on Sustainable Development, Johannesburg, 26 August - 4 September 2002.* Cultural diversity series. Unesco.

[Stibe and Bicevskis, 2009] Stibe, A. and Bicevskis, J. (2009). Web site modeling and prototyping based on a domain-specific language. In *Computer Science and Information Technologies Vol. 751*, pages 7–21, Riga, Latvia. Scientific Paper, University of Latvia.

[Studer et al., 1998] Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data Knowl. Eng.*, pages 161–197.

[Tylor, 1871] Tylor, E. (1871). *Primitive Culture: Researches Into the Development of Mythology, Philosophy, Religion, Art, and Custom.* Number vol. 1 in Primitive Culture: Researches Into the Development of Mythology, Philosophy, Religion, Art, and Custom. J. Murray.

[Ullman, 2013] Ullman, L. (2013). *The Yii Book: Developing Web Applications Using the Yii PHP Framework*. Self-published.

[UNESCO, 1989] UNESCO (1989). *Draft Medium Term Plan 1990 1995*. United Nations Educational, Scientific and Cultural Organization.

[UNESCO, 1997] UNESCO (1997). *New information technologies: a key for adult learning?* Matthew Partidge, Hamburg.

[UNESCO, 2013] UNESCO (2013). *Draft Medium Term Strategy 2014 2021*. United Nations Educational, Scientific and Cultural Organization.

[Uschold and Gruninger, 1996] Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, pages 93–136.

[Uschold and Jasper, 1999] Uschold, M. and Jasper, R. (1999). A framework for understanding and classifying ontology applications.

[Visser, 2008] Visser, E. (2008). Generative and transformational techniques in software engineering ii. chapter WebDSL: A Case Study in Domain-Specific Language Engineering, pages 291–373. Springer-Verlag, Berlin, Heidelberg.

[Waterton and Watson, 2013] Waterton, E. and Watson, S. (2013). *HERITAGE AND COMMUNITY ENGAGEMENT: Collaboration or Contestation?* Taylor & Francis.

[Witten et al., 2009] Witten, I., Bainbridge, D., and Nichols, D. (2009). *How to Build a Digital Library*. Morgan Kaufmann series in multimedia information and systems. Elsevier Science.