Document Management at Finantech

João Miguel de Oliveira Figueiredo Varandas

July 24, 2017
Document Management at Finantech

João Miguel de Oliveira Figueiredo Varandas

Mestrado Integrado em Engenharia Informática e Computação

July 24, 2017
Abstract

Modern trends in business have companies working towards a culture of collaboration, agility and openness. The volume of information and ideas shared every day increases continuously as technologies evolve, as processes adapt to them and to an ever more competitive market. More than ever, the increments of time spent on retrieving important business documents with relevant information for the company and the individual collaborator add up to significant costs and loss of potential value.

The proponent of this work is Finantech, a Porto-based software company focused on the financial sector, where the business-environment work was carried out. They have identified a gap in their internal information systems, namely the lack of structure in the storage of documentation related to administrative services, finances and projects, which this work aims to fill in. To this purpose and as a special requirement, an ontology-based approach was employed for the modeling and structured storing of the documentation.

A review of the state of the art in ontologies and Electronic Document Management was conducted, followed by an analysis of the internal systems and processes of the company, domain model design and creation as an ontology, quad store mapping and querying, document management client development and internal testing with participants from the company. As a result of these activities, a working prototype was delivered, including a database (which communicates with the company’s working systems), loaded with the created structure and mock documents, a web service that abstracts the queries to the database and a web client used to manage and search for documents.
Resumo

As tendências atuais colocam as empresas em direção a uma cultura de colaboração, agilidade e abertura. O volume de informação e ideias partilhadas todos os dias aumenta continuamente com a evolução tecnológica, à medida que os processos são adaptados a esta realidade, num mercado cada vez mais competitivo. Os incrementos de tempo despendidos na procura de documentos importantes, com informação relevante à empresa e ao colaborador, resultam em custos significativos e perda de valor potencial.

O proponente deste trabalho é a Finantech, uma empresa de software com foco no setor financeiro sediada no Porto, onde o trabalho em ambiente empresarial irá decorrer. A Finantech identificou falhas nos seus sistemas de gestão internos, nomeadamente na falta de estrutura no armazenamento de documentação relativa a serviços administrativos, finanças e projetos. Com este problema em vista e como requisito específico, uma abordagem à base de ontologias foi empregue na modelação e armazenamento estruturado da documentação.

Foi conduzida uma revisão do estado da arte em ontologias e Gestão Documental Eletrónica, seguida de uma análise aos sistemas internos e processos da empresa, desenho do modelo do domínio e criação do mesmo como uma ontologia, mapeamento e interrogações de uma quad store, desenvolvimento de um cliente de gestão documental e testes internos com participantes da empresa. Como resultado destas atividades, foi entregue um protótipo funcional, que inclui uma base de dados que comunica com os sistemas em funcionamento na empresa e foi carregada com a estrutura criada e documentos de teste, um serviço web que abstraí as interrogações à base de dados e um cliente web com a capacidade de gerir e procurar documentos.
Acknowledgements

For my mother, much love and gratitude for everything.

A thank you to Professor Ana Paiva for the indispensable guidance in this new and challenging experience.

With special mention to the fine folks at Finantech, whom I wish to thank for welcoming me as one of their own, and especially Francisco Carvalho and João Marta da Cruz for the ideas and mentoring so graciously provided.

I am also grateful to all my colleagues and professors, who inspired and pushed me to overcome and transmitted invaluable knowledge and expertise.

To Mauro, whose generosity and camaraderie helped me throughout and was a constant positive presence along this path. Our friendship made it all worth it.

Another debt of gratitude is owed to my family, without whom I could not have even started this journey.

Finally, to my all, Ana Lídia: thank you for the love, kindness and support.

Thank you all for making it happen.

João Miguel Varandas
“I must not fear.
Fear is the mind-killer.
Fear is the little-death that brings total obliteration.”

Frank Herbert, Dune
## Contents

1 Introduction 1  
1.1 Context 1  
1.2 Motivation and Objectives 2  
1.3 Challenges and Obstacles 2  
1.4 Dissertation Structure 3  

2 Background 5  
2.1 Document Management Systems 5  
2.1.1 What is in a document? 6  
2.1.2 Document Management in an organizational context 7  
2.2 Ontologies 7  
2.2.1 Related concepts and applications 8  
2.2.2 Resource Description Framework (RDF) 9  
2.2.3 Ontology languages 10  
2.2.4 Ontology engineering methodologies 13  
2.2.5 Ontology engineering tools 14  
2.2.6 Existing documentation ontologies 15  
2.2.7 Summary 18  

3 Agidoc 19  
3.1 Problem Analysis 19  
3.2 Solution 20  
3.2.1 Solution Architecture 21  
3.2.2 Tools and technologies 23  
3.2.3 Domain model 24  
3.2.4 Ontology definition and storage 27  
3.2.5 Web Service Queries 31  
3.2.6 Application 35  
3.3 Summary 42  

4 Validation 45  
4.1 Method 45  
4.2 Results 46  
4.3 Discussion 47  

5 Conclusions 49  
5.1 Future Work 49  
5.2 Discussion 51
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1 On ontologies</td>
<td>51</td>
</tr>
<tr>
<td>5.2.2 On the system</td>
<td>51</td>
</tr>
<tr>
<td>5.2.3 On the domain model</td>
<td>51</td>
</tr>
<tr>
<td>5.3 Summary</td>
<td>52</td>
</tr>
<tr>
<td>References</td>
<td>53</td>
</tr>
</tbody>
</table>
List of Figures

2.1 RDF graph triple (adapted from [RDF14a]) ................................................. 9
2.2 Ontology tool categorization ( [SFGCVTGP11]) ............................................. 14
2.3 Virtuoso technical architecture ................................................................. 16

3.1 System architecture ..................................................................................... 21
3.2 Linked Data Views creation screen in Virtuoso Conductor .......................... 22
3.3 Linked Data views resulting definitions screen (partial) in Virtuoso Conductor .. 23
3.4 Condensed class diagram ........................................................................... 24
3.5 Document code format ............................................................................... 26
3.6 File code format ......................................................................................... 27
3.7 Generic Document diagram ....................................................................... 28
3.8 Sales Document diagram ........................................................................... 28
3.9 Helpdesk Document diagram ..................................................................... 29
3.10 Scrum Teams Document diagram ............................................................ 29
3.11 Administrative Services Document diagram ........................................... 30
3.12 Protége class view, with Document class selection panel ....................... 31
3.13 Application Sitemap ................................................................................ 35
3.14 Application Home page ........................................................................... 36
3.15 Document application page ..................................................................... 36
3.16 New Document application modal ........................................................... 37
3.17 File History application page ................................................................... 38
3.18 (Documents) By Department (or Group) application page .................... 39
3.19 (Documents) By Date application page ................................................... 39
3.20 Client Property application page ............................................................. 40
3.21 Projects application page ......................................................................... 41
3.22 Single Project application page ............................................................... 41
3.23 SQLite Application database schema ....................................................... 42
List of Tables

2.1 SPARQL example query results .............................................. 13
3.1 Document-related queries ....................................................... 32
3.2 File-related queries ............................................................. 34
4.1 Validation participant ratings ................................................... 46
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic-link library</td>
</tr>
<tr>
<td>DM</td>
<td>Document Management</td>
</tr>
<tr>
<td>DMS</td>
<td>Document Management System</td>
</tr>
<tr>
<td>EMS</td>
<td>Electronic Document Management</td>
</tr>
<tr>
<td>FIBO</td>
<td>Financial Industry Business Ontology</td>
</tr>
<tr>
<td>GRDDL</td>
<td>Gleaning Resource Descriptions from Dialects of Languages</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IRI</td>
<td>Internationalized Resource Identifier</td>
</tr>
<tr>
<td>IS</td>
<td>Information Science</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization of Standards</td>
</tr>
<tr>
<td>N3</td>
<td>Notation3</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>ODP</td>
<td>Ontology Design Pattern</td>
</tr>
<tr>
<td>OML</td>
<td>Ontology Markup Language</td>
</tr>
<tr>
<td>OMR</td>
<td>Ontology for Media Resources</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format</td>
</tr>
<tr>
<td>PMBOK</td>
<td>Project Management Body of Knowledge</td>
</tr>
<tr>
<td>PMI</td>
<td>Project Management Institute</td>
</tr>
<tr>
<td>PMP</td>
<td>Project Management plan</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>RDF Schema</td>
</tr>
<tr>
<td>SHOE</td>
<td>Simple HTML Ontology Extension</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>Unicode</td>
<td>Universal Coded Character Set</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XOL</td>
<td>Ontology Exchange Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

As Meier et al. put it, "it is hard to identify anything more fundamental and pervasive in organizations than documents" [MS96]. Documents are used for communication, as tools in process chains and as pieces of organizational memory. The reliance on these artifacts and the increasingly competitive, technologically advanced landscape of modern business create a need for tools, platforms and processes to manage documentation, that solve problems ranging from document duplication, retrieval, versioning, traceability, security and storage, among others. The absence of such mechanisms and the lack of (compliance with) standardization represents a barrier in communication between companies and added time in information retrieval, which may be a potential source of loss of value [DTG16].

Ontologies are a powerful tool for the modeling of domains and expression of knowledge in machine-readable syntax, with continuing growth in investigation and application, that enable computers to understand things implicitly defined by humans, bridging a gap in the field of intelligent systems and reducing frustration and disconnect between the intentions of the user and the rigidity of the machine. In using this tool, not only will documentation be properly stored, but also retrievable through related concepts in "tip-of-the-tongue" situations.

The value created by this work comes primarily in the modeling of the domain of documentation at Finantech, and on a secondary level in the system developed based on the model. The use of ontologies to create the model presented itself as a novel idea for the company and an opportunity to explore this emerging technology, the employment of which could serve as a practical example to be learned from and used as a launching point for future projects.

1.1 Context

The work was conducted in an organizational environment at Finantech\(^1\), a Porto-based software development company specialized in the financial sector, and is integrated in the field of Infor-
1.2 Motivation and Objectives

Finantech develops an in-house solution (Agifox) for Project Management based on its own internal processes and practices. However, documentation about projects, business processes, administrative services, sales, and more, is stored in a almost entirely unorganized fashion, which makes it difficult to retrieve documents, connect them with the key concepts and areas that they relate to, such as projects/tasks/clients/etc., establish a workflow and avoid duplication of information. This results in time and opportunities lost, partly because "the common denominator to many of the problems [of business processes] is related into information sharing and exploitation. In many cases, the information exists but is not available and when it is accessible its validity may not be coherent. Documents and their consistency, usage, proper storage and linking provide a source for improvement when business processes are to be developed" [EHL01]. Taking these needs in consideration, the main objectives of this work were defined as:

- Analysis of the company’s information systems and their approach to documentation classification and organization
- Modeling and implementation of an ontology of the problem domain
- Mapping of existing data to the created ontology and consolidation of the information in a database
- Creation of an application for the search and organization of documentation

1.3 Challenges and Obstacles

Apart from some technical challenges that are addressed in chapter 3, there were a few obstacles to the development of the work:

- **Analysis**: The analysis of Finantech’s current information systems proved somewhat challenging. The schema created for their current databases was created under time constraints, has suffered many changes and is undergoing a restructuring for future developments. They have used a very specific nomenclature for fields and tables as well, which required guidance in interpreting.

- **Scoping**: At first, the proponent’s were unsure of the domain of the documentation to be approached, and no real samples were made available to analyze, which created a dependency
on their input to develop a model. There were also unrealistic expectations for what the final software would be able to achieve, given all the research, planning and trial and error the project would require, making it necessary to constantly manage these expectations and parse new ideas, sometimes well into development.

- **Database solution**: The software solution for data storage that the company strongly suggested to use, OpenLink Virtuoso, is one of the few that met its requirements of compatibility with their existing technologies. This proved difficult to use at times due to inexperience with the particular solution on top of the already unfamiliar area of ontology design and use. It proved to be very complex, with many separate modules, and a scattered and sometimes out of date documentation. Additionally, the developed system is dependent on version 7 of the enterprise version of this software, for which a specific license must be acquired after an evaluation period. It is unclear at this time if the solution is compatible with another version of the same software, which is in active development.

### 1.4 Dissertation Structure

In chapter 2, there is brief overview of concepts in Document Management and its place in organizations, as well as a view on the state of the art of ontologies, with a focus on the most recent and relevant standards, technologies, and related work. Chapter 3 contains a detailed characterization of the problem at hand and of the approach taken in designing and implementing a working solution, including the architecture of the system, modeling of the domain, structure and functionality of the developed applications and relevant queries, as well as accounts of issues faced and decisions made. In chapter 5, there is a view on the validation process and its results, provides a global assessment of what was done and lays out some possible future improvements.
Introduction
Chapter 2

Background

The background to this work is split into two major sections, addressing the most important fields approached during project, Document Management (Systems) and Ontologies, with some important concepts and prior work in these fields.

2.1 Document Management Systems

Document Management (DM) is a deep field, defined in many ways by various authors throughout the years, in different degrees of complexity. In 1978, a document-based system was described by Swanson and Culnan by contrast to a data-based system, in that the unit of information of the former is the document, an ordered set of images or textual information and the latter’s information is composed of individual data records [SC78]. Lee et al., in 1984, identified creating, editing, distributing and storing documents as the main activities in DM [LWL84]. In 1995, Sprague et al. defined the subject as the “creation, storage, organization, transmission, retrieval, manipulation, update, and eventual disposition of documents to fulfill an organizational purpose” and presented the concept of Electronic Document Management (EDM) as “the application of technology to save paper, speed up communications, and increase the productivity of business processes” [Spr95]. David et al. [DND13] defined DM as “the activities involving the capture or receipt of documents, version and format control, their storage, maintenance, retrieval and disposal entail”. There are many more definitions of Document Management and while the scope and context of these varies, in Information Science, the definitions often overlap and focus on identifying the activities that this form of management encompasses. Creating, categorizing, storing, versioning and retrieving are the most often cited and defining pursuits of Document Management.

For the purpose of this work, different perspectives on Document Management and existing Document Management Systems (DMS) were gathered and studied, keeping in mind that there were very specific requirements to be fulfilled and many of the concepts observed in the state
Background

of the art are not applicable to their full extent, as presented in the literature. The evaluation of existing technologies was also limited by predetermined constraints, as discussed in section 3.2.2.

2.1.1 What is in a document?

To talk about Document Management, one must first ask the question: "What is a document?". Merriam-Webster defines "Document" as "a writing conveying information", "a material substance (such as a coin or stone) having on it a representation of thoughts by means of some conventional mark or symbol" and "a computer file containing information input by a computer user and usually created with an application (such as a spreadsheet or word processor)", among other definitions pertaining to the field of Law. Obviously, some definitions fit the context of Information Systems more than others, but in a general way, it can be gathered that a document is seen as an artifact that holds information in some way, be it physical or digital.

In Buckland’s 1997 paper posing the direct question of what a document is, many different views are exposed, some of which are more applicable to the current landscape of an ubiquitous digital medium. We see narrow definitions that consider documents as only written records of or any material source of information, and some wider views, where documents are "graphic and written records [that] are representations of ideas or of objects" and "any physical or symbolic sign, preserved or recorded, intended to represent, to reconstruct, or to demonstrate a physical or conceptual phenomenon" [Buc97a]. The latter definitions, some of which predate modern digital systems, encompass the notion that any record of an entity, in whichever form it is taken, can be considered a document. Later, the author applied these notions to the special case of a digital document, finding the definitions tied to format unsatisfactory, given the advancements in technology and its ever-changing nature, with evolving capabilities and constraints. There are two notions to take away from this paper: that there are "pragmatic definitions" that appear due practical needs, such as "a collection of data plus properties of that data that a user chooses to refer to as a logical unit", and that rather than attempt to define documents by their format, it is better to take "a functional approach", to reach a strict definition [Buc97b].

In 2009, Frohmann revisited Buckland’s question and questioned its pertinence, concluding that there is little reason to pursue the problem from a scientific point of view, as the function of Documentation is more important that possessing a strict definition for what it is [Fro09].

Following this line of thought, a more practical definition is given by Eloranta et al., where electronic documents are said to be containers for "written, drawn or dictated information in one separate and accessible source, which can be addressed by the proprietary applications used to generate it" [EHL01].

Perhaps the simplest definition of all, and an often cited one, comes from Levien in 1989, when they define a document as "recorded information structured for human comprehension" [Lev91]. This establishes that documents are different from data because they are structured in such a way that humans easily find meaning (i.e., extract knowledge) in them.
2.1.2 Document Management in an organizational context

As information technologies evolved in the latter half of the 20th century, businesses sought them out to solve problems and improve business processes, as a way to increase productivity while optimizing resources, creating business value.

Sprague [Spr95] outlined the challenges and opportunities of EDM for organizations and in doing so, divided the value that Document Management can create for a business in two categories: one where the document is (part of) the product and the other where all organizations are included. The first category is found in the industry of software in the example of reference manual’s, where they support the product being sold, while not being what is sold directly. The second category pertains to the role that EDM can have in information management, where it has an impact in three fronts:

- Transitioning data records to document format, enabling better storage and communication of concepts/ideas
- Converting business processes tied to document format (instead of data records) from paper to digital
- Indexing organizational memory

Implementing EDM in a company also comes with its challenges. Two major ones are the cost of conversion from physical formats and the quantification of benefits [MS96]. While still applicable nowadays, the first challenge is less pronounced nowadays due to technological advancements and a smaller reliance on paper. The challenge of measuring benefits is still a reality today, as the restructuring of processes to accommodate for the DMS is vital to its successful implementation.

2.2 Ontologies

Although there have been many definitions given to ontologies throughout the years, the most descriptive one might be that “an ontology is a formal, explicit specification of a shared conceptualization” [SBF98]. The term first originated in Philosophy as a branch of metaphysics to analyze and describe the universe, beings and the relationships between them. It was adapted into Information Science (IS) in the 1980s by the Artificial Intelligence (AI) community as a term for “a theory of a modeled world and a component of knowledge systems” [Gru07].

In the early 1990s, Neches et al. foresaw a break in the capability of the intelligent knowledge management systems of the time to handle the growing scale of knowledge acquired, due to the difficulty of modeling complex knowledge in a structured fashion. The knowledge bases had to be purpose built for each new system, creating a very costly overhead. They proposed a modular, reusable structure for knowledge representation that would not only allow for flexible and extensible systems to be created and updated, but also enable the sharing of knowledge between intelligent systems and conventional software. This structure is “a top-level declarative abstraction [hierarchy]” and is called an ontology when “defines the basic terms and relations comprising
the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary". [NFF+91]

A first technical definition of ontology in the context of Computer Science came about in 1995, when Gruber described it as “an explicit specification of a conceptualization” [Gru95]. He also proposed five design criteria (clarity, coherence, extendibility, minimal encoding bias and minimal ontological commitment) for ontologies that are still present in the field today.

Towards the end of the 1990s, a growing need to structure the development of ontologies was made explicit by Mizoguchi and Ikeda [MI98]. In this report, they proposed a field called “Ontology Engineering”, which is the study of the methods and methodologies of building ontologies. In this landmark paper, they provide comprehensive definitions of various key concepts in the field, as well as roles, a ranking of depth in the use of ontologies, their typology, a proposed scope for Ontology Engineering and some examples of research done in the field. The purpose stated for the field is "to provide a basis of building models of all things in which computer science is interested".

Nowadays, ontologies can be described as formal models that collect the vocabulary terms for a domain and are comprised of classes, properties and relations between entities, in such a way that the result is interpretable by both people and machines and ideally represents an entire knowledge domain, enabling knowledge reuse and sharing. They are already used for the representation of large bodies of knowledge and are a powerful tool employed in several fields, including numerous ones in AI (knowledge representation or natural language processing, for example), the Semantic Web, Software Engineering, Systems Engineering, to name a few.

2.2.1 Related concepts and applications

2.2.1.1 Taxonomy

Taxonomies first appeared thousands of years ago, primarily as a tool for botanical classification, and have since been closely associated with the field of Biology [Man10]. In a broader sense, it can be considered simply the science of classification, applicable to multiple fields of knowledge.

While there is no widespread consensus on the distinction between taxonomy and ontology, what is generally accepted is that while an ontology is a complete and complex representation of an entire domain, with meaningful semantic representation, taxonomies are considered to be narrower in scope and easier to navigate and understand, from a human perspective; the meaning of relations between concepts is not explicitly defined but usually adheres to Is-a or Has-a types of relationships, representing class and subclass hierarchies. [HS04] This entails that taxonomies may be (and frequently are) present as a subset of an ontology, whenever there is one or multiple hierarchical relationships.

2.2.1.2 Semantic Web

Currently, one of the most popular applications of ontologies in Information Science is in the building of the Semantic Web. In [BLHL01], Berners-Lee et al. detail a preliminary vision for the
Semantic Web, a machine-focused Web, where information is document-oriented and has semantic meaning. By defining a vocabulary with links and associating to those inference rules, agents would be able to turn knowledge that is implicit and normally inferred by humans into explicit knowledge, as well as enabling the communication of knowledge between agents. In essence, the Semantic Web is a structured and machine-interpretable version of the Web. This vision for the evolution of the Web is based on ontologies as its form of knowledge representation and has propelled research on them in recent times.

2.2.1.3 Linked Data

Linked Data is an essential underlying concept in the Semantic Web and it refers to "using the Web to create typed links between data from different sources" [BHBL09]. The concept sits on the Linked Data principles, a set of four rules devised by Tim Berners-Lee to guide the publishing of data in the Semantic Web [BL06].

Currently, the biggest Linked Data project is DBpedia, a structured version of data extracted from Wikipedia. DBpedia is acting as the center of the Semantic Web, due to an effort on the part of the community around it to connect DBpedia to other open data sources [ABK +07]. This can be consulted in the The Linking Open Data (LOD) cloud diagram project website, at http://lod-cloud.net (last accessed 2017-06-14).

2.2.2 Resource Description Framework (RDF)

RDF, the Resource Description Framework, is the data modeling framework standard, developed by the World Wide Web Consortium (W3C) and used to describe concepts using web resources, by making statements called triples (Figure 2.1), in graph form. It is similar to an EAV or entity-attribute-value model, seeing as how a triple is comprised of a subject, a property and an object, in this order. A possible conceptual mapping of EAV to RDF is entity to object, attribute to property and value to subject. RDF is independent of the particular syntax used to describe a domain and is therefore considered an “abstract syntax”.

![Figure 2.1: RDF graph triple (adapted from [RDF14a])](image)
Background

A statement represents a relationship between two entities, the subject and the object. The property is a predicate which describes the nature of the relationship between the two entities. Entities in RDF can be of three different types: IRI, literal or blank node.

IRI (Internationalized Resource Identifier) is an extension of the URI, or Uniform Resource Identifier, which is itself a string of ASCII characters that identifies a resource. IRIs generalize URIs by enabling the usage of Unicode (Universal Coded Character Set), as defined in ISO/IEC 10646:2014 [DS05]. They are absolute and are the only type of entity that can be present in every position of a triple, as shown in 2.1.

Literals are used to represent basic values such as strings, dates, numbers, and booleans. They are typically comprised of two parts, the lexical form and the datatype IRI. The lexical form is a Unicode string that indicates how the literal is presented, while the datatype IRI points to the datatype that the literal is in and maps the lexical form to the literal value.

Blank nodes, also known as anonymous resources, are not represented by IRIs or literals and are used to denote entities without identifying them. [RDF14a]

2.2.3 Ontology languages

There are several languages that can be used to describe ontologies, many of which extend RDF semantically. Some of the earlier syntaxes developed were simple, not RDF-based and markup-based (XML). These include Ontology Exchange Language (XOL), Ontology Markup Language (OML) and Simple HTML Ontology Extension (SHOE) [GPC02]. Of the languages built on RDF to provide a foundational vocabulary, the most relevant are RDF Schema (RDFS) and its child languages, the Ontology Interchange Language (OIL) and DARPA Agent Markup Language+OIL (DAML+OIL), which spawned the OWL Web Ontology Language (OWL).

Alongside SHOE, there are languages that extend HTML (HyperText Markup Language), to provide webpages with semantic meaning. Examples of these languages are Microdata [HTM13] and RDFa [RDF15].

2.2.3.1 RDF Schema (RDFS)

While RDF provides the structure to model an ontology, a specific schema, such as RDFS, is required to define the modeled concepts semantically. RDFS provides a basic vocabulary with which to describe things.

The basic elements of RDFS are classes and properties, with members of a class being called instances. All classes are a subclass of rdfs:Resource, which is an instance of rdfs:Class. Major classes include rdfs:Literal, rdfs:Datatype and rdfs:Property, all of which correspond to concepts detailed in the RDF model. [RDF14c]

2.2.3.2 OWL Web Ontology Language

As described in the OWL 2 Primer, OWL 2 "[…]is an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and
data values and are stored as Semantic Web documents. OWL 2 ontologies can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents.” [HKP+] OWL was designed with the goal of allowing computers to test ontology knowledge for its consistency and making assertions based on it. A side effect of this objective is that the language becomes less interpretable by humans. OWL operates on 3 notions: axioms (logical statements), entities and expressions (combinations of entities).

OWL 2 comes in two (with OWL Lite remaining a valid OWL 2 ontology, but being deprecated) semantic variants: OWL 2 DL and OWL 2 Full. The former employs a Description Logic style [BCM+10] and is the least expressive and the latter makes use of the RDF-Based Semantics used by RDFS and is the most expressive of the two.

OWL 2 introduced three usage profiles that represent a subset of the language, each targeted at a different use case, making it simpler to implement. They are OWL 2 EL, OWL 2 RL and OWL 2 QL. According to Suárez-Figueroa et al., OWL 2 EL is best suited for large ontologies, OWL 2 QL for relational query access and OWL 2 RL for RDF triple storage operations. [SFGCVTGP11]

OWL is built on RDFS, meaning that every RDFS description is also an OWL description. OWL proper offers a broader vocabulary and stricter rules than RDFS.

2.2.3.3 RDF serialization

Serialization is the process of representing data structures and models in storage formats, to be restored at a later time, in any system. This process must be applied to RDF graphs in order for the data represented within them to be queryable. The reverse process is performed by an RDF parser, utilizing a mechanism such as Gleaning Resource Descriptions from Dialects of Languages (GRDDL) [Con07] for XML-based descriptions.

Some of the standardized (or relevant, submitted for standardization) serialization syntaxes are:

- RDF/XML: XML-based W3C standard [RDF14b] and the foundational RDF representation format (example in listing 2.1)

- JSON-LD: JSON-based W3C standard [JSO14]

- Notation3 (N3): Multi-featured simplification of RDF/XML, also capable of serialization (submitted for Recommendation to W3C) [BLC11]

- Turtle family: Plain text, human-friendly syntax, subset of N3 (example in listing 2.2, taken from [SPA13])
  - Turtle [TUR14]
  - N-Triples and N-Quads: An even simpler subset of Turtle [N-T14]
  - TriG: Extends base Turtle to support named graphs [TRI14]
2.2.3.4 SPARQL Query Language

SPARQL (SPARQL Protocol and RDF Query Language [Bec11]) is a database query language for the RDF graph structure. SPARQL is compatible with languages that are built on RDF, such as RDFS and OWL. It is the de facto standard query language for such structures [HBS09].

A simple example is as follows, first presented in [SPA13]:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE {
    ?x foaf:name ?name .
    ?x foaf:mbox ?mbox
}
```

Listing 2.3: Simple SPARQL query for people with both name and mailbox (data in Turtle listing)

---

1https://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/, last accessed 2017-06-14
The **PREFIX** declaration abbreviates an IRI reference. The **FROM** term defines the dataset to be queried. The **SELECT** term identifies the result variables, which are represented with a "?" prefix. The **WHERE** term precedes the query definition. Many more terms are available, including sorting terms such as **ORDER BY**.

### 2.2.4 Ontology engineering methodologies

In [CFLGP03], Corcho et al. compare a body of methodologies available at the time, referring that many of them were not supported by any tool, with the exception of On-To-Knowledge and METHONTOLOGY.

According to Iqbal et al., there are many different methodologies available but not much in the way of an accepted standard, since many of these emerged from accounts of specific projects. They performed a literature review with the criterion of type of development, collaborative construction, reusability support, degree of application dependency, life cycle recommendation, strategies for identifying concepts and methodology details. [IMMS13]

A notable absentee from this review is SABiO (Systematic Approach for Building Ontologies), an iterative model with four major stages, beginning with a domain analysis and requirements elicitation, a formalization of the terms gathered into a reference ontology, the design of an operational ontology in a specific language, and the implementation of the operational ontology using the design specification and the reference ontology. [DAF14]

[SFGCVTGP11] mention that "METHONTOLOGY, On-To-Knowledge, and DILIGENT were up to 2009 the most referred methodologies for building ontologies". The NeOn Methodology for Building Ontology Networks [SFGPFL10] is a more recent effort to create a flexible and broad methodology that covers set of processes covers and nine different development scenarios and is one of the most mature methodologies available. Of the nine scenarios detailed, this work is best framed by the second scenario, ""Reusing and re-engineering non-ontological resources".

Noy and McGuinness [NM01], one of the most oft-cited articles on the topic of ontology methodologies, aims at providing a step-by-step primer on the process of ontology development, from inception to realization. The recommended steps are:

0. Determine the domain and scope (point 0 because it is an activity which is essential but independent of ontology development)

(a) Consider reusing existing ontologies

1. Enumerate important terms

<table>
<thead>
<tr>
<th>name</th>
<th>mbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Johnny Lee Outlaw&quot;</td>
<td><a href="mailto:jlaw@example.com">mailto:jlaw@example.com</a></td>
</tr>
<tr>
<td>&quot;Peter Goodguy&quot;</td>
<td><a href="mailto:peter@example.com">mailto:peter@example.com</a></td>
</tr>
</tbody>
</table>

Table 2.1: SPARQL example query results
Background

2. Define the classes and the class hierarchy
3. Define the properties of classes - slots
4. Define the facets (restrictions) of the slots
5. Create instances

This article is especially relevant for this work as it makes use of Protégé for practical examples.

2.2.5 Ontology engineering tools

The tools for the practice of ontology engineering can be categorized in a variety of ways since there are many different activities across the process, from design to reasoning, instancing, and storage, to name a few. The range of tools available includes those targeted at a specific task and suites of varying scope. In [SFGCVTGP11], a relatively recent and comprehensive collection of tools is presented. An even more recent survey on development tools and reasoners has been conducted in [KP17].

![Ontology tool categorization (SFGCVTGP11)](image)

Figure 2.2: Ontology tool categorization ([SFGCVTGP11])

The most widespread tool for the development of ontologies is an integrated development environment (IDE) called Protégé [FGA14], developed by the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine, currently in its fifth major version. [MPT15] This IDE provides support for graphical ontology design, built-in reasoning, class tree navigation, graph visualization and instancing/populating, while being compliant with
Background

W3C standards. Ontologies may be exported in various formats, including OWL, RDFS, RDFS-XML.

There are two versions, desktop and web, the latter of which allows for collaborative ontology editing. The desktop version is highly customizable and extensible, with a sizable library of plugins. It is free, open source (under a modified BSD license), actively developed.

2.2.5.1 Triple store

A graph database is a generic type of store that is geared towards graph models and can be of several different types, such as Web-oriented, Document-oriented or RDF-based, for example. [Ang12] The latter, commonly known as triple store, are systems that provide storage, parsing, serializing and reasoning of information stored as a triple graph.

In Angles’ 2012 comparison of database graph models, he found "that most graph database models provide an innate support for different graph structures, query facilities in the form of APIs (most of the models) and query languages (a few of them), and basic notions of integrity constraints‘. The databases analyzed to reach these conclusions were AllegroGraph, DEX, Filament, G-Store, HyperGraphDB, InfiniteGraph, Neo4j, Sones and vertexDB.

A more recent review of RDF store solutions was conducted in [MST14], wherein solutions capable of handling over 2 billion triples and in active development at the time were analyzed according to a set of technical characteristic license, operating system, implementation language, repository mechanism, client connector languages, query languages and latest release data, and compared according to their ability to handle streaming data and binary data, security, and versioning. The solutions were AllegroGraph, Bigdata(R), Apache Marmotta, OpenLink Virtuoso, Oracle 12c and Stardog.

OpenLink Virtuoso was the database system suggested by the proponent to be used as the RDF store component for this project, as described in section 3.2.1 Virtuoso is a system that collects multiple database and data management functionalities, including relational database management, web and file server, RDF graph database, among others. There is a commercial version and an open source version, the former of which has two additional features, virtual database engine (to enable queries across multiple databases) and a data replication functionality. The management interface for this software is called Conductor and allows for data importation, schema loading, document management, and more.

The technical architecture of Virtuoso can be seen in figure 2.3. The relevant components to this project include the RDF Relational Property Graphs, HTTP Web Service, Open Database Connectivity (ODBC) and SPARQL database access and .NET Runtime.

2.2.6 Existing documentation ontologies

As described in section 1.2, this work focuses on modeling the documentation specific to Finantech and filling a gap in their information systems. Although some works were found in the field of DM ontologies, the amount of specificities in this case study made it impractical to follow or
adapt existing models. Many of these works focus on annotating the content or structure of the documents, and are thus distanced from our requirements, which are for metadata pertaining to the management of the documents. Regardless, previous experiences did provide insight on what questions would arise throughout the work.

The Ontology for Media Resources \cite{CSS12} (OMR) is an ontology aimed at creating a common vocabulary for media resources, taking several disperse definitions and mapping them cohesively, and is mostly aimed at Web resources. One of the ontologies OMR is mapped to is Dublin Core \cite{Dub}, a colloquial designation for the Dublin Core Metadata Initiative Metadata Terms specification, a recommendation for a standardized metadata structure which describes both digital and physical media resources. In 2001, the specification was represented in RDF format. In a similar vein, FaBiO, the FRBR-aligned Bibliographic Ontology \cite{PS12} is an ontology for bibliographic references of publishable records which covers textual publications, contents of said publications and intangible records such as "blogs, web pages, datasets, computer algorithms" and more.

\footnote{Retrieved from https://virtuoso.openlinksw.com/virtuoso-technical-architecture/, last accessed 2017-06-06}
Background

In [Kli06], a review of the literature in document management systems and ontologies is conducted, in an effort to evaluate an ontology-based approach to e-government in the German state of Schleswig-Holstein. The author points out that the needs and processes of public administration in this domain are different from those of businesses.

In [Cor06], Corcho takes a look at trends in document annotation using ontologies, and comes upon two different approaches: the generation of metadata about the document, and the structuring and annotation of the content of the document. The author also takes a look at the relevant languages and tools available at the time and identifies a few open problems in the field, namely the conversion of the unstructured web to a semantic one, and questions of quality control, management, versioning, storage, etc.

In [MBSD00], an ontology-driven approach was undertaken to create a "document enrichment" system that allowed to associate creation context and artifact linking information to documents. This allowed them to answer questions built directly (in a form) utilizing the concepts defined in the ontology and extract knowledge from annotations.

In [Eri07b], documents and ontologies are combined to create so-called "semantic documents", i.e., documents enriched with annotations or structured by ontologies that detail common elements such as title, pages, sections, etc. Three ontologies are presented: annotation, document and domain, in order to achieve separation of concerns and layer the modeling of different aspects of the system. Eriksson also demonstrated the use of Protégé as a document repository for Portable Document Format (PDF) files containing semantic annotations, which provided robust search functionalities but faced scalability issues with the increase in number of documents [Eri07a]. Other examples of research into semantic documents includes works by Nešić [Neš09] and Nessah and Kazar [NK12].

The Open Document Format for Office Applications, an XML-based format for digital files, relies on an extensive ontology to model its basic fields, file types, structure, graphic content, etc. [Ope11].

Ontologies have also been employed in a number of particular knowledge domains and use cases:

- HL7 Clinical Document Architecture, a standard for the structure and semantics of documents in healthcare [DAB+01];

- re-use of teaching materials by extracting ontologies from document metadata and relating them with the concepts that constitute the teaching strategy [KH];

- digitization of a library of Bulgarian folk music and an information retrieval tool for song lyrics [NPP11]

Straying away from strictly document-related ontologies, [WAK+07] and [BCF+08] are two works in knowledge extraction from documents and existing materials on the web, offering structure to fragments of information from various sources and enabling semantic querying. These
are just two examples of work in this field, which is closely related to the domain of this work, although it does not encompass our requirements.

2.2.7 Summary

An ontology is a conceptual model that formalizes a given knowledge domain. The representation of entities and relationships that is made possible by ontologies is of the utmost usefulness and potential in fields such as AI, Knowledge Management and the Semantic Web, with Linked Data. There have been many developments in terms of purpose-built languages to translate the theoretic structures to machine-readable syntax, but the standards defined by the W3C are the preferred way to go about employing ontologies in real-world applications and the publication of rich, semantic data. RDF, RDFS, SPARQL, and the many syntaxes available for RDF are some of these standards.

In terms of methodologies, no single one has been adopted by the research community or business world as the standard, who seemingly prefer to follow a more organic approach, adjusted to the specific project. The most prominent methodology that can potentially be adopted is the NeOn Methodology, given its broad set of project scenarios.

Relative to tools, the most relevant development in design and editing is the Protégé IDE. Triple store solutions, however, are aplenty. A highlight goes to OpenLink Virtuoso for being the solution chosen by the DBpedia project [EM09].

Many related works were mentioned, in a non-comprehensive manner, to give a picture of what has been done and what is possible to do for metadata creation and management by making use of ontologies. These provided some insight in terms of approaches to take in the design of the model and implementation of the system. In spite of this, no model was found which could be repurposed for Finantech’s needs. Additionally, as mentioned in the Introduction (section 1.1), part of the company’s motivation for this project was the acquisition of knowledge and lessons on ontologies, a goal answered in part with this Background and further with the practical component of the project.
This chapter is devoted to the detailing of the problems at hand and the solutions created for them. Its name is a reference to the designation that the potential DMS solution within Finantech was given, "Agidoc". Section 3.1 looks at what exactly were the problems found by the company and during the project itself, and section 3.2 looks in detail at the solution found, component by component.

3.1 Problem Analysis

As mentioned in section 1.2, Finantech has a need for an indexing and search solution regarding their internal documentation. This documentation is saved in many different formats, including Microsoft Word .doc/.docx and similar, .pdf, spreadsheets, text files, and more. They were stored in a shared network folder, email, and employees’ machines’ file systems, creating a situation where multiple copies of the same file exist in different places, versions branch out from different parents, making it difficult to know which version of a document anyone is working with or viewing. There was no connection between documents/files to any data in the company’s databases (hereafter loosely mentioned as "Agifox DB", as they provide storage for Finantech’s internal project management system, Agifox), meaning that relevant relationships between the documents and the actors and activities they are about were not easily discoverable and retrievable.

Across the meetings conducted with the company, a few key requirements were determined by looking at what their needs were and, per their initiative, some existing solutions in the problem space. These informal requirements were:

- The system must be easy to use
- The system must force users to catalog the files they input into it
• Users without administrative privileges must not be able to access files in any way other than through the system
• The system must only allow access to Finantech employees, on the company’s machines, with no connection to the Internet
• The system must employ ontologies and make use of OpenLink Virtuoso

These requirements follow the assumption that a culture is developed where the system is of mandatory use and were ultimately appraised by the proponent in a loose fashion.

While not strictly referring to document management, Fitsilis et al. give some foundation to the employment of ontologies in software development, stating that "ontology development could facilitate or improve substantially the software development process through the improvement of knowledge management, the increase of software and artifacts reusability, and the establishment of internal consistency within project management processes of various phases of software life cycle" [FGA14]. Further support for this concept is mentioned in section 2.2.6, namely in Klischewski’s work [Kli06] and the various applications mentioned, in other domains. This gives strength to the use of ontologies in this project, despite the fact that this route would be pursued regardless of prior efforts in the field.

3.2 Solution

Approaching the problem presented by Finantech without constraints would allow for a vast number of solutions. Using existing frameworks and middleware to build a new system or implementing an already developed and robust solution, the gap in document storage and retrieval could be solved with not much effort on the technical side, leaving modeling and organizational process changes as the main tasks. However, constraints were placed on the solution that were a product of additional goals for this project which made the solution specific in its technical aspects.

As mentioned in 1.1, aside from the primary objective of developing a solution for the document management problem of Finantech, the project had the goal of exploring ontologies as a concept and as a practical tool, to later use the lessons learned and acquired experience in evaluating the possibility of using FIBO in future Finantech products and services. This meant that a significant portion of time would be allocated to learning how exactly ontologies could be used for our purposes and how to use the existing tools both for the development of the ontology and the database that stored it.

Taking these circumstances into consideration, the following steps were taken in implementing the solution, after the problem definition stage, and will be further developed in the next subsections:

• Architecture planning, including tool selection (sections 3.2.1 and 3.2.2)
• Exploration and learning of OpenLink Virtuoso, with tests for the mapping with the Agifox DB

• Domain analysis and definition (section 3.2.3)

• Ontology development, storage loading and final mapping with Agifox DB (section 3.2.4)

• Development of a web service responsible for querying the database (section 3.2.5)

• Development of an application which acts as the client for the system and is the point with which the users interact (section 3.2.6)

3.2.1 Solution Architecture

The architecture of the system (figure 3.1) was envisioned in collaboration with the company supervisor to meet their functional and technical requirements.

![System architecture diagram](image)

Figure 3.1: System architecture

The core of the information storage is in the Virtuoso server. Virtuoso is responsible for the mapping of the Agifox DB tables in Microsoft SQL Server to a quad graph structure (a graph of triple graphs). This mapping is done automatically by Virtuoso through its administration panel, Conductor. For the mapping to occur, one must establish an ODBC connection between the two systems and select which tables to read from the ODBC source (in this case, Agifox DB). After that, in the Linked Data > Views panel (fig. 3.2), the tables to be mapped are selected, and the user gets the option to generate a mapping immediately or use a wizard to, among other things, change which fields are primary keys and choose to map certain fields to ontologies other than the one that
Figure 3.2: Linked Data Views creation screen in Virtuoso Conductor

is going to be generated. Using the wizard and skipping any configuration, it was possible obtain
the mappings and most importantly, the generated ontology (fig. 3.3). This generated ontology
is most important as it represented the point of connection between the ontology written for the
domain model and the existing information structure at Finantech. Upon retrieving this definition,
the Linked Data Views were published and a new, virtual graph becomes available in Virtuoso.
This graph is virtual because the information is not replicated from the source database, and instead
the queries are interpreted from SPARQL to SQL and the information is read in real time using
ODBC.

The upper-left "Ontology" bubble in the architecture diagram (3.1) represents the ontology
that was created for the domain. Its loading into Virtuoso took place after the mapping described
above so that equivalences could be established between it and the ontology generated for the
Linked Data Views, thus connecting the two. The process of creating this ontology is detailed in
section 3.2.4.

With all of the mappings and data structures in place, the next component is the Web service.
This component is solely responsible for establishing a bridge between the storage components
and the application, abstracting the SPARQL queries made to Virtuoso. The reason for creating a
separate component for these tasks is that it makes the system modular and allows for the creation
of different clients in the future (mobile, for instance). The connection between the Web service
and Virtuoso is accomplished using the virtado4 Dynamic-link library (DLL) provided by Open-
Link, an ADO.NET library which leverages Microsoft’s .NET framework data access technology
ADO. The development of this component and the queries created within it are detailed in section
3.2.5.

Finally, connected to the Web service is the application itself, the client available to users
Figure 3.3: Linked Data views resulting definitions screen (partial) in Virtuoso Conductor

to manage and find documents. This component was produced with basic functionalities, as a prototype, in order to provide an easy-to-use graphical interface that mainly invokes the Web service’s methods to query the database.

3.2.2 Tools and technologies

On the ontology side, Protégé was chosen to develop the ontology itself. It provides a clear visual interface to create a class structure, the classes themselves and their relations, be it object relations with other classes or data relations with literal values, and allows exporting the ontology in several serialization syntaxes, most notably RDF/XML, Turtle, Manchester OWL, LaTeX and JSON-LD. Protégé is also highly extensible and has a wide range of addons available, such as different inference engines and graphical graph visualizers.

For ontology storage and querying, OpenLink Virtuoso was chosen. This choice came from the company, justified by Virtuoso’s support for RDF views and .NET drivers, as well as being one of the most prominent technologies in its category, with it being the technology used to power DBpedia. The RDF views (available only in the enterprise edition) are supported by a quad graph, i.e. a graph of graphs, of an automatically generated ontology, which is mapped to an external database via ODBC drivers, allowing for retrieval of information from a relational database management system (RDBMS) in real-time, through SPARQL queries to the quad graph, which are then translated to the flavor of Structured Query Language (SQL) that the target RDBMS understands.

Finantech’s products are mostly developed in Microsoft languages and technologies, for Microsoft systems, which includes their database systems in Microsoft SQL Server and a multitude
of apps developed within the .NET framework with C#. Most decisions on technologies to use on the web service and application side were based on this fact. The web service was developed using the ASP.NET Web API 2 library and the application was built in ASP.NET MVC 5, the latest libraries within the .NET environment for their respective purposes at the time of development and available in the Visual Studio 2015 IDE provided by the company for development.

### 3.2.3 Domain model

As one of the main goals of this work, the modeling of the domain was a continuous process of analysis and negotiation. The model encompasses some of the information present in Finantech’s databases, coupled with new classes representing the types of documents deemed most important for their business needs.

The central concept in this model is that of the Document. A Document is a container for both metadata and physical files (represented as the File class) and the unit that the user of the final system creates when they wish to archive a file from their local file system. The metadata it contains is a title, a description, a creation date, an author, a version number, a flag to identify if it is the latest version, an ID, a unique code, a group, a type and, depending on the combination of group and type, different properties. Optionally, the Document may also have a modification date and an editor. Besides this metadata, it must also contain one or more Files, which are all closely related and cannot exist outside of the scope of the Document. The Document’s metadata fields are defined in detail as follows:

- **Title**: the name of the Document
• **Description**: a detailed description of the Document

• **Creation date**: the date and time of the creation of the Document

• **Author**: the user who created the Document

• **ID**: a number unique to the Document, shared by all its versions

• **Version**: a number indicating the place of the Document on the version history of the Documents with the same ID

• **IsLatestVersion**: a boolean flag indicating whether or not the specific instance of a Document is its latest version

• **Code**: the unique identifier of a given version of a Document, composed of group, type, ID, and version (format in fig. 3.5)

• **Editor**: the user who modified the Document, present when the Document version is greater than 1

• **Modify date**: the date and time of the modification to the Document, essentially the creation date of this version of the Document, present when the Document version is greater than 1

• **Group**: the business area the Document belongs to. Possible groups are: Sales, Helpdesk, Scrum Teams, Administrative Services and Generic. Generic is a special case as it does not correspond to any business area, it is simply a way of making sure any document which can’t be categorized in one of the four main groups can still be introduced in the system. The different groups each have a set of possible types of documents, with the set for Generic being all the types.

• **Type**: the type of the Document. These are:

  • **Properties**: these properties are a category of metadata, pertaining to the relations a Document may have to the business-relevant entities coming from the Agifox DB. These are: Entity, Client, Family, Collaborator (also known as Employee within the application domain), Opportunities, Products and Proposals. Each type of Document in each group has different properties that it can and must have, except for Generic, which can contain any type with any property.

All of the different groups and types of Document were modeled separately, despite different groups sharing a certain type of Document. This was done to ensure that different attributes and relations could easily be added to classes within different groups, as the "same" type can have different attributes depending on the group to which it belongs to.

The File class is strictly a metadata container, as the physical file it corresponds to is stored in Finantech’s shared file system, nicknamed "Hermes". Its fields are:
Figure 3.5: Document code format

- **Title**: the name of the File
- **Address**: the address to the physical file in the shared network
- **Creation date**: the date and time of the creation of the File
- **Author**: the user who introduced the File in the system
- **ID**: a number unique to the File, shared by all its versions
- **Version**: a number indicating the place of this File on the version history of the Files with the same ID
- **IsLatestVersion**: a boolean flag indicating whether or not the specific instance of a File is its latest version
- **Code**: the unique identifier of a given version of a File, composed of ID, ID of the Document the File belongs to, and version (format in fig. 3.6).
- **Editor**: the user who checked-in this version of the File, present when the File version is greater than 1
- **Modify date**: the date and time of the version creation to the File, present when the File version is greater than 1
- **DocumentIRI**: the IRI of the Document the File belongs to, as it is generated in the database
- **DocumentID**: the ID of the Document the File belongs to
- **Extension**: the extension of the physical file
- **Size**: the size of the physical file in bytes

The Collection is a generic container for Documents with basic authoring metadata fields. They are meant to collect Documents which are not linked by any relationship, i.e. sharing properties, creation date or other metadata. A Project is a Collection with a corresponding list of Users (represented by their user name) and a flag to show whether or not the Project is active or closed.
The four classes mentioned above represent the basis of the domain’s model and are represented in figure 3.4. The Document children, representative of the different Groups and their respective possible types, are shown in figures 3.8 (Sales, with all associations many to 1 or more, just as other class diagrams), 3.9 (Helpdesk), 3.10 (Scrum Teams), 3.11 (Administrative Services) and 3.7 (Generic). All of these diagrams are in UML notation, with empty arrows denoting inheritance, straight lines denoting association, empty diamonds denoting aggregation and full diamonds denoting composition. All of the groups and types have a "descriptor", a fixed string code used to identify them, which are included the Document code format (see class diagram figures for descriptors). Additionally, all of the types are associated to one or more of the Agifox Properties, such as Family, Opportunity and Collaborator, with the multiplicity of these relations defining whether or not the Properties are mandatory or optional attributes of the Document Types.

The properties taken from the Agifox DB and connected to the Document domain, which were listed along with the metadata fields for Documents, can be summarized in five different classes: Family, WorkItem, Entity, Product and Employee. Families are the company’s designation for the different business units and their respective projects. They can be CRM processes with opportunities and requirements, projects, backlogs and more, depending on their need to allocate requirements, opportunities, features, etc. These items are known as WorkItems, of which the Proposal and Opportunity classes are children. Entities can be of many different types, and the Entity table in Agifox includes Finantech itself, clients, suppliers, and more. For the purposes of the Document types present in this work, the Client type of Entities is taken from this set. However, the Entity class, superclass of Client, is also modeled and its queries allow to consult all types of Entity, without distinction.

All of the Agifox DB properties can be seen in the class diagrams, connected to the relevant Document classes.

### 3.2.4 Ontology definition and storage

The specification of the ontology has been separated from the domain definition in the writing of this report to highlight the process of creating the Agidoc ontology and to explain some aspects of the employment of the relevant tools. However, the two stages were connected intrinsically, as the ontology is the domain model, with the only other modeling tools used being the class diagrams (some of which came after the definition of the ontology) and informal documents such
as spreadsheets, used to communicate with the company. In fact, the ontology acts solely as domain model, due to the closed nature of the work, meant to live exclusively within Finantech’s Intranet.

As mentioned in section 2.2.4, a valuable methodology is that of Ontology Development 101 [NM01] (Onto101). While the adaptation to the ontology development mindset was difficult, Onto101 was loosely followed and helped in guiding the process. Per its instructions, the first step was the determination of the domain and scope, which has been detailed in the previous section. As part of this step, however, is also the analysis of existing ontologies and the possibility of their reuse. For the project at hand, this step was somewhat overlooked, given that the creation of an ontology was, in and of itself, a goal. There were a few equivalences established, such as
between the FOAF definitions of name and the title for Documents/Files/Properties, FOAF Project and Agidoc Project and FOAF Document and Agidoc Document, as a test of this sort of linking. No further connections were pursued as there wouldn’t be much gain in doing it, given that the system is detached from the Internet and thus cannot integrate the Linked Web.

The following step, "Enumerate important terms", was executed in an ad-hoc fashion. The author’s defend building a list of terms with no regard to their connection or overlap, which was an activity conducted in meetings on the domain and scope, through dialog, with no materialization of such a list.

According to Noy and McGuinness, the next two steps (class/class hierarchy and slot definition) are "closely intertwined" and constitute the core and most important part of the ontology development process. Of a few cited approaches to tackling the task of defining classes, the one taken here was a top-down one, starting the most generic concepts, even if the class tree is at most 3 levels deep. The use of a top-down approach was mostly due to the concept of groups and types of documents not being present for nearly half of the duration of the project, which meant it was a flat structure for some time. This was the stage of ontology development where the classes and relations of classes present in the class diagrams of the previous section were defined, including...
those of the properties from Agifox DB. The view in Protégé utilized for this task is shown in figure 3.12, a class hierarchy building view based on the owl:Class and owl:subClassOf definitions. Slot restriction was done simultaneously with slot definition, as the domain was already defined and it was already known what ranges and domains attributes and relations would have. This involved choosing the domain (subject) and range (object) of the object properties and data properties. The former establish relations between different classes and the latter between a class and a literal value, as defined in the XML Schema Definition (XSD)\(^1\). Of note is the use of a subclass of the rdfs:label, a meta annotation property (owl:AnnotationProperty), to attach the document descriptors to their Document’s class definitions, and of the FOAF name property to define human readable names for groups and types of documents.

The next and final stage, creating class instances, was not strictly a part of ontology development, as this is part of the system’s functionality and is handled by Virtuoso, on answering requests from the Web Service. The only case of instance creation was in the definition of names for classes, as mentioned above.

The completed ontology was serialized in Turtle syntax and loaded onto Virtuoso with the DB.DBA.TTLP command through the isql Interactive SQL utility, which "parses TTL (TURTLE or N3 resource) and places its triples into DB.DBA.RDF_QUAD", according to the Virtuoso documentation\(^2\). A new graph was created for the Agidoc ontology within which all the definitions and information for the system was stored and queryable. As was quickly apparent, at least to our knowledge, there was no way to enforce the definitions which were loaded onto the graph, meaning that new definitions could be inserted with a simple SPARQL query. This did not constitute a vulnerability to the consistency of the structures in place because the Virtuoso server was abstracted via the Web Service and thus, not directly queryable. However, it is a major departure from the typical relational database model and something that one must always keep in mind while writing SPARQL queries. It also presents an opportunity which wasn’t explored in this work: the

---

\(^1\)https://www.w3.org/TR/xmlschema11-2/, available on 2017-06-17

\(^2\)http://docs.openlinksw.com/virtuoso/fn_ttlp/
possibility of dynamic definitions of attributes for classes or even new classes. A possible scenario where this could be useful is that introduction of new types of Documents into the system.

### 3.2.5 Web Service Queries

As seen in section 3.2.1, the system component responsible for handling queries to the storage component is the Web service. This component was developed using the Microsoft ASP.NET Web API 2 framework, which allows for a quick deployment and abstracts many tasks in creating a web service. It is comprised of several controllers that provide different routes and together create an application programming interface (API) that allows to retrieve, create, modify and delete resources from the Virtuoso graph for the documentation.

The Document controller is the largest and most complex of the HTTP API controllers created to contain queries to the graphs. There are multiple SELECT queries, as well as INSERTs and DELETE. The complete list of Document related queries is visible in table 3.1.
### Agidoc

<table>
<thead>
<tr>
<th>Name</th>
<th>Route</th>
<th>HTTP Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAllDocuments</td>
<td>api/documents/all</td>
<td>GET</td>
<td>Returns all versions of all documents</td>
</tr>
<tr>
<td>GetDocument</td>
<td>api/documents/{code}</td>
<td>GET</td>
<td>Returns a document given its unique code</td>
</tr>
<tr>
<td>GetDocumentHistory</td>
<td>api/documents/{code}/history</td>
<td>GET</td>
<td>Returns all versions of a document by its unique code, which it parses to retrieve its ID (as the ID is what connects all the document versions)</td>
</tr>
<tr>
<td>GetDocumentFiles</td>
<td>api/documents/{id}/files</td>
<td>GET</td>
<td>Returns all files of a document, given its ID</td>
</tr>
<tr>
<td>GetAllDocumentGroups</td>
<td>api/documents/groups</td>
<td>GET</td>
<td>Returns all groups, with their name and descriptor</td>
</tr>
<tr>
<td>GetAllDocumentTypes</td>
<td>api/documents/{group_d}/types</td>
<td>GET</td>
<td>Returns all types of a given group, given its descriptor</td>
</tr>
<tr>
<td>GetSpecificDocuments</td>
<td>api/documents/list</td>
<td>GET</td>
<td>Returns list of Documents, given a list of integer IDs</td>
</tr>
<tr>
<td>CreateNewDocument</td>
<td>api/documents/new</td>
<td>POST</td>
<td>Takes a Document object and creates a Document in the graph</td>
</tr>
<tr>
<td>EditDocument</td>
<td>api/documents/{code}</td>
<td>PUT</td>
<td>Takes a Document object and edits it in the graph, given its code, by deleting the previous entry and generating a new one</td>
</tr>
<tr>
<td>DeleteDocument</td>
<td>api/documents/{code}</td>
<td>DELETE</td>
<td>Deletes a Document and all its Files, given its code</td>
</tr>
</tbody>
</table>

Table 3.1: Document-related queries

---


2. SELECT ?type ?group FROM <https://idawebdev/TesteOntologia/agidoc> WHERE {{ iri } agidoc:hasFamily|agidoc:hasEntity|agidoc:hasProcess|agidoc:hasClient|agidoc:hasProposal|agidoc:hasOpportunity|agidoc:hasProduct ?group . {{ iri } ?type ?group}

Listing 3.1: GetDocument SPARQL query

To show some examples, starting with a SELECT, the GetDocument method is split into two queries, one for the Document’s own fields and another for every property related to the Document (code listing 3.1). In this query, the "iri" parameter is constructed previously using the code of the Document received in the HTTP GET request. Of note is the "agidoc:" prefix, which appears in most queries. This prefix is defined in the Namespace section of Virtuoso Conductor and is thus globally available to every query performed to the server. The prefix "agd:" may also appear, as this is the prefix defined for the graph Virtuoso generated for the mapping to the Agifox DB.

The CreateNewDocument query (code listing 3.2) was created as a member of the Document model, as it is constructed dynamically based on the attributes that have been provided in the HTTP request body. Most of the metadata is mandatory and expected, however, the properties are often optional or do not apply to certain types of Documents, hence the conditional clauses in the construction of this query. The method was condensed (where "[...]" is present) for the sake of brevity.
string query = $"sparql INSERT DATA {{GRAPH <{ GlobalStrings.AgidocGraph }> {{
  agidoc:{ code } a agidoc:Document . agidoc:{ code } agidoc:hasCreateDate "{ creationDate.ToString("yyyy'-'MM'-'dd'T'HH':'mm':'ss'.'fff") }" . agidoc:{ code } agidoc:hasTitle "{ title }" . agidoc:{ code } agidoc:hasAuthor "{ author }" . agidoc:{ code } agidoc:hasID { id } . agidoc:{ code } agidoc:hasVersion { version } . agidoc:{ code } agidoc:isLatestVersion 1 . ";

if (familyIris != null) {
  foreach (string fam in familyIris) {
    query += $"agidoc:{ code } agidoc:hasFamily <{ fam }> . <{ fam }> agidoc:isFamilyOf agidoc:{ code } . ";
  }
}

if (modifyDate.Year > 1) {
  query += $"agidoc:{ code } agidoc:hasModifyDate "{ modifyDate.ToString("yyyy'-''MM'-'dd'T'HH':'mm':'ss'.'fff") }" . ";
}
query += "}};

return query;

---

Listing 3.2: CreateNewDocument SPARQL query

The DeleteDocument query starts by retrieving a list of the codes of Files belonging to the Document in question, in order to delete the physical files from the shared network. After this retrieval, the deletion query is executed and, upon its success, the physical files are effectively removed. This last query is represented in code listing 3.3.


Listing 3.3: DeleteDocument SPARQL query

The Files controller shares similar functionality with the Document controller and its API can be seen in table 3.2.

In the smaller controllers, there is the Collection controller, which houses the API for Collections and Projects and includes methods for creating, deleting and retrieving of these classes. There is also the User controller, which simply allows to retrieve lists of Documents and Collections of a User, given their user name as it is defined in their Windows session and the company’s Intranet.

The Search controller contains one method, which allows for a basic text search over the object part of all triple relations on the graph. The SPARQL query used for this purpose is shown in code listing 3.4. This query translates to "The subject s is related to object o by predicate p, the object o contains string q and does not start with the IRI of the document graph and the Virtuoso generated
<table>
<thead>
<tr>
<th>Name</th>
<th>Route</th>
<th>HTTP Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetFileVersions</td>
<td>api/files/{id}/history</td>
<td>GET</td>
<td>Returns all versions of a File, given its ID</td>
</tr>
<tr>
<td>GetFileVersion</td>
<td>api/files/{code}</td>
<td>GET</td>
<td>Returns a single version of a File, given its code</td>
</tr>
<tr>
<td>GetFileExtension</td>
<td>api/files/{code}/extension</td>
<td>GET</td>
<td>Returns a File's extension, given its ID</td>
</tr>
<tr>
<td>GetFileDocumentCodeById</td>
<td>api/files/{id}/document</td>
<td>GET</td>
<td>Returns the code of a File's Document, given the File's ID</td>
</tr>
<tr>
<td>CreateNewFile</td>
<td>api/files/new</td>
<td>POST</td>
<td>Takes a File object and edits it in the graph, given its code, by deleting the previous entry and generating a new one</td>
</tr>
<tr>
<td>EditFile</td>
<td>api/files/{code}</td>
<td>PUT</td>
<td>Takes a File object and edits it in the graph</td>
</tr>
<tr>
<td>DeleteFile</td>
<td>api/files/{id}</td>
<td>DELETE</td>
<td>Deletes all versions of a File, by its ID</td>
</tr>
<tr>
<td>UploadPhysicalFile</td>
<td>api/files/upload</td>
<td>POST</td>
<td>Takes a file from a multipart-data request and places it in the Hermes shared network</td>
</tr>
</tbody>
</table>

Table 3.2: File-related queries

Graph, and the subject s starts with the IRI of the document graph", which essentially means that only relations where the subject belongs to the document graph and where the object is not an IRI will be returned when their object contains the query string "q". The usage of regular expressions with the "i" modifier for case insensitiveness came about due to issues with Virtuoso’s handling of UTF-8 encoding through ODBC that were beyond the scope of this work to solve, which made the use of the built-in LCASE and UCASE functions for string manipulation impossible, thus restricting the results to case matching ones. With this issue, string results fed to the functions mentioned returned as a "?" symbol.

```
```

Listing 3.4: Search SPARQL query

Continuing on to the properties controllers, most of them share similar queries that follow a certain pattern. These queries are all SELECTs because they all retrieve data from the Agifox DB, through Virtuoso. They are built in two parts: the first query involves retrieving the predicates of triple relations from the Agifox DB ontology in the document graph, by establishing equivalences with the document ontology using the owl:equivalentClass predicate, and the second query retrieves the relevant information from the Agifox DB through Virtuoso, using the predicates obtained in the first part. This could be done by directly querying the graph automatically generated by Virtuoso, but the premise is that one should not need to know that mapping ontology and should resort to the definitions in the Agidoc ontology. The properties controllers are: Client, Collaborator, Entity, Family, Product and WorkItem (which contains methods for Opportunities and Proposals). The main format for queries for all of these classes are as follows:

- **SimpleList**: returns a condensed list of the most relevant attributes for every instance of a given class.
• *GetX*: returns all of the attributes of a single instance of a class, by its ID. X stands for the type of property, for example GetFamily or GetClient.

• *GetXDocs*: returns all documents of an instance of class X, identified by its ID.

### 3.2.6 Application

The application component is a web-based client developed in Microsoft ASP.NET MVC 5 which allows the user to interact with the system. It is a prototype with basic Document and File creation, edition and deletion functionality, as well as pages that display query results. The page’s hierarchy, the Properties pages collapsed into its category, is shown in figure 3.13. This section gives an overview of the organization of this client and explains its functionality. Note: some pages were trimmed to better display their content.

![Application Sitemap](image)

**Figure 3.13: Application Sitemap**

Common elements to all pages are the top navigation bar (navbar), which provides links to most pages on the website, and a left sidebar, for user-specific information and pages. All of the pages shown in the upper level (dark yellow section) of figure 3.13 are accessible through the navbar.

The home page (fig. 3.14) presents the user with all the navigation they might require and additional views on Documents in the center of the page, two of which are not available elsewhere, with information relative to the logged-in user. First, there is a panel for quick access to the user’s (at most) 5 most recently created Documents. The second panel lists up to 5 Documents most recently accessed by the user. The third panel shows the user’s 5 most viewed Documents. A view is counted when the user accessed the Document’s page or highlights the Document in a view with a Document table, such as any of the Property pages.

Clicking on any of the Documents in this page (as well as other pages that list Documents) leads to the page of that version of the Document (fig. 3.15). Usually, pages show the latest version of all Documents, but it is possible to access other versions as well, given their codes. This page
Figure 3.14: Application Home page

Figure 3.15: Document application page
shows all of the metadata fields of that Document version, as well as all of the Files belonging to the Document, with the option to checkout or checkin any of them. There are also links to show the Document version history and to edit it. The version history information is a modal with a very similar structure to the one showed in figure 3.17, excluding the checkout buttons.

Also available from any page is the "New Document" button, which launches a modal containing a form to create a new Document (fig. 3.16). The form is dynamic, meaning that it swaps out fields depending on the Group and Type the user selects in the dropdown menus. The user must fill at least the Title, Description and the mandatory Properties for the Type they’ve selected, as well as add a File. This is done to prevent users from creating empty Documents and not filling in metadata.

Documents may be edited by users in order to change their title, description and properties, which will create a new version of the Document. The new version will have the same ID number, the editor who performed the changes and the date in which they were made. With this, the ID number connects all versions of the same Document, while each individual version is uniquely identified by its code.

The individual Files within a Document can be "checked-out" by a user in order to edit them, which will lock the file from being accessed by other users (see fig. 3.15). When the work is done, the user who retrieved the File will have to check it back into the system, which will generate a new version of the File’s metadata, with a modification date, an editor, incremented version number, updated file size, new address, with the remaining fields remaining the same as
Every new version of a File is attached to the head of the version history, meaning that, even if the user checks out an earlier version of the File, the version created upon check-in will be at the top, linked linearly to the history at the previous recent version. The version history is visible in the File’s own page (fig. 3.17). The code is unique and thus also changes. A new version of a File does not imply a new version of the Document. Every check-in or check-out is registered in one table in a small SQLite database. The current status of a file is determined by querying this table for the most recent entry which has the file’s ID. If a record’s "status" boolean field is set to "true", the file is checked-in and available for checkout; if "status" is false and "userName" is the same as the current user’s name, the file is in a checked-out state and available for checkin; if "status" is false and "userName" is different from the current user’s name, the file is in a "not available" state, meaning that someone else has checked it out.

As far as views on Documents go, there are several pages which correspond to different queries. The first one visible in the navbar is the Documents By Group page (fig. 3.18), shown in the navbar as "By Department". This change was made due to a difficulty in users understanding what the page was for before opening it. As Documents groups closely relate to the different departments at Finantech, this was a logical change that made it easily readable by the users. This page allows the user to view Document by group or by type within the group, filtering the list as the user makes changes to the dropdown selection boxes.

There is another view for filtering Documents called "By Date" (fig. 3.19). This page shows all of the years in which Documents were created, and by clicking on the year, the selection is narrowed down to only the months when a Document was created. Choosing a month displays a reverse chronological list of the Documents created in that month and year. This query was one
Figure 3.18: (Documents) By Department (or Group) application page

Figure 3.19: (Documents) By Date application page
of the earliest to be developed, at the request of the proponent, as they felt it was one of the most useful for their business needs.

Documents may also be filtered depending on the Agifox DB Properties they have. There are pages for every Property available, which all consist of a dropdown with a list of all Properties, a table for the Documents which relate to the selected Property and a panel, located on the right side of the page, which shows the details (metadata) for a Document selected from the table. Figure 3.20 shows an example of one of these pages, namely of the Client property, with the navbar dropdown showing all available Property pages.

Collections and Projects share similar pages, as Projects are extensions of Collections. They both have a page (such as the one in fig. 3.21) which displays all of the items and allows the user to create new ones, and each item has its own page (fig. 3.22), with the Collection/Project description, author and creation date, as well as the list of Documents it contains and, in the case of Projects, the list of users associated to it.

The user has access to a basic profile page from the left sidebar. The page lists all of the user’s created Documents, the Projects the user is assigned to and the files that are currently checked-out by the user.

Finally, there is a rudimentary Search page for displaying the results of a text search, detailed above in the Queries section (3.2.5). It displays a table of triples where the query string is matched to the object of relations, visible on the third column.

The aforementioned SQLite database (fig. 3.23) is located at the application level, not the web service level, and was created to store simple records of information that changes frequently. Besides the change management data, this database also stores data on document views by users. This was designed to accommodate any type of view, as the table also stores the type of object
Figure 3.21: Projects application page

Figure 3.22: Single Project application page

41
that was viewed. Currently, this is only used for views of Documents in the application (choosing object type "0").

As mentioned in 3.2.3, the physical files are stored in a directory in a shared network nicknamed "Hermes". All permissions on this directory are reserved to administrators and the application itself, so as to prevent employees from directly accessing the files and producing new, untracked versions, or even removing the stored data. Upon entry in the system, all files get renamed to their corresponding File code and are stripped of extension, and are stored in a folder structure which mimics the class diagram, i.e. the root folders are the Document groups, then the types, with the files therein. On retrieval from Hermes (when a checkout is performed), the file gets its extension re-attached to its name.

3.3 Summary

Finantech required a system that allowed them to store, organize and retrieve digital documents/files, with a focus on enforcing cataloging processes upon data introduction and retrieval solely through said system. They also had a significant interest in exploring ontologies and obtaining knowledge on them, specifically the possibility of application in programs as data models.

To answer these needs, a full-stack solution was envisioned, which encompasses data storage in a Virtuoso RDF graph, connected with the company’s own information systems, a web service to query the back-end and serve an API, and a prototypical client to navigate through the query results and introduce documents into the system. The development began with an analysis of the problem domain and creation of a model which characterizes Finantech’s document categorization and properties when connected to their existing data structures. Simultaneously, the specification of the domain was executed as an ontology, with the exploration of the Protégé tool. Following
this, Virtuoso was configured to receive data from the company’s information systems and loaded with the Agidoc ontology, creating with this process two graphs that encompass all of the data used by the system, with the exception of the physical files themselves, which were stored in a shared network folder with special permissions. With the back-end constructed, a web service built on the .NET framework was created to modularize and abstract queries to the data server, and a client was developed on the same framework to allow users to manage documents.

With this solution, the company has a single back-end for all Document data, which can be extended to link to more information on their other information systems, a centralized web service with an API capable of responding to many different types of clients, making the system modular and versatile, and a prototype of a client that connects to this API and demonstrates some of the document management capabilities of this system. Furthermore, they received insights onto the complexities and possibilities of using ontologies as tools to model data.
Chapter 4

Validation

Validation took place at the tail end of the project’s duration and involved company employees testing the prototype according to a predefined script and giving their opinion. This chapter details the reasoning for and the details of the validation methodology, followed by the results and their discussion.

4.1 Method

The main hurdle to testing the solution was the lack of an existing system or even standard processes. Each department, and sometimes individual employees, had their own way of handling files, which meant there was nothing concrete to compare the solution to. The option of comparing a relational database solution to Agidoc was considered and discarded, because it would imply implementing a new solution just to test the one contemplated in this work, or at the very least collecting empirical data, which did not seem feasible, since the information obtained from this would have no context.

Considering the goals set at the beginning and the hurdles described, the method chosen to validate the system was a set of interactive demonstrations, prefaced by questioning the participants on what they felt were the main Document Management issues within the company and followed by a short questionnaire on whether they felt the system was a better, worse or equal solution to what was done at the time, in each point gathered in the questionnaire. This would give an idea of whether or not the end users found the product to be useful and if it met their needs, which is, in my opinion, the most important factor to decide whether or not the tool is successful and worth implementing across the entire company.

With this method in mind, and given that the system was bound to the development machine due to deployment issues and the time-consuming nature of the method, the number of tests conducted was small and so the participants were mostly chosen from the same office room and based on their availability, with an effort to pick employees from different teams. In this particular
Validation

room, three teams operate, two of which are devoted to internal developments and another with designers.

The problems initially identified, which the participants would have to verify and, if they felt the problems applied to them or the company, rate were:

P1. Document duplication: multiple copies of the same file
P2. Misaligned versions: discrepancies in versions held by different employees
P3. Difficulty in finding documents
P4. Loss of documents: mistakes or errors resulting in permanent loss of files

Participants were first asked to confirm these problems (and point out different ones if they found them relevant) and, after the demonstration, rate on a scale of 1 to 5 (1 - Much worse, 2 - Worse, 3 - Same, 4 - Better, 5 - Much better) how they felt the system would compare to the way things were done at the time, on each point they had previously verified. The demonstration itself involved creating a Document, finding it within the system based on the various different views available, viewing its details and files, adding a file to it, editing the Document itself, checking files out and in of the system and viewing different lists in the Home and User pages.

4.2 Results

The participants in the validation process were volunteers from the office room where the work was developed and included a Product Owner, four Developers and one Designer, for a total of six employees. Every employee responded that they felt all four problems were present at Finantech.

After performing the demonstration for the participants, allowing for some interaction and feedback throughout (more on that in the Discussion, section 4.3), they were asked to perform the rating. The results of this are shown in table 4.1.

<table>
<thead>
<tr>
<th>Participant</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Owner</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Developer 1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Developer 2</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Developer 3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Developer 4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Designer</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

It is important to note that each test was conducted one-on-one, so as not to let the participants influence each other’s answers and perception of the system.
4.3 Discussion

Overall, the feedback was very positive. All of the participants expressed a great need for a Document Management system, as there is currently nothing implemented at the company. During the tests, one of the participants disclosed that they had worked on the same problem for a year, using Microsoft Sharepoint, but ultimately the company discarded that solution. This participant showed excitement about the functionalities of Agidoc and at the prospect of this work being further developed.

Participants were also asked for additional problems that they might feel are relevant to the domain. Of all those inquired, only the Product Owner identified one problem that was not contemplated — discrepancies in document formatting and language between different departments and even amongst employees, with such examples as some people writing in a very detailed style while others wrote in bullet points and Word documents from a certain department having one look and feel while another produced Word documents in plain text. This problem is beyond the scope of this work, despite being briefly discussed early on, and it remains to be seen whether the strongest cause is the lack of tools or the lack of process.

Developers 1, 2 and 3 pointed out P3 as the main problem out of all those listed. As document retrieval was one of the main focuses of this work, with multiple views on the documents available based on metadata, they felt the system tackled this issue the best and that focus was well placed.

Regarding the informal requirements identified in section 3.1, the participants felt the system was simple and clear, and during the interactions they had with it showed no confusion or trouble finding the information they were for or any pages, meeting the first requirement mentioned. In addition, the system does force users to insert the metadata for every document they create. It is not clear if there is any way to make users input relevant metadata other than process implementation and making adhesion to standards and rigor a strong part of the company’s culture. Another requirement was that users could only access files from the system, which is implemented, as the shared network folder where the physical files reside has special permissions that only permit administrators and the application to access it. Along with this, the system is prepared to only allow Finantech’s employees to access it, as it is not connected to the Internet, is behind the company’s network security and is tied to the user’s work machine account. This was not tested company wide. Finally, fulfilling the final requirement, both ontologies and OpenLink Virtuoso were successfully used to bring the solution to life.

From the ratings attributed to the issues, it is possible to conclude that the potential users of Agidoc found it to be, on average, much better than what is currently done at Finantech in terms of Document Management. The most successful and useful features are, according to the participants, finding documents and the prevention of loss of documents. These results indicate that the outcome of this project was positive, in the potential end users point of view.
Validation
Chapter 5

Conclusions

The overall outcome of this project is positive, in my view. Lessons were learned and the model and framework delivered were well received by Finantech. This chapter offers some reflections on the project as a whole and lists some specific points for future developments of the final product.

5.1 Future Work

During the implementation of the system and upon evaluation, several future improvements for this system were determined. Some came about relatively early in development but weren’t considered, given the prototypical nature of the system and time constraints. The following list gathers some of the points of improvement for the developed system, noted throughout development.

- **User class**
  At the time of the analysis of Finantech’s information systems, the question arose of how to handle authentication. The method the company practices in some of their systems (authentication in Microsoft SQL Server Management Studio for access to database schemas and direct querying, for example) is of Windows authentication, i.e. every employee has an account in the Intranet and a corresponding profile on their work machine, with logging in to individual programs occurring automatically. Such an approach does not require the reading of any tables with authentication information and simplified the creation of the prototype application.

  However, this decision restricted the ability of creating user profiles linked to the Agifox DB (which also houses employee and employee as user information), with accompanying features such as private collections and a favorite document flagging system.

- **Team class**
  The above point also impacted the ability to create the concept of Teams in the system, which would allow the enrichment of the Project class and improve traceability and assignment of
Conclusions

documents, extending the platform further into Content Management and making it more helpful with Project Management.

- **Complete view history and edit options**
  View history is currently in a very limited state. It could be extended with clear and sorting features.

- **Physical file versioning**
  The versioning capabilities of the system are restricted to the metadata on Documents and Files. The possibility of integrating a file versioning library such as git, which would allow for better traceability and version comparison, was discussed with the project proponents but ultimately discarded for the time being, due to time constraints.

- **Workflows**
  The ability to integrate business processes in the system and have documents go through a sequence of stages until they are somehow considered approved or finalized, as well as assign tasks on the documents to employees, is an appealing extension of Document Management for Finantech and a logical next step in the solution’s development.

- **Read/write permissions**
  An important feature of a Document Management System, left out for time constraints as it was deemed non-essential for this prototype.

- **File previews in the application**
  The final product of this project allows users to "check out” files to read or write to them. However, there is currently no way to only read a file when there is no intention altering its contents. Being able to preview files within the application would not only be a fast way to execute this task, there are also not predictable interferences of the implementation of this feature with the current checkout system.

- **Elasticsearch / Docker**
  The implementation of Elasticsearch for search indexing to improve scalability and the deployment of the Virtuoso server as a Docker container was suggested within the company as technical improvements, which were briefly considered but deemed out of scope for this work.

- **Temporary files in local file system**
  Suggested by a developer at Finantech, the checkout system could create a temporary file in the user’s local file system, which would be removed when checking the file back into the system, avoiding the cluttering of their workspace with multiple versions of the same file.

All of these improvements are listed under the assumption that they apply to the system as it was developed and described in chapter 3.
Conclusions

5.2 Discussion

A few conclusions can be drawn about the project in general, in terms of the difficulties faced, lessons learned and the final product. The major difficulties were in learning how to use some of the ontology-related tools to our purposes, especially when the time came to connect the web service to the back-end. There were also some setbacks in the gathering of the necessary information and structures for the domain model specification, where the time it took to get a response from the company was non-insignificant. These challenges were the ones that took the most time away from development of the final product.

Some specific points are made about different aspects of the work in the next subsections.

5.2.1 On ontologies

The use of ontologies in this project was a fundamental goal. However, it turned out to be superficial, mostly due to the learning curve on the concepts and the tools, and a focus on "using ontologies" instead of the use that the ontologies would have, such as some Artificial Intelligence functionality, for example. Instead, they were used merely as a data model, which proved to be unproductive for delivering a complete DMS. However, the lessons learned on ontologies, theoretical and practical, are of great importance to Finantech in future endeavors in analyzing and eventually implementing FIBO into its products. The nature of these products, often related to trading and other financial activities which involve information exchange), is a perfect match for the linking capabilities and shared terminology achieved with ontologies, which constitutes a potential competitive advantage in a crowded business area.

5.2.2 On the system

The system’s architecture proved to be successful in answering the demands of the company and in being future-proof, as far as creating different document management clients and supporting a great amount of data goes. It is perhaps overqualified for the latter, with reports of Virtuoso being capable of handling triples in the order of the hundreds of millions, at least [EM09, MLANN11]. It is unclear whether Finantech will implement this system as is, due to the fact that the use of Virtuoso for their purposes would require a paid license, which they have said to be less of an obstacle for internal use, but a tough sell to their own clients. Therefore, an investment in this exact architecture is harder to justify.

Despite its prototypical nature, the client developed already met many of the company’s needs according to the validation conducted, and shows many avenues for future improvement.

5.2.3 On the domain model

The domain model proved to be one of the strongest contributions for Finantech, as the project progressed. Beyond not having any system or process in place to handle documents, they did not even have a concise and organized vision of what exactly the documents they deal with are,
Conclusions

across the various teams and departments. Even if the developed solution doesn’t receive further
development and implementation, the model may certainly be repurposed for any number of other
solutions and for the development of processes.

5.3 Summary

The goals set at the outset of this project were primarily to create a Document Management system
that filled a gap in Finantech’s information systems and to explore ontologies and their use in
software development. A working system was delivered, one that made use of ontologies to model
the information it is meant to store and organize. The final product was well received by the
proponent of the work and the potential end users, and the goals have been considered met, with
space for future improvement.
References


REFERENCES


REFERENCES


[Dub] Dublin core metadata initiative (dcmi).


REFERENCES


REFERENCES


REFERENCES


REFERENCES


