From File Folders to Knowledge Graphs : An Automatic and Configurable Symbiotic Approach

Emmanouil Smyrnakis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Yannis Tzitzikas*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

From File Folders to Knowledge Graphs : An Automatic and Configurable Symbiotic Approach

Thesis submitted by
Emmanouil Smyrnakis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author:	Emmanouil Smyrnakis
Committee approvals:	Yannis Tzitzikas Professor, Thesis Supervisor
	Dimitris Plexousakis Professor, Committee Member
	Kostas Magoutis Professor, Committee Member
Departmental approval:	Polyvios Pratikakis Professor, Director of Graduate Studies

Heraklion, March 2024

From File Folders to Knowledge Graphs : An Automatic and Configurable Symbiotic Approach

Abstract

We daily all use the file system of our operating system to organize our files of any kind (from documents to data and applications), both plain users and IT professionals. The tree-structured and semantics-neutral approach of file systems is the dominant method for organizing information, decades now. In this paper we elaborate on the following two questions: (a) can a file system structure be benefited by a Knowledge Graph (KG), (b) can the construction of a KG be facilitated by the file system? We elaborate on these questions and then we propose an automatic method for producing KGs from folder structures. The method can be configured through small, and easy to write, configuration files that can be placed in the desired folders to guide the KG construction. We present FS2KG, an implementation of a proof-of-concept prototype that includes a file explorer (enriched with KG-related information). Then we present empirical and experimental results, as well as a task-based evaluation with users. In brief, the approach can facilitate the rapid creation of KGs, as well as various file system related tasks. Beyond the empirical validation, we include a discussion of potential advancements and applications.

Από Φακέλους Αρχείων σε Γνωσιακούς Γράφους: Μια Αυτόματη και Διαμορφώσιμη Συμβιωτική Προσέγγιση

Περίληψη

Όλοι μας καθημερινά χρησιμοποιούμε το σύστημα αρχείων του λειτουργικού μας συστήματος για να οργανώσουμε αρχεία οποιουδήποτε τύπου (από αρχεία δεδομένων έως αρχεία λογισμικού) τόσο απλοί χρήστες όσο και επαγγελματίες της πληροφορικής. Η ιεραρχικά δομημένη και σημασιολογικά ουδέτερη προσέγγιση των συστημάτων αρχείων είναι η κυρίαρχη μέθοδος για την οργάνωση πληροφοριών εδώ και αρκετές δεκαετίες. Σε αυτή την μεταπτυχιακή εργασία, επικεντρωνόμαστε σε δύο ερωτήσεις: α) μπορεί ένα σύστημα αρχείων να επωφεληθεί από την ύπαρξη ενός γνωσιακού γράφου· β) μπορεί η δημιουργία ενος γνωσιακού γράφου να διευκολυνθεί από το σύστημα αρχείων· Συζητάμε αυτές τις δύο ερωτήσεις και έπειτα προτείνουμε μια αυτόματη μέθοδο για την παραγωγή γνωσιακών γράφων από δομές φακέλων. Η μέθοδος αυτή μπορεί να διαμορφωθεί μέσω μικρών και εύκολων στη δημιουργία, αρχείων ρυθμίσεων που μπορούν να τοποθετηθούν στους επιθυμητούς φακέλους για να οδηγήσουν την δημιουργία του γνωσιακού γράφου. Συγκεκριμένα, παρουσιάζουμε το FS2KG, ένα ερευνητικό πρωτότυπο που υλοποιεί αυτήν την ιδέα και περιλαμβάνει και έναν εξερευνητή αρχείων ο οποίος είναι εμπλουτισμένος με λειτουργίες που εδράζονται στο Γνωσιακό Γράφο. Κατόπιν παρουσιάζουμε κάποια εμπειρικά και πειραματικά αποτελέσματα καθώς και μια αξιολόγηση βάσει εργασιών με χρήστες. Επίσης περιγράφουμε μια πληθώρα από πιθανές επεκτάσεις τόσο στην προσέγγιση όσο και στην υλοποίηση.

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον κ. Γιάννη Τζίτζικα για την στήριξη και την καθοδήγηση τόσο σε προπτυχιακό επίπεδο όσο και καθόλη την διάρκεια της ακαδημαικής μου καριέρας μέχρι την ολοκλήρωση της εργασίας αυτής. Επίσης, θα ήθελα να εκφράσω τις ευχαριστίες μου στον κ. Δημήτρη Πλεξουσάκη και στον κ. Κώστα Μαγκούτη για την προθυμία τους να συμμετέχουν στην τριμελή επιτροπή. Ακόμη, ευχαριστώ το Τμήμα Επιστήμης Ύπολογιστων του Πανεπιστημίου Κρήτης για την αποδοχή στο μεταπτυχιακό πρόγραμμα και το Ινστιτούτο Πληροφορικής του ΙΤΕ για την υποτροφία που μου προσέφερε κατα τη διάρκεια της μεταπτυχιακής μου εργασίας. Τέλος, θα ήθελα να ευχαριστήσω την οικογενειά μου για τη συνεχή στήριξη και εμπιστοσύνη τους.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

All of us, either plain users or IT professionals, use daily the file system of our computer to create, update and manage our files. File systems offer a tree structure consisting of folders and files. The same structuring is offered by cloud-based file systems. This simple tree-structured and semantics-neutral approach of file systems is the dominant method with which we organize information decades now. The idea of using the term (and metaphor) folder for designing hierarchical file systems dates back to 1958 [2], while the first file system to support arbitrary hierarchies of directories was used in the Multics operating system [4], in 1965, half a century ago! We could say that the main benefits from the typical hierarchical organization of file systems is that: (a) it allows grouping resources (through folders with names and unlimited nesting level), (b) it allows naming resources relative to their parent folder, and (c) it allows moving/copying/deleting these resources in one shot, i.e. all contained resources are moved/copied/deleted. However, traditional file systems typically follow a hierarchical folder structure. While this is organized, it can be limiting for scenarios that require more complex relationships or multiple categorizations. Another weakness of this structuring method is that (a) each resource (file or folder) should be placed (and appears) in one place. The "shortcuts" that file systems typically offer is a remedy, but it is quite weak (one way links; not bidirectional). Consequently, file systems do not support a multi-faceted approach for locating resources (e.g. if a user has a folder about conferences attended, and inside each conference folder has a subfolder with photos from that conference, then the user cannot get all such conference photos easily). This thesis is inspired by the demo description presented in [27]. In comparison to that paper, this thesis (a) explains the motivation, (b) includes the related work, (c) provides more examples and the full specification of the configuration language, (d) showcases the implementation of a file explorer that is enriched with KG-related functionality, and (e) presents the results of task-based evaluation with users.

1.1 Research Questions

Two questions that arise are:
($Q1$) since Knowledge Graphs (KG) are labeled graphs, and not trees, could this extra expressiveness be leveraged for the contents of our file system?, and
($Q2$) since there is a need for practical and effective methods for producing KG, as automatically as possible, could the ubiquitous use (and knowledge of using) file systems, be leveraged for speeding up the creation of KGs?

Both directions could have significant impact. The first would enable leveraging the Semantic Web technologies in every day tasks. The second would assist the creation of KGs, something desirable, since there is a need for practical and mature tools to foster knowledge engineering (there are some critiques about the practicality and availability of tools for the Semantic Web, e.g. see [28], and an elaborated discussion of these critiques at [11]).

Indeed, although there are several successful applications of semantic technologies for background knowledge (e.g. DBpedia [1]), for collaborative knowledge creation (e.g. Wikidata [29]), for building domain specific semantic repositories that aggregate data from several cultural sources (e.g., Europeana [10]), marine sources (e.g. GRSF [16]), historical sources (e.g. [14], [8]) and recently covid-19 sources (e.g. [18, 26]), a successful application for our daily activities is still missing.

1.2 Challenges

Enriching a file system with a KG is a challenging task, since a file system contains very heterogeneous material since it is used for various purposes and tasks. For instance, one part of the file system may contain training material (books, papers, slides, assignments, student exercises), another part various personal material (family documents, photos and videos, travel information), software code and systems and others. Also, achieving semantic interoperability between the file system's native structure and the ontological representations in the KG requires careful mapping and alignment to ensure meaningful relationships. Someone could also say that as file systems are dynamic, and their content can change frequently, keeping the KG updated to reflect these changes in real-time requires efficient synchronization mechanisms. Finally, as the volume of data within the file system grows, maintaining a scalable KG becomes challenging. Efficient methods for handling large-scale data and ensuring quick query response times are essential. For surveys related to how users actually use the file system see [6, 5].

1.3 Approach

In this paper we elaborate on questions $Q1$ and $Q2$, and we propose a method for constructing KGs from the file system with emphasis on automation and flexibility, i.e. the approach does not restrict the freedom of changing the structure of our file system. The method supports small configuration files that can be placed in the desired folders to guide the KG construction at that folder. In brief, the key contributions are: (a) the introduction of a method for automatically creating KGs from file systems that is based on a modular and easy-to-use configuration, (b) an implementation of the approach (configuration language and tool `FS2KG`), (c) empirical and experimental results from applying this approach on real file systems.

We propose supporting two fundamental interrelated aspects: *folder structure* and *semantic network* with connections between these two. The core schema is illustrated in Figure 1.1 where with "*" we denote multiplicity (as in UML Class Diagrams). The approach is equipped with methods that create entities based on the files and folders of the file system, as well as by extracting them from csv files.

In comparison to the line of research under the term "semantic desktop", we could say that the current work has a more modest, but realistic, vision: not to integrate data, applications, and tasks, but to focus on the data part (folders and files). The proposed approach is more tightly related with the classical file system usage. It adopts a modular configuration approach, there is no dependency to a central repository, or central configuration, or any other service.

The big picture is sketched in Figure 1.2.

1.4 Implementation

In this paper we will present a proof-of-concept prototype that stands as a Java-based file explorer application. FS2KG seamlessly integrates traditional file management functionalities with advanced semantic technologies. Through user configuration and the utilization of a knowledge graph, the application facilitates semantic querying, resulting in an intelligent, context-aware tool for users. This integration offers a unified interface for efficient file navigation coupled with enhanced contextual understanding. By harmoniously combining robust file management capabilities with advanced semantic functionalities, FS2KG empowers users to uncover valuable insights and relationships within their data. The result is a comprehensive and user-friendly tool that transcends traditional file explorers by providing a deeper understanding of data through semantic integration.

1.5 Thesis Structure

This thesis is organized as follows: Chapter 1 introduces the topic of this thesis, the research questions that need to be answered, the challenges enriching a file system with a KG and the approach we suggest to tackle this issue. Chapter 2 includes key findings and methodologies from previous studies, laying the groundwork for the innovative contributions of the present research. Chapter 3 focuses on the the methodology and techniques employed in the research, providing a detailed rationale for the selection of the proposed approach. Chapter 4 dives into practical aspects of the research, detailing the implementation of the proposed approach and suggests some possible extensions. Chapter 5 is dedicated to the evaluation of the proposed approach, assessing the validity of the findings and the effectiveness of the current implementation. Finally, Chapter 6 summarizes the main contributions of the research, emphasizing their significance while some potential areas for future research are suggested, providing a launching pad for subsequent investigations.

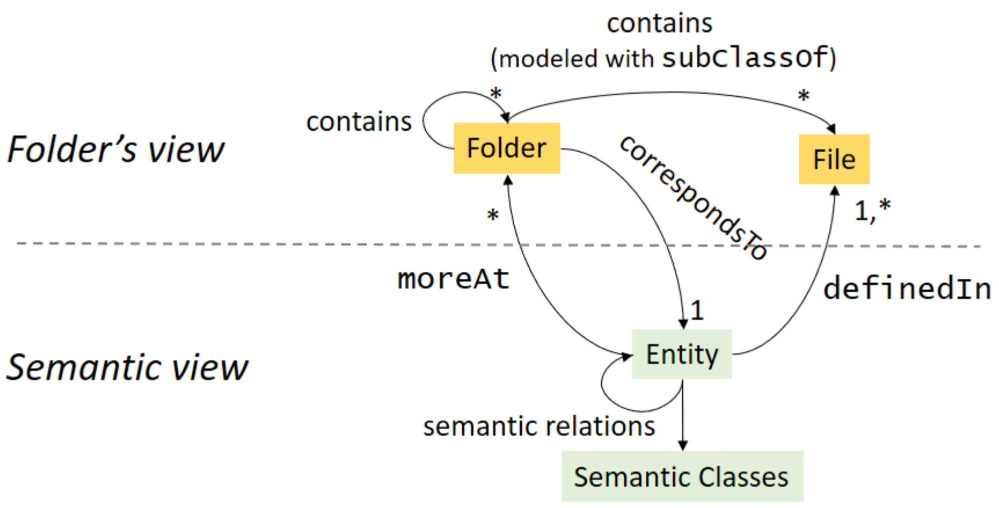


Figure 1.1: The core connections

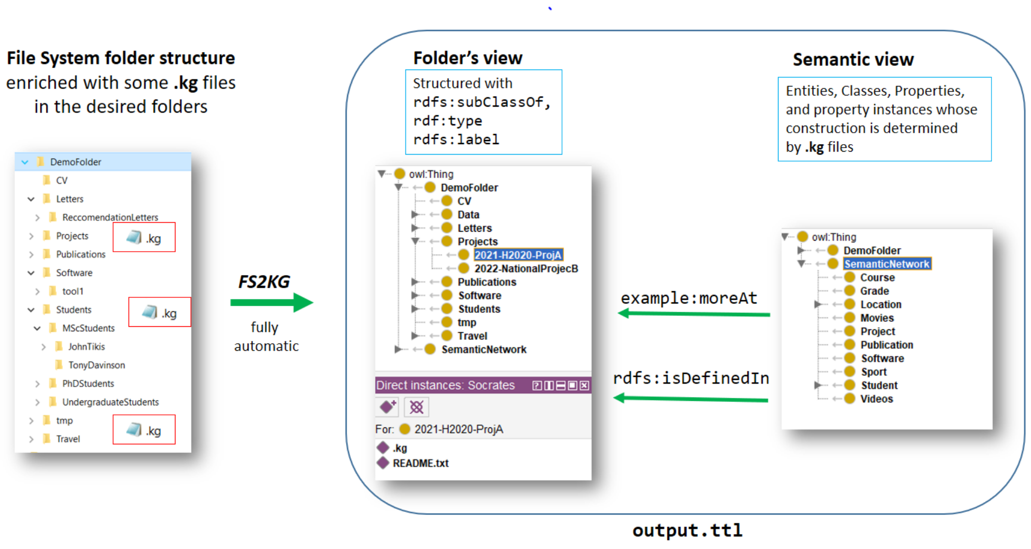


Figure 1.2: The big picture

Chapter 2

Background and Related Work

2.1 Background: Semantic Web

The Semantic Web is an extension of the World Wide Web that aims to make online information more meaningful and interconnected. It was conceptualized by Sir Tim Berners-Lee , the inventor of the World Wide Web. The fundamental idea behind the Semantic Web is to add a layer of semantics, or meaning, to the existing web content, allowing machines to understand and process information more intelligently. In traditional web environments, information is primarily designed for human consumption, and computers struggle to interpret the context and relationships between different pieces of data. The Semantic Web addresses this limitation by providing a set of technologies and standards that enable the creation of structured and linked data. Key technologies include the Resource Description Framework (RDF), the Web Ontology Language (OWL) and the SPARQL Protocol and RDF Query Language (SPARQL). By utilizing these technologies, the Semantic Web aims to create a network of interconnected and semantically annotated data. This enhanced level of comprehension opens up new possibilities for automated reasoning, knowledge discovery, and more effective data integration across diverse sources. Ultimately, the vision of the Semantic Web is to enable more intelligent applications, improved information retrieval, and seamless interoperability between different systems, leading to a more efficient and powerful web experience.

2.1.1 RDF

RDF 2.3 allows information to be expressed in a machine-readable format by describing resources and the relationships between them. Key features include its triple-based structure (subject-predicate-object), enabling the expression of relationships between resources. RDF creates interconnected graphs of data, allowing resources in one triple to link to others, forming a web of linked data. Resources are identified by Uniform Resource Identifiers (URIs), ensuring global uniqueness. Overall, RDF forms the foundation of the Semantic Web, enabling the structured

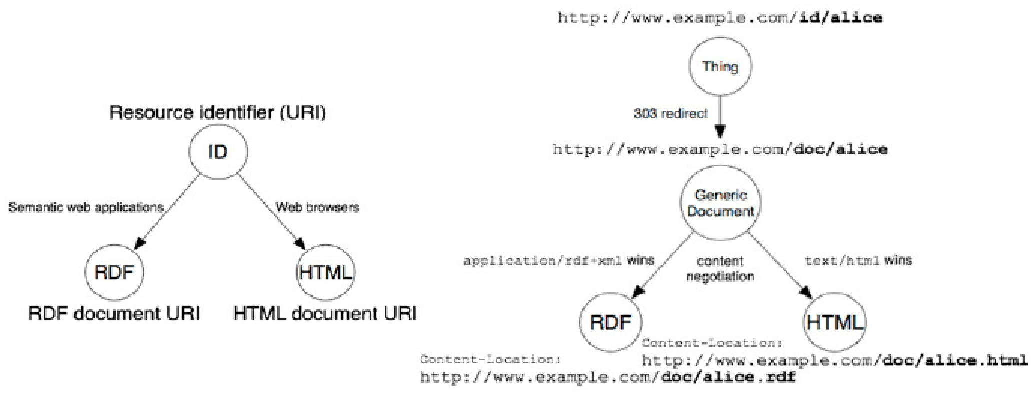

Resource identifier (URI)
ID
Semantic web applications
Web browsers
RDF
HTML
RDF document URI
HTML document URI
http://www.example.com/id/alice
Thing
303 redirect
http://www.example.com/doc/alice
Generic Document
application/rdf+xml wins
content negotiation
text/html wins
RDF
HTML
Content-Location: http://www.example.com/doc/alice.rdf
Content-Location: http://www.example.com/doc/alice.html

Figure 2.1: Linked-data principles

representation, querying, and discovery of knowledge.

2.1.2 Triplestore

A triplestore is a specialized type of database designed for the storage, retrieval, and management of data represented in the Resource Description Framework (RDF). RDF is a model for representing information in the form of triples, which consist of subject-predicate-object statements. These triples form a graph-like structure that captures relationships between resources.

2.1.3 Linked Data and URI

Linked Data 2.1 is a concept and set of best practices for publishing, connecting, and interlinking structured data on the web in a way that allows for seamless and meaningful exploration by both humans and machines. Linked Data follows a set of principles to enable the creation of a global network of interlinked datasets, contributing to the vision of a more interconnected and accessible web of data. A key principle in Linked Data is **URI**, or Uniform Resource Identifier, is a string of characters that provides a unique and standardized way to identify a resource on the internet. URIs are used to identify and locate resources, which can be anything from web pages, documents, and images to services and specific parts of a document.

2.1.4 Ontology

An ontology 2.2 is a formal and explicit representation of the knowledge within a specific domain. It defines the concepts, entities, relationships, and rules that characterize that domain, providing a shared understanding and a structured framework for organizing information. Ontologies are used to capture the semantics of a particular domain, making it possible for both humans and machines to understand and interpret information consistently.Key components include classes,

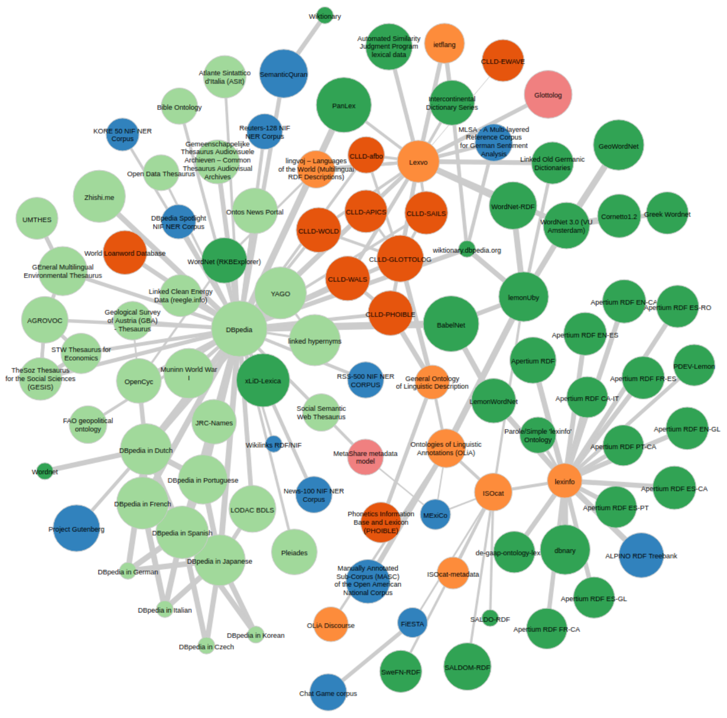


Figure 2.2: Example ontology

properties, instances, axioms, and a hierarchical structure. Also, ontologies enable semantic interoperability by providing a shared and unambiguous understanding of terms and relationships. This allows different systems and applications to exchange information and communicate effectively.

2.1.5 OWL

OWL, or Web Ontology Language 2.3, is a language designed to represent and reason about knowledge in the context of the Semantic Web. It is a W3C (World Wide Web Consortium) standard and is used to create ontologies, which are formalized representations of knowledge in a specific domain. OWL provides a way to express relationships, constraints, and classes, allowing for the creation of rich and complex semantic models.

2.1.6 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is a key component of the Semantic Web stack, designed specifically for querying and manipulating data expressed in the Resource Description Framework (RDF). The Semantic Web aims to enhance the meaning and interconnectivity of data on the web, and SPARQL plays a crucial role in enabling users to retrieve and work with this semantically enriched information.

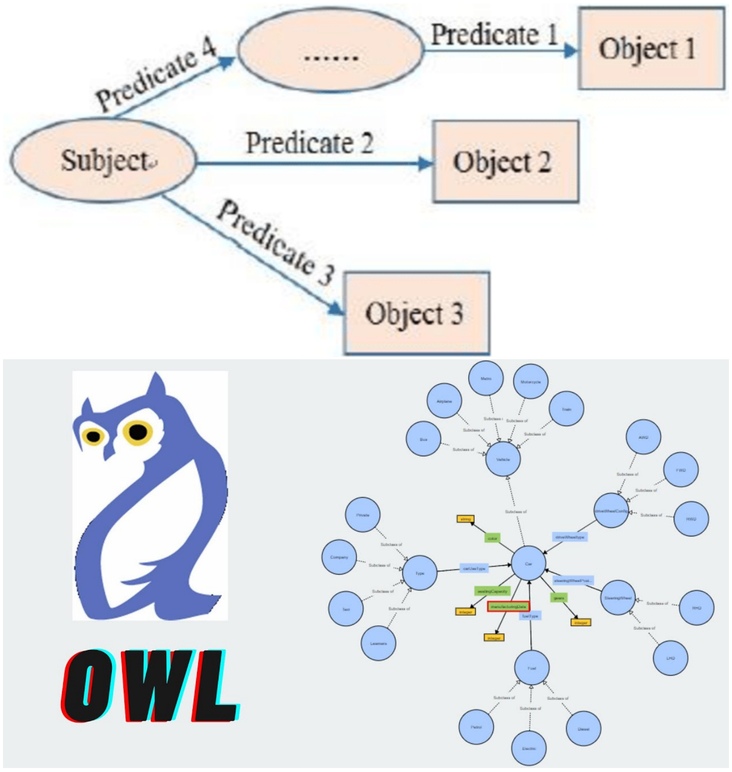


Figure 2.3: RDF model and Web Ontology Language (OWL)

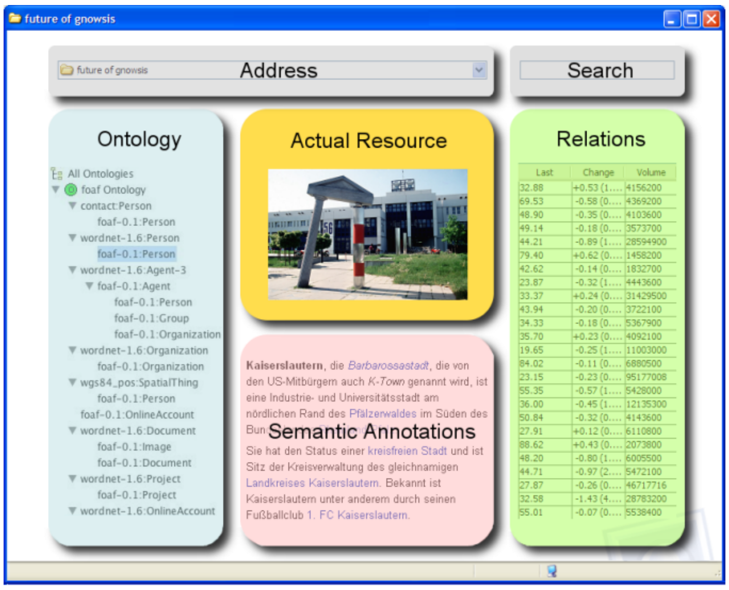
2.2 Related Work

The main input for producing linked data are *structured data* (csv files and databases) and *texts* (over which information extraction is applied). To use directly the folder structure of a file system as input, to the best of our knowledge has not been explored adequately.

There are a few works related to the notion, or vision, of *semantic desktop* that was developed 15 years ago. However, as pointed out in [13], existing Semantic Desktops are either too complicated, or not scale well, and a real "killer app" is still missing. In general we could say that this line of research has not flourished yet, since it has not affected the way we use our desktop. A brief discussion such works, and others, (in chronological order) follows.

Back in 1999, [12] proposed the automatic generation of metadata, expressed in RDF, for improved resource discovery. This paper describes how an automatic classifier, that classifies HTML documents according to Dewey Decimal Classification, can be used to extract context sensitive metadata which is then represented using RDF. At the time the paper was written the inclusion of of RDF meta information was not so encouraging. Following, [20] gave an overview of the vision of 'semantic desktop', the year of 2005, see 2.4. This paper analyzes existing systems and proposes two software architecture paradigms, one for the Semantic Desktop at large and another for applications running on a Semantic Desktop. A view on the context aspect of the Semantic Desktop and the Knowledge Management aspect is given.

In 2006, [23] points the problem with traditional file systems that do not provide sufficient means for organizing and annotating directories based on ontology-based classification schemes, especially when multiple users access the same file inventory. On that basis, the *SemDav* project is described, that aimed at enriching the storage of filesystems with semantic capabilities. Another attempt is *SemDesk*, based on a Personal Information Model (PIMO) [22] (2007). PIMO is used to represent a single users' concepts, such as projects, tasks, contacts, organizations, allowing files, e-mails, and other resources of interest to the user to be categorized. This categorization using multiple criteria was used to integrate information across different applications and file formats. Based on RDF/S, multiple layers were defined: an upper-layer for a minimal set of generic concepts, a mid-layer for refinements, and a user-layer for concepts of the individual user. The previous attempt was accompanied by a user study conducted later in 2008 [21]. Using the open source software prototype Gnowsis, they evaluated the approach in a two month case study which concluded in that simple has-Part and is-related relations are sufficient for users to file and re-find information, and that the personal semantic wiki was used creatively to note information. In 2006, [3] analyzed the different semantics between strictly hierarchical and tagging-based organisation, and proposed a mapping of non-hierarchical tagging and query semantics to the commonly used hierarchical file system semantics, for combining the benefits of both worlds. TagFS, their WebDAV-based implementation allowed



Kaiserslautern, die *Barbarossastadt*, die von den US-Mitbürgern auch *K-Town* genannt wird, ist eine Industrie- und Universitätsstadt am nördlichen Rand des Pfälzerwaldes im Süden des Bun

Sie hat den Status einer kreisfreien Stadt und ist Sitz der Kreisverwaltung des gleichnamigen Landkreises Kaiserslautern. Bekannt ist Kaiserslautern unter anderem durch seinen Fußballclub 1. FC Kaiserslautern.

Last	Change	Volume
32.88	+0.53 (1....	4156200
69.53	-0.58 (0....	4369200
48.90	-0.35 (0....	4103600
49.14	-0.18 (0....	3573700
44.21	-0.89 (1....	28594900
79.40	+0.62 (0....	1458200
42.62	-0.14 (0....	1832700
23.87	-0.32 (1....	4443600
33.37	+0.24 (0....	31429500
43.94	-0.20 (0....	3722100
34.33	-0.18 (0....	5367900
35.70	+0.23 (0....	4092100
19.65	-0.25 (1....	11003000
84.02	-0.11 (0....	6880500
23.15	-0.23 (0....	95177008
55.35	-0.57 (1....	5428000
36.00	-0.45 (1....	12135300
50.84	-0.32 (0....	4143600
27.91	+0.12 (0....	6110800
88.62	+0.43 (0....	2073800
48.20	-0.80 (1....	6005500
44.71	-0.97 (2....	5472100
27.87	-0.26 (0....	46717716
32.58	-1.43 (4....	28783200
55.01	-0.07 (0....	5538400

Figure 2.4: A semantic desktop example (added from [20])

users to collaboratively manage their files in a tag-based manner via existing, file system explorers. TagFS 2.5, a file system with tag semantics, as a means for managing,browsing and retrieving large amounts of files efficiently. On the first sight, the resulting system behaves like a traditional file system but the retrieval and change operations like copy or move carry tagging based semantics.

In 2009, [15] presented the tool *PreScan* 2.6 that scans the file system, extracts the embedded metadata from the files, transforms them to RDF and places them in a triplestore (the focus was on digital preservation, not on access and querying, nor the production of semantic entities). Generally, this tool allows the automation of the extraction, the transformation and the maintenance of the embedded metadata of digital objects.

Probably the most related that we have found is from the page `https://www.w3.org/wiki/ConverterToRdf`, that refers to tool *TripFS* 2.7 (the provided link does not work, however there is a publication from 2010: [25]). That tool exposes an entire file system as linked data, tracks changes, and links files to external data sources. The focus of that work was to publish linked data, not to support daily fs-related activities. The idea is to publish parts of a local file system as linked data, where files and directories become RDF resources. It produces random UUIDs to be used in global distributed context (for uniqueness). They use their own vocabulary to model low-level file system meta data (parent/child relationships, path, size, creation date). Extractors can be plugged into TripFS (to read files of certain format and extract RDF graph) and the main focus is on detecting file create, remove, move (rename) and update events.

In 2012 [7] described the various Semantic Desktop systems up to that year. This paper describes and explores their similarities, what they do differently, the

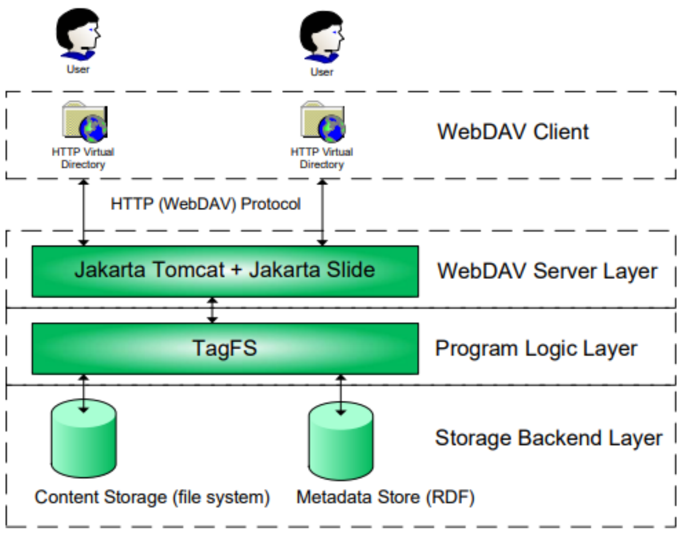
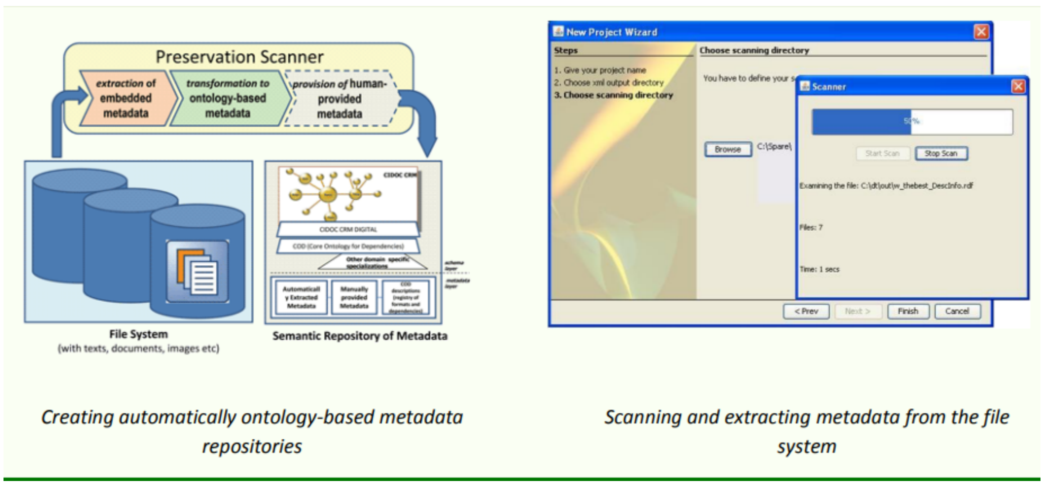


Figure 2.5: TagFS architecture (added from [23])


Preservation Scanner
extraction of embedded metadata
transformation to ontology-based metadata
provision of human-provided metadata
CIDOC CRM
CIDOC CRM DIGITAL
COD (Core Ontology for Dependencies)
Other domain specific specializations
schema layer
metadata layer
Automatically Extracted Metadata
Manually provided Metadata
COD descriptions (registry of formats and dependencies)
File System
(with texts, documents, images etc)
Semantic Repository of Metadata
New Project Wizard
Steps
1. Give your project name
2. Choose xml output directory
3. Choose scanning directory
Choose scanning directory
You have to define your s
Browse
C:\Spare\
Scanner
50%
Start Scan
Stop Scan
< Prev
Next >
Finish
Cancel
Creating automatically ontology-based metadata repositories
Scanning and extracting metadata from the file system

Examining the file: C:\dt\out\w_thebest_DescInfo.rdf

Files: 7

Time: 1 secs

Figure 2.6: PreScan tool overview (added from [15])

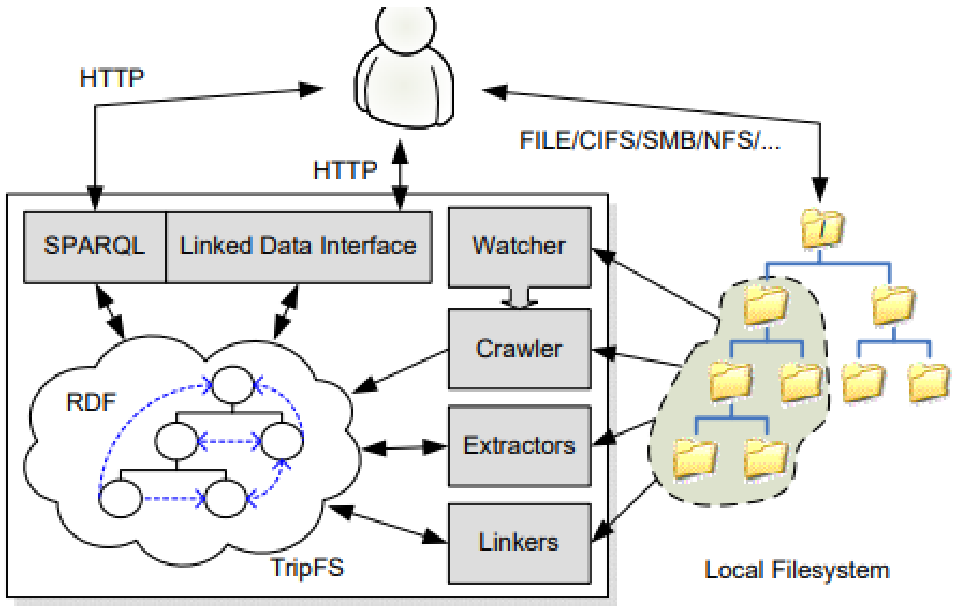


Figure 2.7: TripFS tool architecture (added from [25])

features they provide, as well as some common shortcomings and sensitive areas.

In 2018 [13] presented a new SemDesk-based prototype with emphasis on context spaces that users directly interact with and work on. The system is transparently integrated using mostly standard protocols complemented by a sidebar for advanced features. By exploiting collected context information and applying Managed Forgetting features (like hiding, condensation or deletion), the system is able to dynamically reorganize itself, which also includes a kind of tidy-up-itself functionality.

In 2016, [17] investigates the exploitation of Linked Data technology in file systems and provides a comprehensive review on the possible use of Linked Data Technology in file systems. Finally [24] points that on typical desktops , the majority of applications are not aware of Web standards, but use hierarchical file systems to organize and store information. This results in a gap between the two distinct information spaces of the Web and the desktop. To bridge this gap, is proposed a virtual file system representation of Linked Open Data (LOD) sets 2.8, through which they can be accessed as if they were present in the file system and thus easily be used within desktop applications.

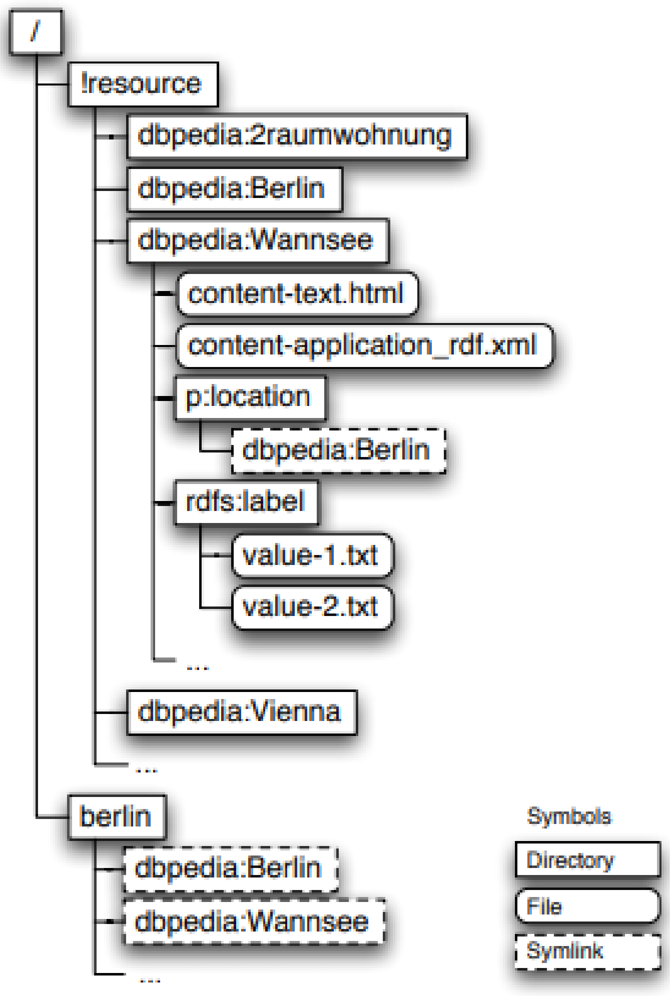


Figure 2.8: A complete virtual file system, representing resources from DBpedia (added from [17])

Chapter 3

Proposed Approach

We have identified the following key requirements, or *desiderata*:

$\mathcal{R}_1$ Do not restrict (for the shake of maintaining integrity) the way file system is used.

$\mathcal{R}_2$ Full automation, even without any configuration, that can capture the containment relationships

$\mathcal{R}_3$ Ability to configure/guide the KS construction if required, in an easy and modular way, without sacrificing the flexibility in changing the file system structure and without needing special tools/editors.

Based on this desiderata we have decided to define a process that scans ($\mathcal{R}_2$) the file system starting from the desired folder, without any restriction on its structure ($\mathcal{R}_2$), and even without configuration, produces a KG with the structure of the file system that will be directly loadable and leveraged for browsing and querying and captures the containment relationships of file systems. As regards configurability ($\mathcal{R}_3$), we propose a configuration approach relying on small configuration files (with extension `.kg`) that can be placed at the desired folders to guide the process regarding these folders. This method satisfies ($\mathcal{R}_1$), in the sense that when the user reorganizes the folders he does not have to make any change in the "`.kg`" files. Consequently, no integrity problems arise. These configuration files are small and easily edited. The configuration language is described using examples in the subsections that follow.

3.1 The Default Operation Mode (the Folder's view)

According to the default mode, that does not require any configuration, each folder becomes a class, each subfolder becomes subclass of the parent folder class, and each file becomes instance of the class of its folder. As running example, consider three nested folders $c1/c2/c3$ ($c3$ is included in $c2$ which in turn is included in $c1$) and a file $f2$ in $c2$. The essential triples related to folder $c2$ that will be produced are: { `(c2,rdfs:subClassOf,c1), (f2, rdf:type,c2)`}. The complete set of triples about $c2$ follows, where with $\langle n \rangle$ we denote the URI that is produced using the full path of an element (folder or file) with name n.

```
<c2> rdf:type  owl:Class;
  rdfs:label "c2";
  rdfs:subClassOf <c1>.
<f2> rdf:type owl:NamedIndividual;
    rdf:type <c2> .
```

The benefit from modeling folders and files in this way, is that we can use the KG to browse the folders structure and their contents (as it can be seen in the upper left part of Figure 3.1), we can exploit the semantics (the interplay of `rdf:type` and `rdfs:subClassOf`) in queries, e.g. get all files under subfolder x that satisfy a condition (e.g. with extension png).

3.2 Configuration

We can control and adapt the KG construction related to a specific folder, by placing to the desired folder a text file named "`.kg`" that contains *property-value* pairs. Table 3.1 shows an overview of all configuration commands, each described in the following subsections.

3.3 Scope Restriction

For restricting the scope of the scan, and the folders/files to be included, we can use the following two properties.
• **traverse=off**
With this line the contents (files and subfolders) of the current folder are ignored. It is useful when we do not want to include in the KG parts of the hierarchy that are not useful, like tmp folders, backup files, software binaries, etc.
• **ignoreExt=tmp;aux;class**
With this particular line all files with extension ".tmp", ".aux" or ."class" will be ignored. The user can include as many extensions as she wants to, for excluding from the KG files with no special value.

3.4 Generation of Semantic Classes and Entities

• **subFoldersClass=example:Student**
With this line for each subfolder of the current folder, an *entity instance* will be created and classified under the class `example:Student`. In our running example, if we have this line in the ".kg" file of $c2$, and by considering also the default

Syntax	Example and Notes
traverse=off	`traverse=off` Stops the traversal under the current directory (and its subfolders)
ignoreExt= ext_1;..;ext_k	`ignoreExt=tmp;aux;class` Ignores all files having the extensions ext_1 .. ext_k
subFoldersClass = $curi_1; ..; curi_k$	`subFoldersClass=example:Student` For each subfolder a new entity is created and classified under the classes $curi_1$, ..., $curi_k$. The new entity is connected with `moreAt` with the URI of the subfolder.
readme=on	`readme=on` Makes the contents of `readme.txt` (if it exists in current directory) a literal connected with `rdfs:comment` with the URI of the container folder
extraTriples $t1$;..;tk	`extraTriples= triple;tiple;...` Adds to the KG the triples $t1$, .. tk.
Extraction language	
C$_x$=$curi$	`C1=foaf:Person` All values extracted for column x of the csv file will be classified under the class $curi$
R=C$_x$,p_1,C$_y$;C$_{x2}$,p_2, C$_{y2}$	`R=C1,example:livesAt,C2;C1,example:likes,C3` Each value at column x of the csv file will be connected through property p_1 with the value at column y of the same row. Many such rules can be separated with ";"
C0	`C0=example:GreatBook` C0 corresponds to the URI of the file to which the '.kg' file refers to. Here it will be classified under the class `GreatBook`.
provenance=on	All extracted data are connected through `rdfs:isDefinedBy` with the URI of the file that contains the data

Table 3.1: Overview of the commands that can be included in '.kg' files

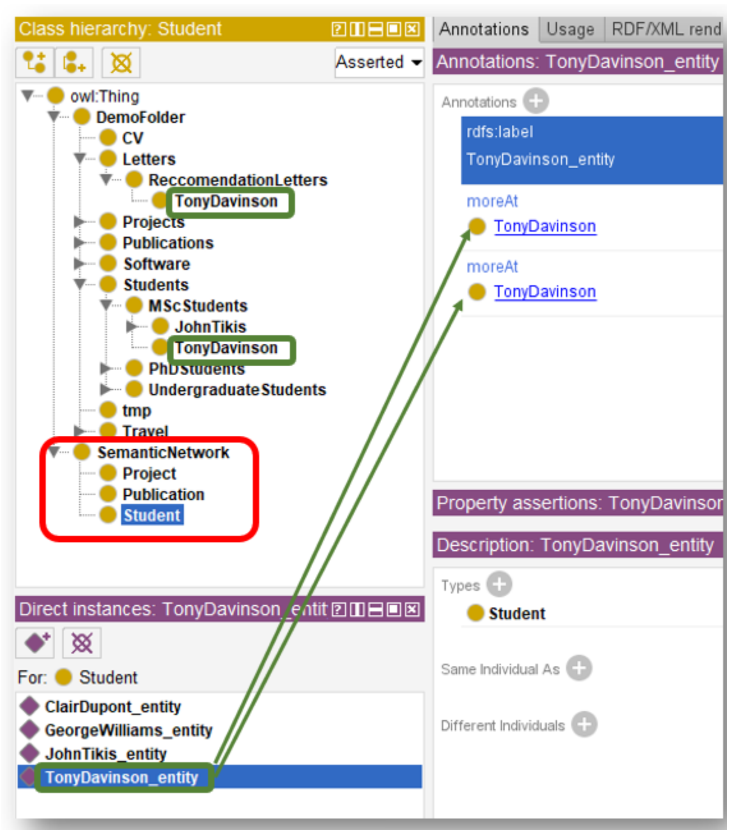


Figure 3.1: The two class hierarchies (folder's view and Semantic Network), and their connection through entities

operation (of §3.2), the following triples will be generated (the last four lines are the new ones):

```
<c2> rdf:type  owl:Class;
   rdfs:label "c2";
   rdfs:subClassOf <c1>.
<f2> rdf:type owl:NamedIndividual;
     rdf:type <c2> .
example:c2_entity
    rdf:type example:Student;
    rdfs:label "c2_entity";
    example:moreAt <c2>
```

Notice that here we have classes that do not correspond to folders (i.e. the class `example:Student`), and entities whose name is based on the folder names, (i.e. the entity `example:c2_entity`). Moreover notice that this entity is connected through `example:moreAt` with the class corresponding to *c*2.

An example of the gain is evident from Figure 3.1 that shows the result of opening the produced `output.ttl` file with Protégé. At first notice that we have two class hierarchies: the one corresponding to folders (DemoFolder), and another one, called *Semantic Network*. The latter is the superclass of the user defined classes, `example:Student` in our case. The figure shows the instances of the class `Student`, four instances in our case. Notice that `Tony Davinson` has a folder under that 'MSC Students' but also under 'Recommendation Letters'. With the proposed approach one single entity is created for that person, and that entity is related to the two folders. This facilitates browsing and querying. Note that the URIs of these entities are not location based, enabling the connection of information across various folders.

This construction is useful in various cases, e.g. in folders that like "Projects", "Students", etc., i.e. we can get all names (of projects, students, etc) as entities classified to the class name that we provided (not necessarily the same with the folder name). Moreover each such entity is related with `example:moreAt` with the folder, enabling at browsing/query time to get the resources (files) that are related to each such entity.

If we have placed this ".kg file" in every folder that contains student subfolders, then we can make queries of the form: "Get all students and their files over their folders", something that is not possible in the file system structure.

Multiple classification is also supported, for example with `subFoldersClass=example:Movies;example:Videos` each subfolder will be classified to both classes.

• **readme=on**
With this line if the current folder contains a file `readme.txt` (with any capitalization), then its contents (as a string) becomes an `rdfs:comment` of the URI of the folder. For example, if the folder `Projects/2021-H2020-ProjA` contains a `readme.txt` that contains "This is a H2020 project started in 2021", then the following triple will be produced:

```
<file:/C:/Users/..../DemoFolder/Projects/2021-H2020-ProjA/>
    rdfs:comment "This is a H2020 project started in 2021".
```

3.5 Generation of Semantic Relationships

So far we have used `rdfs:subclassOf` (for capturing the hierarchical organization of folders), `rdf:type` (for connecting instances to classes, either in the folder's KG or Semantic Network), `example:moreAt` (to relate an entity to folders containing resources about these entities), `rdfs:comment` (to express the comments of readme files as comment). A question that arise is: *How could we connect entities by semantic relations, if we wish so?* There is no direct mapping of file system constructs to this (the mapping problem is also described in [24]). To close this gap, we propose a method that is based on csv files, since these files are very commonly used and are very easy to write. If we have a csv file (text) then we can place in the same folder a file with the same name but with extension ".kg" where we can specify *rules* that control the creation of entities and relationships from the contents of the csv file. For example, suppose a file with name `Connections.txt` with the following contents:

```
David;London;Tennis
Leonardo;Rome;Football
Nicolas;Paris;Billiards
Socrates;Athens;Running
```

We can place in the same folder a file `Connections.kg` with the following contents:

```
C1=example:Student
C2=example:Location
C3=example:Sport
R=C1,example:livesAt,C2;C1,example:likes,C3
```

Property `C1` refers to the first column, and its value means that the values that occur in that column should become instances of the class `example:Student` Consequently, with the first three lines (properties C1-C3), we managed to classify all values that appear in the csv file to the classes `Student, Location` and `Sport`. The last row contains rules for creating relationships. In our case it has two rules separated by semicolon. The first is "`C1,example:livesAt,C2`" that states that the values in C1 should be connected via `example:livesAt` with the values of C2. Analogously, the second rule relates the values of the first column with the values of the third column. There is no restriction about how many columns to include in `Connections.kg` and how many rules (if any) to include. In our running example, the following triples will be produced.

```
example:David rdf:type example:Student.
example:London rdf:type example:Location.
example:Tennis rdf:type example:Sport.
example:Leonardo rdf:type example:Student.
example:Rome rdf:type example:Location.
example:Football rdf:type example:Sport.
example:Nicolas rdf:type example:Student.
example:Paris rdf:type example:Location.
example:Billiards rdf:type example:Sport.
example:Socrates rdf:type example:Student.
example:Athens rdf:type example:Location.
example:Running rdf:type example:Sport.

example:David example:livesAt example:London.
example:Leonardo example:livesAt example:Rome.
example:Nicolas example:livesAt example:Paris.
example:Socrates example:livesAt example:Athens.

example:David example:likes example:Tennis.
example:Leonardo example:likes example:Football.
```

```
example:Nicolas example:likes example:Billiards.
example:Socrates example:likes example:Running.

% schema part
example:Student rdfs:subClassOf example:SemanticNetwork .
example:Location rdfs:subClassOf example:SemanticNetwork .
example:Sport rdfs:subClassOf example:SemanticNetwork .
example:livesAt rdf:type owl:ObjectProperty.
example:likes rdf:type owl:ObjectProperty.
```

Note that there is no need to define any schema in advance; as we see, the last lines define the classes and properties encountered in the configuration file. Also note that it is not obligatory to use the `R` property; one could use only `Cx` properties for defining entities and their classes. If the `R` property is used, one can write there as many rules as she wishes to. This simple extraction language can be used also for defining schema elements, if that is required. For instance, we can define a taxonomy structured with `rdfs:subClassOf` as shown in Figure 3.2 (instead of `rdfs:subClassOf`, SKOS could be used as well).

(a) taxonomy.txt

```
EU;Location
Austria;EU
Belgium;EU
Bulgaria;EU
Croatia;EU
Cyprus;EU
CzechRepublic;EU
Brussels;Belgium
Vienna;Austria
Finland;EU
Helsinki;Finland
Rome;Italy
Italy;EU
```

(b) taxonomy.kg

```
C1=owl:Class
C2=owl:Class
R=C1,rdfs:subClassOf,C2
```

(c) Produced taxonomy

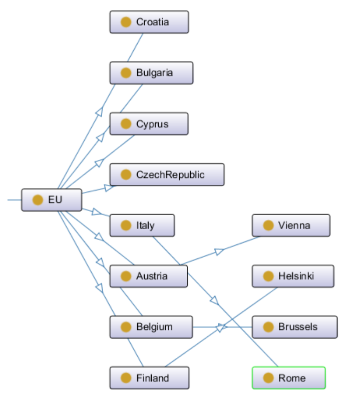


Figure 3.2: Defining a taxonomy through a csv file

The same method can be used for defining a schema (classes and properties), e.g. we can define classes and the domain and range of properties as shown next:

```
//ontology.txt                 |//ontology.kg
Car;hasEngine;Engine           | C1=owl:Class
Car;hasColor;Color             | C2=owl:ObjectProperty
Car;hasDoors;rdfs:Literal      | C3=owl:Class
                               | R=C2,rdfs:domain,C1;C2,rdfs:range,C3
```

During application testing (described later in §??), we realized that it is convenient to use the same convention for adding metadata to files that are not csv files. For this purpose we decided `C0` to refer to the URI of the current file, for example, if we have a file `BookDataManagement2022.pdf` and we create a `BookDataManagement2022.kg` with:

```
C0=example:GreatBook
R=C0,example:year,2022
```

then the URI of that book will be classified to the class `example:GreatBook` and the year 2022 will be associated with that URI. This is a convenient way to add arbitrary structured metadata to files, and make evident in the filesystem the association between the original file and the file that contain its metadata, consequently in folder/file movements it is easy to relocate both.

3.6 Extra Triples

- **extraTriples=example:Alumni rdfs:subClassOf example:Student; u1 rdf:type example:Alumni**

We noticed that sometimes there is a need to add some extra triples to connect related classes or entities, and "extraTriples" enables exactly this. It can be used for creating a common superclass (between UnderGraduateStudent and MSCStudent), for creating `rdfs:subPropertyOf` relationships which are important for query interoperability.

3.7 Provenance (connecting the Folder's view with the Semantic View)

As regards provenance, i.e. the origin of the semantic data, for the classes and individuals that correspond to folders and files (§3.1), there is no need for additional information since their real URI is used. The same is true for the semantic entities defined from subfolders's names, since they are connected with `moreAt` with the corresponding folders (as described in §3.4).

However, for entities, classes, and properties defined by csv files (as described in §3.5) there is no connection to the file from which they have been extracted from. To this end we write as comment in the ttl file the filepath of the file from which each triple was extracted, enabling one to see the origin of every produced triple. If we would like to keep this provenance information also as triples for using it during browsing or querying, then we can use the `rdfs:isDefinedBy` property and connect the URIs of such classes and individuals with the URI of the corresponding file. Since that would increase the number of triples, this mode is activated only if we add to the `.kg` file the statement: **provenance=on**. In our example, with

```
C1=example:Location
C2=example:Location
R=C1,rdfs:subClassOf,C2
provenance=on
```

for the example of Figure **??**, and the entity Croatia, we get the following extra triple:

```
example:Croatia rdfs:isDefinedBy
    <file:/C:/Users/..... /DemoFolder/Data/LocationsTaxonomy.txt> .
```

3.8 Query Manager

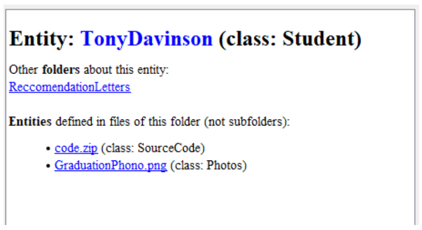
Apart from exploiting the output RDF file with an ontology editor (if the file is small), there is a need for a handy tool that could indeed help daily activities. For this purpose we have developed `FS2KG`-Q, a tool that leverages the produced triples and offers to the user useful information while working at the file system. The user can call it for the current folder, or through a file selection box can select a file or folder. Then a window shows information about the selected entry. Below we describe the information given about the selected entity.

Some notations first: For an entity e we shall use $Folders(e)$, to denote those folders that contain information about that entity, e.g. $Folders(e) = \{\ fd \mid (e, moreAt, fd) \in K\}$. We shall also use $DefinedIn(e)$ to denote the files from which e was mined, i.e. $DefinedIn(e) = \{\ f \mid (e, rdfs : isDefinedBy, f)\}$. In brief, the query manager exploits the relationships that have been created, whose schema was illustrated in Figure 1.1. In brief, for each such e we compute and show both $Folders(e)$ and $DefinedIn(e)$. Below we describe the information given of the selected entity if it is file or folder.

File fl**.** For a file fl we show the entities defined in that file (if any), i.e. all entities e such that $fl \in DefinedIn(e)$, and for each such entity e we show the folder(s) that contain information about that entity, i.e. $Folders(e)$.

Folder fr**.** If there is an entity corresponding to folder fr, its name would be `fr_entity`. If such entity exists, let's denote it by e_{fr}, we show the entity name, its class (or classes), and the other the folders that might exist that contain resources about that folder, i.e. $Folders(e_{fr})$. If there is a readme file associated with this folder we also show it. In addition, we show the entities mined from files in that folder, i.e. the entities in $MinedEnt(fr) = \cup\{\ DefinedIn(f) \mid (f, rdfs :$

$subClassOf, fr) \in K\}$ and for each such entity we could either show the related folders (i.e. if it extracted from other files), or enable the user to open that entity to get this information. The latter option is followed by our prototype. A screenshot of the card for the entity `TonyDavinson` is shown in Figure 3.3(left). We can see the additional folder that contains information about this entity, as well as entities defined in files in that folder: in this example the corresponding folder contains two files each with manually specified metadata. Currently we are in the process of implementing an explorer that combines the functionality of the classical file explorer with `FS2KG`-Q, Figure 3.3(right).

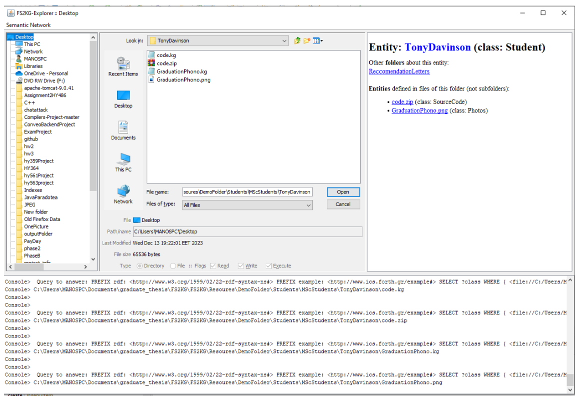


Entity: TonyDavinson (class: Student)

Other **folders** about this entity:
ReccomendationLetters

Entities defined in files of this folder (not subfolders):

- code.zip (class: SourceCode)
- GraduationPhono.png (class: Photos)



Look in: TonyDavinson

File name: soures\DemoFolder\Students\MScStudents\TonyDavinson

Files of type: All Files

File: Desktop

Path/name: C:\Users\MANOSPC\Desktop

Last Modified: Wed Dec 13 19:22:01 EET 2023

File size: 65536 bytes

Type: [✓ Directory] File :: Flags [✓ Read] [✓ Write] [✓ Execute]

Entity: TonyDavinson (class: Student)

Other **folders** about this entity:
ReccomendationLetters

Entities defined in files of this folder (not subfolders):

- code.zip (class: SourceCode)
- GraduationPhono.png (class: Photos)

```
Console>  Query to answer: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX example: <http://www.ics.forth.gr/example#> SELECT ?class WHERE { <file://C:/Users/M
Console> C:\Users\MANOSPC\Documents\graduate_thesis\FS2KG\FS2KG\Resoures\DemoFolder\Students\MScStudents\TonyDavinson\code.kg
Console>
Console>
Console>  Query to answer: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX example: <http://www.ics.forth.gr/example#> SELECT ?class WHERE { <file://C:/Users/M
Console> C:\Users\MANOSPC\Documents\graduate_thesis\FS2KG\FS2KG\Resoures\DemoFolder\Students\MScStudents\TonyDavinson\code.zip
Console>
Console>
Console>  Query to answer: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX example: <http://www.ics.forth.gr/example#> SELECT ?class WHERE { <file://C:/Users/M
Console> C:\Users\MANOSPC\Documents\graduate_thesis\FS2KG\FS2KG\Resoures\DemoFolder\Students\MScStudents\TonyDavinson\GraduationPhono.kg
Console>
Console>
Console>  Query to answer: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX example: <http://www.ics.forth.gr/example#> SELECT ?class WHERE { <file://C:/Users/M
Console> C:\Users\MANOSPC\Documents\graduate_thesis\FS2KG\FS2KG\Resoures\DemoFolder\Students\MScStudents\TonyDavinson\GraduationPhono.png
```

Figure 3.3: `FS2KG`: Query/browsing interfaces

Chapter 4

Implementation, Applications and Extensions

4.1 Implementation

We have implemented a proof-of-concept prototype called `FS2KG`, a Java-based file explorer application that integrates traditional file management functionalities with semantic technologies,where the user configuration and the knowledge graph enable semantic querying ,resulting in an intelligent and context-aware tool for users. The application provides users with a unified interface for efficient file navigation and enhanced contextual understanding. Key features include a central workspace for file interaction, a hierarchical file tree structure for intuitive navigation, and a dedicated panel for displaying semantic information associated with selected files. The console at the bottom of the application logs executed SPARQL queries, enabling users to interact with linked data and explore semantic connections within their file system. By seamlessly combining file management with semantic capabilities, the application empowers users to discover valuable insights and relationships within their data.

4.1.1 Technical Overview

The application utilizes Java for its cross-platform compatibility and object-oriented programming capabilities. The whole application is composed of 3 major components: The FileSystemToKG , the QueryManager and the FileBrowser class. All three classes are dependent on the Jena framework, an open-source project developed by the Apache Software Foundation and provides a set of tools and APIs for working with RDF (Resource Description Framework) data, ontologies, and semantic technologies. The **FileSystemToKG** class contains all the methods needed to create a Knowledge Graph(KG) out of a directory. It scans all files and sub-folders of a given folder and expresses that structure in RDF. It supports a modular configuration approach relying on .kg files and various conventions. The most important method is :

```
public static void traverseAndCreateKG (String startupfolder,
String filenameToWrite);
```

This method starts the traveral and the creation of the knowledge graph. The *startupfolder* is the folder where scan should begin and *filenameToWrite* is the file to write the produced RDF triples.

The **QueryManager** class encapsulates the Knowledge Base interaction logic, providing a clean and organized way to handle queries. Uses a ttl file which is the Knowledge Graph, a starting folder and offers a programmatic query functionality(SPARQL independent). The main methods in this class are:

```
List folders(String e);
```

Takes as input an entity and returns the folders related to that entity. As parameter we pass a string variable *e* which is the URI of an entity.

```
List entitiesDefinedInFile(String f)
```

Takes as input the URI of a file and returns the entities defined in that file. As parameter we pass a string variable *f* which is the URI of a file.

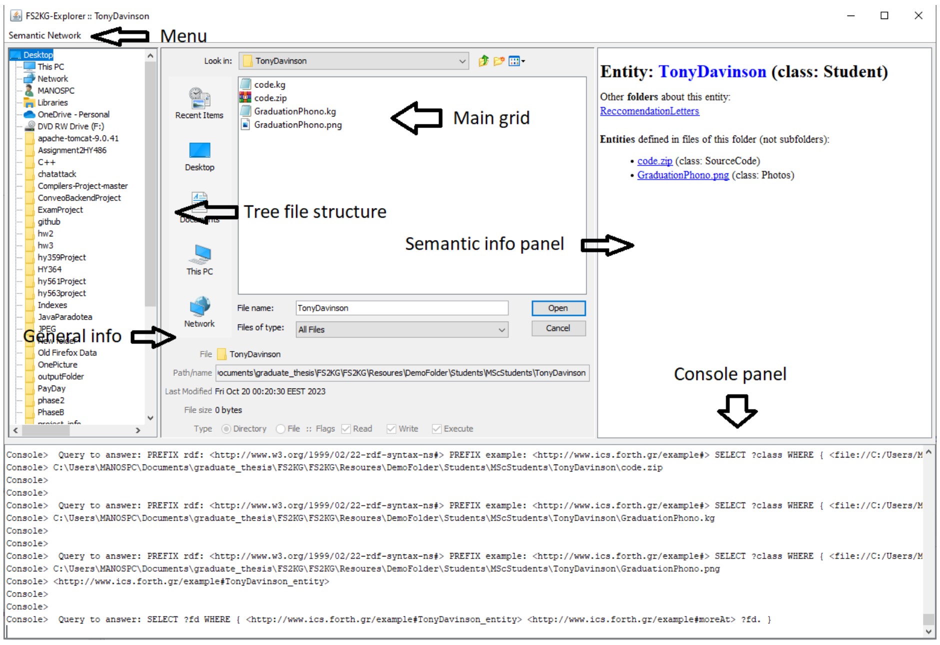
```
List entityOffolder(String fd) {
```

Takes as input a folder fd and return the corresponding entity if it exists. As parameter we pass a string variable *fd* which is the URI of a folder. The convention is that if entity exists then its URI = folder-URI +entity.

```
List getComment(String f) {
```

Takes as input a folder fd and returns the corresponding comment if it exists. As parameter we pass a string variable *f* which is the URI of a folder.

The user interface of this app is located in the **FileBrowser** 4.1 class which utilizes the both of previously mentioned classes. It is constructed on Java Swing's threading model and is consisted of a main JFrame. The JFrame contains a JMenu with the option to create a KG, a JTree which displays the file tree structure and 3 major JPanels which are the main grid (center), the semantic info (right) and the console (below). Some JButtons are displayed such as *Open* and *Cancel* while there are some disabled fields below the main grid which show basic info. The whole functionality is based on EventListeners and PropertyChangeListeners which combine the QueryManager object with a GUI update method. In more details, there is a render function that initializes the UI of the application:



File TonyDavinson
Path/name ocuments\graduate_thesis\FS2KG\FS2KG\Resoures\DemoFolder\Students\MScStudents\TonyDavinson
Last Modified Fri Oct 20 00:20:30 EEST 2023
File size 0 bytes
Type (● Directory) ( ) File :: Flags ✓ Read ✓ Write ✓ Execute

Entity: TonyDavinson (class: Student)

Other **folders** about this entity:
ReccomendationLetters

Entities defined in files of this folder (not subfolders):

- code.zip (class: SourceCode)
- GraduationPhono.png (class: Photos)

```
Console>  Query to answer: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX example: <http://www.ics.forth.gr/example#> SELECT ?class WHERE { <file://C:/Users/M
Console> C:\Users\MANOSPC\Documents\graduate_thesis\FS2KG\FS2KG\Resoures\DemoFolder\Students\MScStudents\TonyDavinson\code.zip
Console>
Console>
Console>  Query to answer: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX example: <http://www.ics.forth.gr/example#> SELECT ?class WHERE { <file://C:/Users/M
Console> C:\Users\MANOSPC\Documents\graduate_thesis\FS2KG\FS2KG\Resoures\DemoFolder\Students\MScStudents\TonyDavinson\GraduationPhono.kg
Console>
Console>
Console>  Query to answer: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX example: <http://www.ics.forth.gr/example#> SELECT ?class WHERE { <file://C:/Users/M
Console> C:\Users\MANOSPC\Documents\graduate_thesis\FS2KG\FS2KG\Resoures\DemoFolder\Students\MScStudents\TonyDavinson\GraduationPhono.png
Console> <http://www.ics.forth.gr/example#TonyDavinson_entity>
Console>
Console>
Console>  Query to answer: SELECT ?fd WHERE { <http://www.ics.forth.gr/example#TonyDavinson_entity> <http://www.ics.forth.gr/example#moreAt> ?fd. }
```

Figure 4.1: FS2KG technical overview

```
public Container getGui() throws TreeException;
```

Especially the main grid where the user has the most interaction with is a JFileChooser, which is bind to an EventListener and a PropertyChangeListener:

```
    fileChoser = new JFileChooser();
    fileChoser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
    fileChoser.addPropertyChangeListener(listener);
    fileChoser.addActionListener(listener);
```

Every time a user clicks on the main grid and fires an event, the *actionPerformed* or the *propertyChange* function is called. The application has a state, in which keeps the current entity, current folder or current file. Every event that is handled, updates the state and accordingly calls a method to update the UI based on the application's state:

```
     public void paintHTML(String queryResults,String path);
```

4.1.2 Main grid

The application starts from the root folder. This is the primary workspace where users can interact with their files. It likely displays a grid or list of files and folders for easy navigation and management. This includes actions such as opening files, creating new folders, renaming items, and moving or copying files. As mentioned earlier, it is consisted of a file chooser component(JFileChooser) with all the features provided by that class. The *Open* button will update the UI to the specific location and the *Cancel* button will close the whole app. There is also a component below, which is a main panel(JPanel) with two panels next to each other with some info about the current location the user is. The first panel of the main, is the for file details labels while the other one is for the values.

```
// details for a File
JPanel fileMainDetails = new JPanel(new BorderLayout(4,2));
fileMainDetails.setBorder(new EmptyBorder(0,6,0,6));

JPanel fileDetailsLabels = new JPanel(new GridLayout(0,1,2,2));
fileMainDetails.add(fileDetailsLabels, BorderLayout.WEST);

JPanel fileDetailsValues = new JPanel(new GridLayout(0,1,2,2));
fileMainDetails.add(fileDetailsValues, BorderLayout.CENTER);
```

Some of the info that are displayed are: The type of the file(icon) followed by the file name, the absolute path of the file selected, last modified date, the file size and some readonly radio buttons which indicate whether its a file or a directory and the permissions. See 4.2

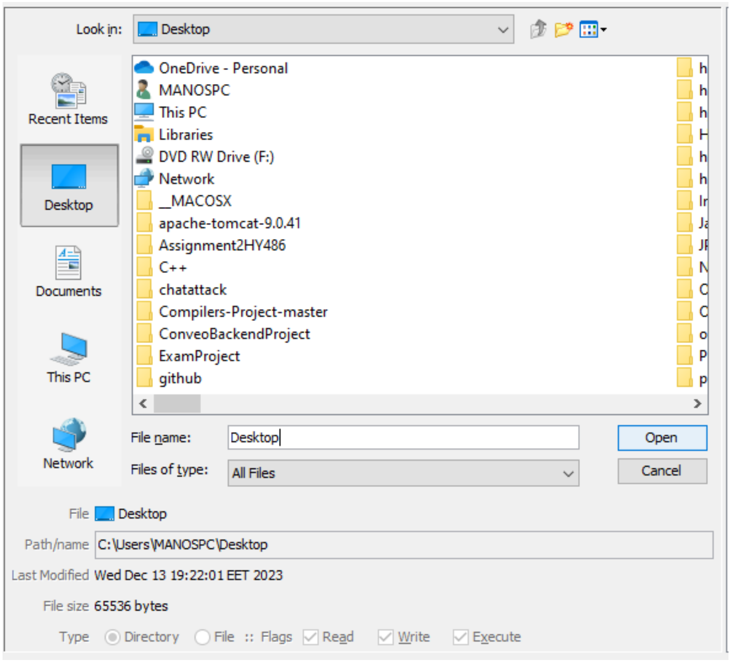
```
//file type
fileDetailsLabels.add(new JLabel("File", JLabel.TRAILING));
fileName = new JLabel();
fileDetailsValues.add(fileName);

//absolute path
fileDetailsLabels.add(new JLabel("Path/name", JLabel.TRAILING));
path = new JTextField(5);
path.setEditable(false);
fileDetailsValues.add(path);

//last modified date
fileDetailsLabels.add(new JLabel("Last Modified", JLabel.TRAILING));
date = new JLabel();
fileDetailsValues.add(date);

//file size
fileDetailsLabels.add(new JLabel("File size", JLabel.TRAILING));
size = new JLabel();
fileDetailsValues.add(size);

//directory or file and the permissions
```



Look in: Desktop

File name: Desktop

Files of type: All Files

File Desktop

Path/name C:\Users\MANOSPC\Desktop

Last Modified Wed Dec 13 19:22:01 EET 2023

File size 65536 bytes

Type ◉ Directory ○ File :: Flags ✓ Read ✓ Write ✓ Execute

Figure 4.2: The main grid

```
fileDetailsLabels.add(new JLabel("Type", JLabel.TRAILING));
JPanel flags = new JPanel(new FlowLayout(FlowLayout.LEADING,4,0));
isDirectory = new JRadioButton("Directory");
flags.add(isDirectory);
isFile = new JRadioButton("File");
flags.add(isFile);
fileDetailsValues.add(flags);
```

4.1.3 File tree structure

The file tree structure visually represents the hierarchical organization of files and folders on the user's system. Users can expand and collapse nodes to navigate through different levels of the directory structure. The file tree structure reveals its sub-directories and files allowing efficient exploration and navigation, especially when dealing with deep folder structures. Technically, this is a hierarchical tree component(DefaultMutableTreeNode) used in conjunction with the JTree component. The file tree hierarchy is constructed when the UI is initialized. The first step is to retrieve the root directories of the file system using *fileSystemView.getRoots()*. The *fileSystemView* object is an instance of *javax.swing.filechooser.FileSystemView*, which provides access to the file system. If the roots are null, it throws a custom exception (TreeException) indicating that the file system view roots are null. The next part iterates through each root directory obtained from the file system. For each root, it creates a *DefaultMutableTreeNode* with the root File object and adds it to the root node (assuming root is the root of your tree). Then, it retrieves the files within the current root using *fileSystemView.getFiles(fileSystemRoot, true)*. The second parameter true in the getFiles method indicates that it should recursively retrieve files within sub-directories. For each file in the obtained list, it checks if it's a directory (file.isDirectory()). If it is, it creates a new DefaultMutableTreeNode with the directory File object and adds it as a child to the current root node (node). This process is repeated for each root directory, creating a tree structure where the root node represents the file system roots, and each child node represents a directory within the file system. See figure 4.3

```
// the File tree
DefaultMutableTreeNode root = new DefaultMutableTreeNode(rootDirectory);
treeModel = new DefaultTreeModel(root);

// show the file system roots.
File[] roots = fileSystemView.getRoots();
if(roots == null) {
    throw new TreeException("File system view roots are null");
}
for (File fileSystemRoot : roots) {
    DefaultMutableTreeNode node = new DefaultMutableTreeNode(fileSystemRoot);
    root.add( node );
    File[] files = fileSystemView.getFiles(fileSystemRoot, true);
    for (File file : files) {
        if (file.isDirectory()) {
            node.add(new DefaultMutableTreeNode(file));
        }
    }
    //
}
```

4.1.4 Semantic info panel

The panel dynamically updates its content based on user interactions, providing real-time semantic information related to the selected file or folder in the main grid. This dynamic nature ensures that users receive up-to-date and relevant details. The panel may showcases the semantic metadata associated with the selected folder/file. The semantic info panel is a panel (JPanel) which has a Swing component that is used for displaying and editing formatted text (JEditorPane) in conjunction with JScrollPane which is a Swing component that provides a scrollable view of a component or a group of components. Every time the state changes the UI on the semantic info panel is repainted. Basically, this panel resembles the

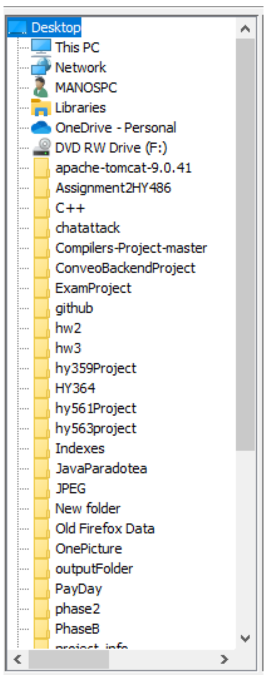


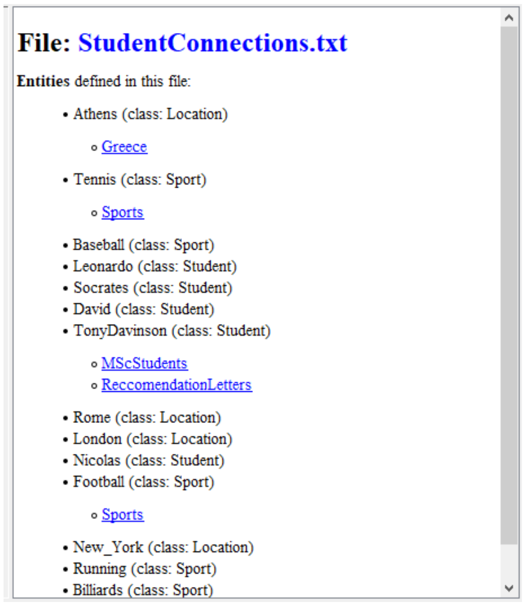
Figure 4.3: The tree file structure

state of the application in HTML format. However a user cannot type and edit the semantic info panel as it is read-only. There are 2 main states displayed in this panel: Entity and File (See 4.4 and 4.5). The entity is displayed as: *Entity: name*, a comment if it is defined in a gray color, other folders that this entity can be found and the entities that are defined in the current folder. We also support multiple classification as some entities may have more than one classes (see 4.6). On the other hand the file is displayed as: *File: name* and the enities that are defined in this file as a list. Every list option has a clickable sub-list option which indicates the folder a user can find these entities. Finally every option that is clickable inside the semantic info panel, triggers a state update and therefore a UI update, redirecting the user to the clicked option.

```
//semantic info panel
html = new JPanel(new BorderLayout(3,3));

//html editor pane
JEditorPane editorPane = new JEditorPane();
editorPane.setEditorKit(JEditorPane.createEditorKitForContentType("text"));
editorPane.setEditable(false);

//Put the editor pane in a scroll pane.
JScrollPane editorScrollPane = new JScrollPane(editorPane);
editorScrollPane.setVerticalScrollBarPolicy(
```


File: StudentConnections.txt

Entities defined in this file:

- Athens (class: Location)
  - Greece
- Tennis (class: Sport)
  - Sports
- Baseball (class: Sport)
- Leonardo (class: Student)
- Socrates (class: Student)
- David (class: Student)
- TonyDavinson (class: Student)
  - MScStudents
  - ReccomendationLetters
- Rome (class: Location)
- London (class: Location)
- Nicolas (class: Student)
- Football (class: Sport)
  - Sports
- New_York (class: Location)
- Running (class: Sport)
- Billiards (class: Sport)

Figure 4.4: The semantic info panel displaying a file

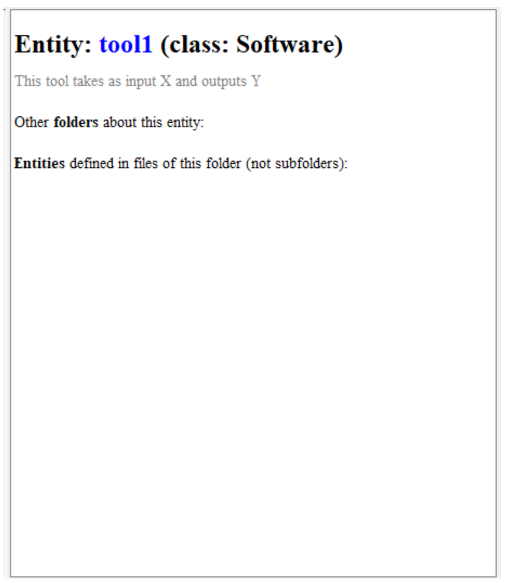
```
                JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
editorScrollPane.setPreferredSize(new Dimension(450, 345));
editorScrollPane.setMinimumSize(new Dimension(10, 10));

html.add(editorScrollPane);
```

4.1.5 Console panel

The console logs the execution of SPARQL queries, providing users with a clear record of queries they have run. This logging is valuable for users to review their query history, identify patterns, and track their interactions with the semantic features of the application. Also the console provides detailed error messages and feedback for query execution errors or other system events. This helps users quickly identify and address issues, enhancing the overall usability of the application. The console panel is constructed by a JPanel which is used to store the system output and system error. In more details, we created a component *TextAreaOutputStream* which appears to be a custom implementation of the OutputStream class in Java. It allows us to capture and display data written to the output stream in a graphical user interface which is a JTextArea. Upon the application initialization, we use a function setConsoleLog() that creates a component of this custom class and redirects the PrintStream to itself. See figure 4.7

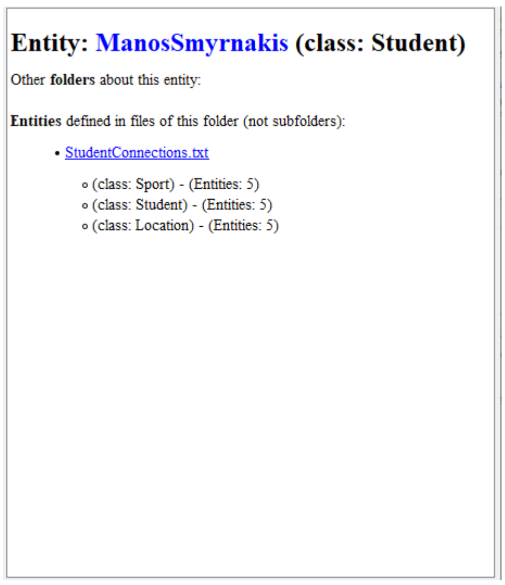
```
//Set console function
 public void setConsoleLog() {
```


Entity: tool1 (class: Software)

This tool takes as input X and outputs Y

Other **folders** about this entity:

Entities defined in files of this folder (not subfolders):

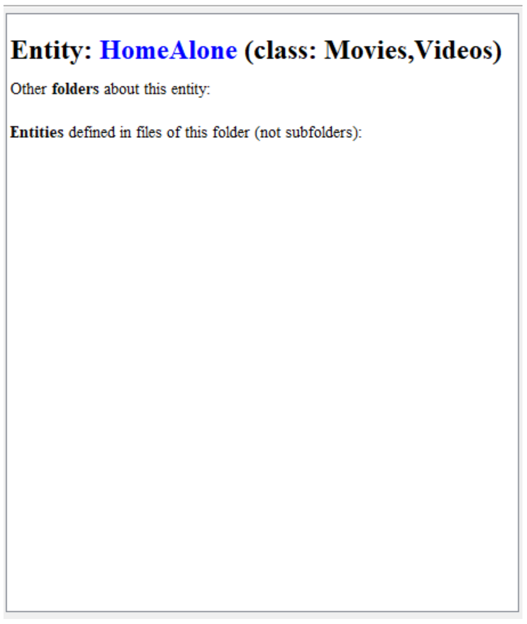

Entity: ManosSmyrnakis (class: Student)

Other **folders** about this entity:

Entities defined in files of this folder (not subfolders):

- StudentConnections.txt
  - (class: Sport) - (Entities: 5)
  - (class: Student) - (Entities: 5)
  - (class: Location) - (Entities: 5)

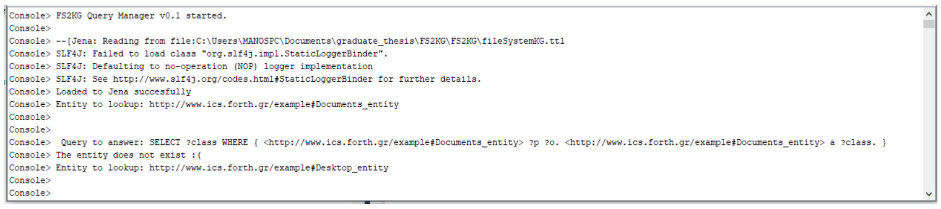
Figure 4.5: The semantic info panel displaying entities


Entity: HomeAlone (class: Movies,Videos)

Other **folders** about this entity:

Entities defined in files of this folder (not subfolders):

Figure 4.6: Multiple classified entity



```
Console> FS2KG Query Manager v0.1 started.
Console>
Console> --[Jena: Reading from file:C:\Users\MANOSPC\Documents\graduate_thesis\FS2KG\FS2KG\fileSystemKG.ttl
Console> SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
Console> SLF4J: Defaulting to no-operation (NOP) logger implementation
Console> SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Console> Loaded to Jena succesfully
Console> Entity to lookup: http://www.ics.forth.gr/example#Documents_entity
Console>
Console>
Console>  Query to answer: SELECT ?class WHERE { <http://www.ics.forth.gr/example#Documents_entity> ?p ?o. <http://www.ics.forth.gr/example#Documents_entity> a ?class. }
Console> The entity does not exist :(
Console> Entity to lookup: http://www.ics.forth.gr/example#Desktop_entity
Console>
Console>
```

Figure 4.7: The console panel

```
        //create a JTextArea
        JTextArea textArea = new JTextArea(15, 30);
        textArea.setFont(textArea.getFont().deriveFont(12f));

        //Create a custom TextAreaOutputStream component
        TextAreaOutputStream taOutputStream = new TextAreaOutputStream(
                 textArea, "Console");

        output= new PrintStream(taOutputStream);

         console.setLayout(new BorderLayout());
         console.add(new JScrollPane(textArea,
         JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
         JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));

         //redirect sysout and syserror to custom component
         System.setOut(output);
         System.setErr(output);
    }
```

4.1.6 Knowledge graph creation

Users can generate Knowledge Graphs by specifying source and output folders. Clicking on the menu on top right will direct them to choose the source folder and the folder where the KG file should be placed. A Turtle (.ttl) file is generated, capturing semantic relationships and metadata from the file system and the produced file is directly loadable by an ontology editor, like Protégé. If the size of the .ttl file is larger than the size the editor can handle,the user can import the triples to a triplestore. The process of the KG creation is represented as a JMenuOption, which is bind to a JMenuListener, of the JMenu on top left When a user clicks the option, the *actionPerformed* function of the JMenuListener is called, asking the user to choose the source folder at first with a JOptionPane and a JFileChooser. Choosing the source folder, the application will ask for the destination folder the same way it did with the source. Once both source folder and destination folder are defined the KG is created and stored in the destination folder under the name *fileSystemKG.ttl.* while the system will automatically open the file with the default selected app. As a new KG is created, a new instance of the QueryManager component is created with the new KG. See figure 4.8

```
if (returnValue1 == JFileChooser.APPROVE_OPTION && !sourceFolder.equals("")) {
    File selectedFile = jfc1.getSelectedFile();

    //selected destination folder
    String destinationFolder = selectedFile.getAbsolutePath();
    //absolute path for the ttl file
    String createdTTLFile=destinationFolder+"/fileSystemKG.ttl";

    // creates the knowlege graph
    FileSystemToKG.traverseAndCreateKG(sourceFolder,createdTTLFile );
    // opens the ttl using the associated application/editor (if any)
    in the host computing system
    FileSystemToKG.open(createdTTLFile);

    //creates a new instance of the QueryManager
    qm = new QueryManager(startFolder,createdTTLFile);
}
```

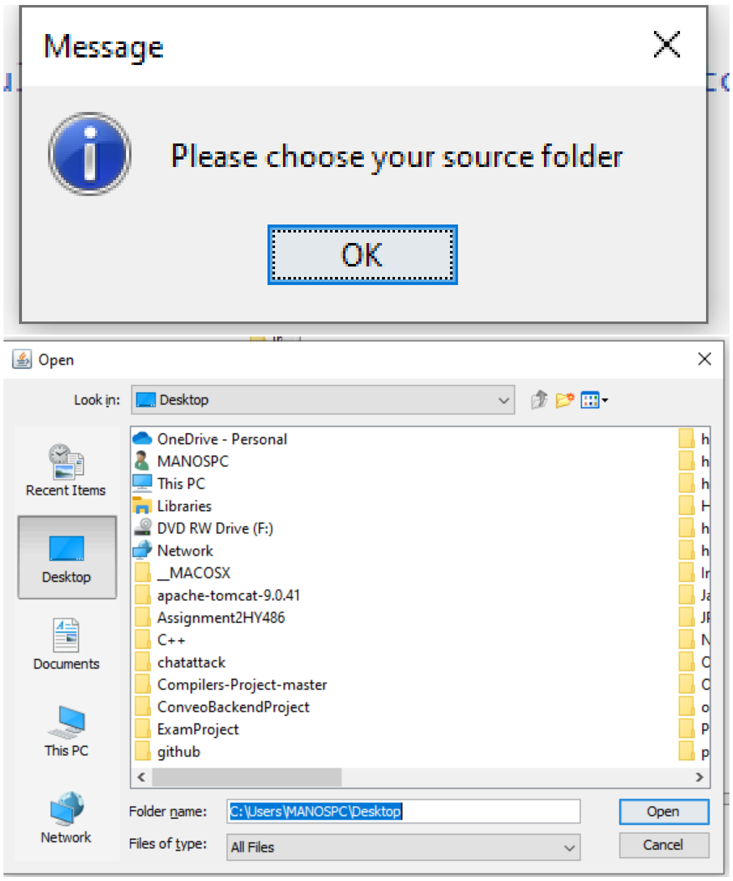


Figure 4.8: Creating a Knowledge graph

4.2 Applicability

We can identify two main scenarios: ($\mathcal{S}_1$) Over existing file systems to enable querying, identification and grouping of entities scattered in different subfolders, and ($\mathcal{S}_2$) Over folder structures created for facilitating KG construction (e.g. papers organized in categories, the user can use the file system for this purpose instead of using an taxonomy/ontology editor).

The first scenario FS2KG will allow users to perform sophisticated queries within their existing file systems. Instead of relying solely on traditional search methods, users can now leverage semantic technologies to execute queries based on the meaning and relationships embedded in the files. Also, through semantic annotations and metadata, FS2KG enhances the identification process, providing a more intelligent and context-aware approach to locating specific entities. By recognizing semantic connections between files, it organizes them into meaningful groups, reducing the manual effort required for categorization.

Based on the second scenario, users can leverage existing folder structures for KG construction. For example, they can organize research papers into categories and subcategories within the file system, effectively constructing a KG without the need for a separate taxonomy or ontology editor. The file system serves as an intuitive and user-friendly interface for KG construction. Finally, as users continue to organize and modify their folder structures, FS2KG dynamically updates the underlying KG. This ensures that the KG evolves in sync with changes in the file system, maintaining a real-time representation of the knowledge landscape.

4.3 Test Case: Personal Library

We tested the process and the tool over a personal library with books and research papers, that are organized in folders, containing no metadata not any other system. Some empirical results from using it follow. This exercise made the owner of the library to fix errors and improve the taxonomy (she increased the nesting level and performed various renamings). By inspecting and visualizing the produced RDF using an ontology editor, the author further changed the file structure (names and nesting). Overall, this exercise improved the folder structure as well as the names of the folders and files. Since the library contained a folder "byAuthor" that contains books and papers organized by authors, she used the `subFoldersClass=example:Author` to get as entities all these authors. She used `traverse=off` for excluding folders with administrative data (book borrowings, etc). Moreover, for cases where a paper/book should in two (semantic) classes, she added metadata using the *filename.kg* option (and $C0$) to specify the extra classes, to achieve a multi-faceted organosis.

Test Case	FS size	#Folders	#Files	Output ttl size	Time required
test1-small	11.2 GB	9,513	42,982	18.8 MB (19,795,667 bytes)	15 secs
test2-big	139.1 GB	60,730	382,601	139 MB	98 secs

Table 4.1: Experiments related to efficiency

4.4 Efficiency

The efficiency is that of plain folder structure scanning. The main memory requirements are very limited, since `FS2KG` it does not keep anything in main memory, it directly writes the '`.ttl`' file while scanning. The only main memory that is required is that for supporting the recursive calls (depth-first-search, thus low main memory requirements). In general the proposed approach is scalable (time complexity $O(n)$ where n is the number of folders and files). By testing the tool in various file systems, we observed that the measured times are faster than the "right-click Properties" of a folder (since the later also counts the size in disk). Table 4.1 shows some indicative measurement over real file systems (the measurements were performed using an ordinary laptop with 1.8 GHz i7, 4MB cache and 16 GB of RAM, running Microsoft Windows 10 Pro). We can see that it takes less than 90 seconds to produce a ttl file of size 140 MB for a file system with 382K files and 60K folders (the size of the file system is 140 GB).

4.5 Possible Extensions and Discussion

We have decided to include functionality related to the main hypothesis. On top of this functionality, several straightforward extensions are applicable (since they have already been studied in isolation). This includes *KG enrichment*, e.g. (a) representation of the filesystem's file metadata in RDF (as in [12]), (b) extraction of the embedded in the files metadata and representation in RDF (as in [15]), (c) instance matching over the KG to establish connections between entities whose name is slightly different in different folders, *traversal services*, e.g. (d) regex-based specification of the desired files/folders (as in web crawlers), *extraction services*, e.g. (e) augment with information extraction capabilities from files according to their type (text, etc) based on the application content and requirements at hand (including scripts in the '.kg' files), *access services*, e.g. (f) materialize the extracted triples from big csv files, to avoid re-extracting them in the next KG reconstruction, if the files have not been changed in the meantime, and (g) keyword search based on both the contents of the files and produced KG (as in [19]).

Chapter 5

Evaluation

In the initial phases of our project development, we executed a meticulous and deliberately small-scale task-based evaluation involving end-users. The primary objectives of this evaluation were twofold: firstly, to assess the users' proficiency and affinity towards the interaction paradigm introduced in our system, and secondly, to solicit valuable feedback aimed at enhancing both the graphical user interface (GUI) and the underlying procedural framework.

This exploratory evaluation was designed to gauge the effectiveness and user-friendliness of the interaction paradigm under consideration. By formulating specific tasks representative of real-world scenarios, we sought to determine the users' ability to seamlessly navigate through the system and to comprehend and utilize the novel interaction model. Simultaneously, we aimed to uncover insights into the users' subjective experiences by investigating their preferences and attitudes towards the introduced paradigm.

The limited scale of the evaluation was intentional, serving as an initial exploration to identify potential challenges, strengths, and areas for improvement. Emphasizing a task-based approach allowed us to observe users in action, providing valuable insights into how well the system aligns with their expectations and requirements.

Crucially, the evaluation served as a conduit for gathering comprehensive feedback from participants. Users were encouraged to share their thoughts, suggestions, and concerns regarding both the GUI and the overall user interaction process. This qualitative feedback proved instrumental in recognizing nuances that might have otherwise gone unnoticed and in shaping a roadmap for iterative refinement.

During the preliminary and deliberately small-scale task-based evaluation conducted with end-users, we carefully designed and executed a series of three distinct scenarios. Each scenario was thoughtfully crafted to encompass a range of user interactions and challenges that users might encounter within our system. The inclusion of multiple scenarios aimed to provide a comprehensive assessment of user experiences across various aspects of the interaction paradigm.

5.1 Evaluation scenarios

5.1.1 Scenario A

The first scenario has as (Scenario A) a primary goal to evaluate the usability of the explorer tool when used by individuals without specialized technical knowledge. This scenario focuses on the user's ability to navigate the system, perform specific tasks, and provide feedback on their experience. Participants were presented with a demo folder containing pre-configured files and folders. This setup ensures a standardized environment for testing. After the setup, users were instructed to use the explorer tool to navigate through the provided demo folder. Participants were assigned specific tasks 5.1, each accompanied by a corresponding question to be addressed. The purpose was to guide users through predefined activities and prompt them to articulate their insights, observations, and responses to the associated questions for a more comprehensive evaluation. This approach facilitated a structured examination of user interactions and experiences within the context of the designated tasks.

Table 5.1: Evaluation Tasks: Scenario A

ID	Task
TA1	Navigate into the folder Evaluation/DemoFolder that you have created. How many (direct) sub-folders it contains?
TA2	Is there any folder named TonyDavidson inside the DemoFolder? If so which is its parent folder?
TA3	Provide the class of the TonyDavidon entity.
TA4	How many folders about the entity TonyDavidson exist?
TA5	Which other entities did you find in the TonyDavidson directory, and which are their classes? (If there are any)
TA6	Navigate to the folder Data/2-LocationsTaxonomy and find how many European countries are listed.

5.1.2 Scenario B

Scenario B aims to evaluate the proficiency of expert users in utilizing the configuration language of the explorer tool. This scenario involves providing a designated folder, a tutorial on configuration options, and tasks to assess the users' ability to write configuration files and effectively use the explorer. Participants are given access to a folder, akin to the demo folder used in previous scenarios. This folder serves as the environment for configuring and testing the explorer tool. Also, users are provided with a tutorial detailing the various configuration options available within the tool. The tutorial offers guidance on syntax, parameters, and best practices for configuring. At first, they are tasked with writing a few configuration options or even whole files based on the tutorial provided. The goal is to assess their proficiency in translating their understanding of the configuration language into functional and effective configuration files. Based on each configuration file that was constructed/edited, participants utilize the explorer tool with their newly created configuration files to evaluate their ability to apply the configurations and navigate through the system seamlessly. Tasks 5.2 were also handed in this scenario each accompanied by a corresponding question to be answered.

Table 5.2: Evaluation Tasks: Scenario B

ID	Task
TB1-a	Navigate to the Data/2-LocationsTaxonomy and add Paris. What did you have to write and where?
TB1-b	Using FS2KG-Explorer, create the KG and navigate to Data/2-LocationsTaxonomy. Can you see Paris as entity when clicking the LocationsTaxonomy.txt?
TB2-a	Enrich the .kg file at the folder Sports so that all subfolders are classified to a class example:Sport. What did you have to write and where?
TB2-b	Using FS2KG-Explorer, create the KG. Open with Protege the produced ttl. Is the class example:Sport and its two instances defined properly? Write down the instances.
TB3-a	Enrich the folder Software with a subfolder "tool2", write inside that folder a Readme file, and create a .kg file so that the contents of the readme file become rdfs:comment of tool2. What did you have to write and where?
TB3-b	Using FS2KG-Explorer navigate to tool2 and ensure that the comment is visible at the right frame. Is that visible?
TB4-a	Place in the subfolder "tool2", a "main.java" file and classify it with the class "example:Java". What did you have to write and where?
TB4-b	Using FS2KG-Explorer navigate to tool2. Is the file main.java and its class visible at the right frame?

5.1.3 Scenario C

Here users were asked to use the approach over their own file systems for obtaining a functionality that is useful for them. They were free to propose and implement whatever they wanted. Obviously, this scenario involves running the system, exploring outputs using ontology editors and knowledge management systems, connecting folders related to the same entity using configuration files, producing a Knowledge Graph (KG), evaluating the functionality of the system's browser, and implementing new functionality. Seven teams comprised of computer science graduate students were established. Among them, four proposals fit to the category ($\mathcal{S}_1$), i.e. construction a KG to improve the management of the file system contents, while the remaining teams employed both the FS2KG explorer and the configuration procedure for knowledge graph construction ($\mathcal{S}_2$). Below we summarize these projects.

1. **User friendly configuration ($\mathcal{S}_1$).** Certain students pointed out that the configuration process could pose challenges for users, particularly as the file system expands. They proposed the implementation of an innovative assistive system for the FS2KG platform, resulting in a more user friendly experience configuring the file system. In detail, the process of creating additional '.kg' files for the Knowledge Graph (KG), within the FS2KG platform,

is simplified through a web application which uses Java Spring and Angular JS.

2. **Google Drive cloud extension ($\mathcal{S}_1$).** Some users identified the current limitation that lies in FS2KG exclusive compatibility with local file systems. This restriction requires the presence of the entire folder structure within the local storage, posing significant storage challenges for large-scale projects. Consequently, the users proposed a cloud extension tailored to Google Drive storage which enables KG creation without the need to download large files and folders, thereby reducing storage requirements. However, the integration of cloud functionality introduces its own set of challenges, notably network overhead, and dependency on cloud providers.

3. **Streamline code and software organization ($\mathcal{S}_1$).** Based on our approach and implementation a user wanted to tackle an issue that many developers may face nowadays which is navigating through extensive code bases and repositories. The solution that is proposed has as main goal to create knowledge graphs for programs or code snippets, capturing elements such as the content of the code and the programming language. User extended the FS2KG system's functionality and combined it with an intuitive GUI which facilitates queries, providing information about the desired code. The fields are defined to inquire about programs with a specific language and/or specific content.

4. **Comprehensive KG for organizing academic workload ($\mathcal{S}_1$, $\mathcal{S}_2$).** A user recognized the necessity of organizing academic tasks within a student's workspace. Utilizing both the FS2KG and the configuration system, the user developed a Knowledge Graph that cleverly leverages the hierarchical structure of a File System to empower students in organizing their academic folders efficiently. The primary idea in this Knowledge Graph is that seamlessly connects all student assignments based on key criteria including associated course modules, conceptual relevance, assignment type, whether programming or theoretical, and performance grade, whether pass or fail.

5. **Organizing a file system based on topics, dates, and geographical locations ($\mathcal{S}_2$).** Similar to the preceding user, the primary objective of this implementation was to efficiently structure a file system. Consequently, harnessing both the FS2KG and the configuration system, user created a Knowledge Graph that associates and categorizes my collection of images and videos based on the topic, the date or the geographical location. Main *entities* that were used in the KG: (a) Image, (b) Video, (c) Location, (d) Topic/Theme, (e) Person. Some *relationships* between these entities: (1) captured at (Image- Location), (2) filmed at (Video - Location), (3) associated with (Location - Topic/Theme), (4) photographed by (Image - Person),

(5) related to (Image or Video - Topic/Theme), (6) recorded by (Video - Person).

6. **Establishing connections within a file system housing scientific reports ($\mathcal{S}_2$).** Like previous users, this student employed both FS2KG and the configuration procedure to make an infrastructure that could be prove helpful in a meta-analysis based on data contained in articles. Precisely, a number of files were organized in a folder hierarchy, while adding some configuration files and rules in order to create the KG.

7. **Extracting information related to enzymes from a file system ($\mathcal{S}_1$, $\mathcal{S}_2$).** A user recognized the need of creating an infrastructure to handle information that could be prove helpful in a meta-analysis, concerning a hypothesis the structure-function relations of enzymes, based on data contained in databases, and therefore in bibliography. In this context, the term "file system based" suggests that the data extraction process involves accessing and retrieving information stored within files or directories on a computer's file system while configuring each folder based on our configuration process.

5.2 Evaluation results

5.2.1 Participants

We extended invitations via email to individuals, inviting them to participate voluntarily in the evaluation process. Every participant was handed with a Google Form link, where they were requested to perform designated tasks and anonymously complete a pre-prepared questionnaire. In the initial scenario (A), there were 20 participants, while in the subsequent two scenarios (B,C), there were 10 participants each. The chosen number of evaluators met our requirements effectively, as indicated in [9], where it is suggested that employing 20 evaluators is adequate for identifying over 95% of the usability problems within a user interface.

In scenario A, users were provided with a brief PowerPoint presentation of the FS2KG-explorer which gave them an overview of the app. The participants were equally split to female (50%) and male (50%) 5.3, with ages ranging from 22 to 56 years. The distribution appears nearly uniform, with the exception of the age 22, which stands out as the most prevalent, accounting for 35% of the distribution, as illustrated (see 5.5). Comprehensively, 50% of the evaluators were UOC-CS561 students, 20% of them Computer Science Professionals, and 10% each for the rest 3 categories (ELMEPA student, Other Computer Science-Engineering student, Not computer science/engineering related) (see 5.1).

For the second scenario (B) evaluators were given a short demo paper of FS2KG offering them a more detailed and comprehensive view of the application. The evaluators that participated were 10, 60% of them were male and 40% female (see 5.4), with ages ranging from 22 to 25 years (see 5.6). Based on the age distribution in this scenario, it is indicated that it was handed to more experienced users and mostly to students with Semantic Web knowledge. Precisely, 90% of the evaluators were UOC-CS561 students and 10% of them Other Computer Science-Engineering students (see 5.2).

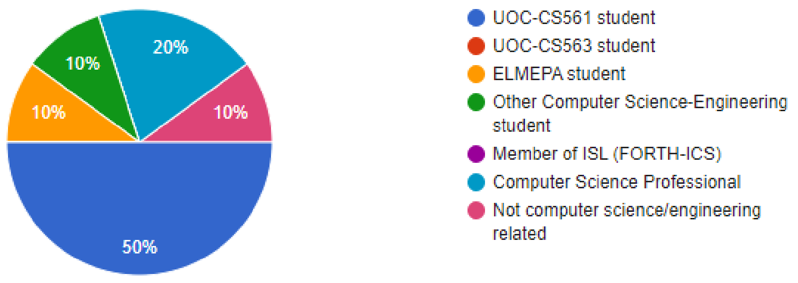


Figure 5.1: Scenario A: Evaluators academic/working background chart

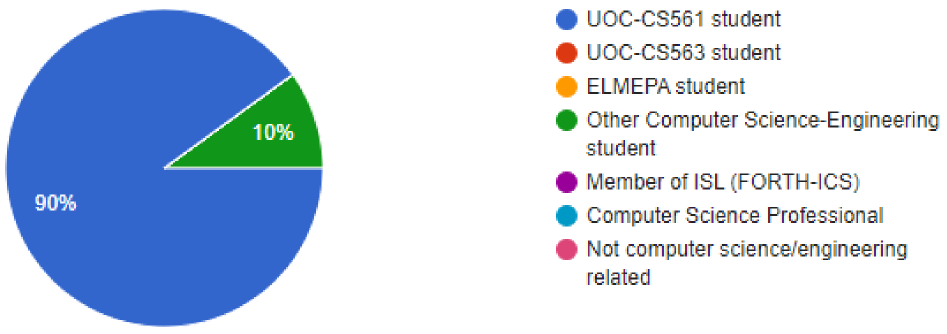


Figure 5.2: Scenario B: Evaluators academic/working background chart

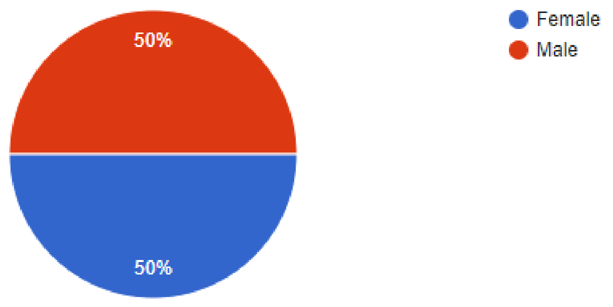


Figure 5.3: Scenario A: Evaluators gender chart

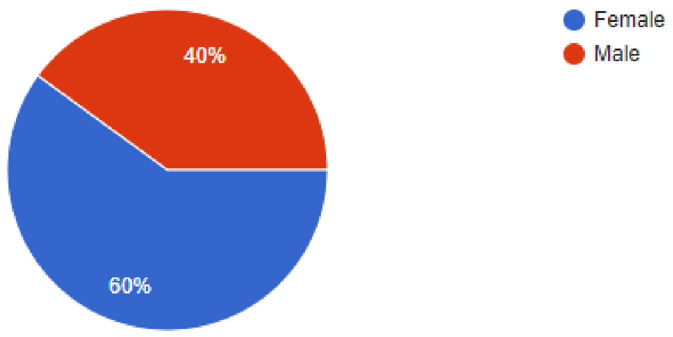


Figure 5.4: Scenario B: Evaluators gender chart

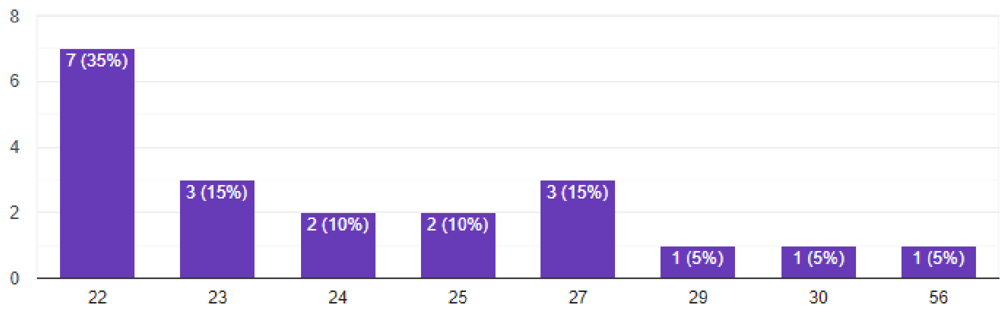


Figure 5.5: Scenario A: Evaluators age distribution

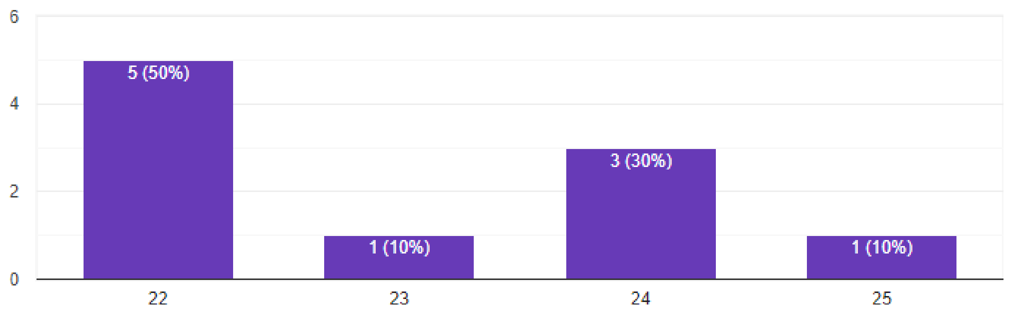
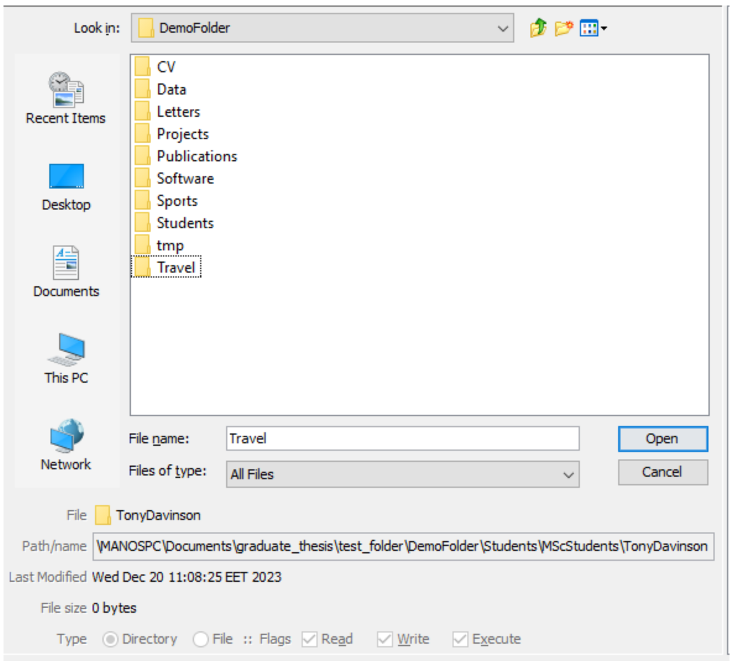


Figure 5.6: Scenario B: Evaluators age distribution

5.2.2 Scenario A

TA1 *Use FS2KG-Explorer to navigate into the folder Evaluation/DemoFolder that you have created. How many (direct) subfolders it contains?*: 10 (**80%**), 11 (**10%**), 22 (**5%**), 49 (**5%**)

The first task in this scenario was a simple interaction with the application. We asked users to navigate through the app, find and note how many direct sub-folders the demo folder, that they were given, contains. Most of the users, replied with the number of 10 folders which is the correct answer (see 5.7). Specifically 80% of the users answered 10 folders, 10% of the users answered 11 folders and 5% each answered 22 and 49 folders (see 5.8). There was probably a misunderstanding, as users reported more than 10 folders, on the word **direct** which actually meant only inside the DemoFolder and not its sub-folders too.


Look in: DemoFolder

CV
Data
Letters
Projects
Publications
Software
Sports
Students
tmp
Travel

File name: Travel
Files of type: All Files

File TonyDavinson
Path/name \MANOSPC\Documents\graduate_thesis\test_folder\DemoFolder\Students\MScStudents\TonyDavinson
Last Modified Wed Dec 20 11:08:25 EET 2023
File size 0 bytes
Type ○ Directory ○ File :: Flags ✓ Read ✓ Write ✓ Execute

Figure 5.7: Scenario A: TA1 direct sub-folders

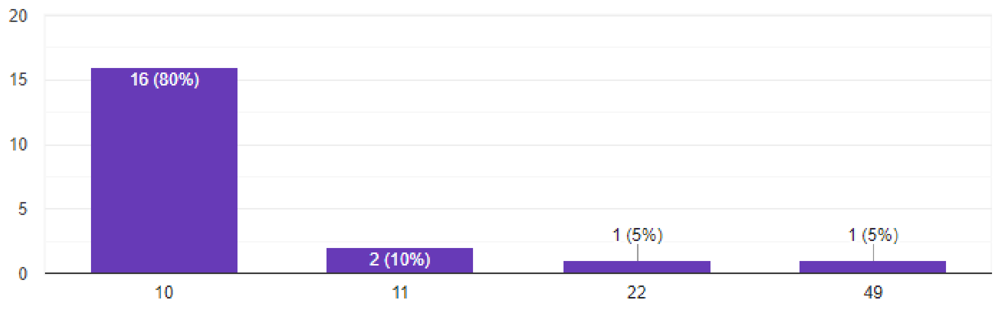


Figure 5.8: Scenario A: TA1 answers

TA2 *Is there any folder named TonyDavidson inside the DemoFolder? If so which is its parent folder?*: MScStudents (**75%**), RecommendationLetters and MScStudents (**25%**)

After that, users had to search for a specific folder (TonyDavidson) and write down its parent folder. In detail 5.9, the 75% of the users answered one folder (MScStudents) as its parent while 25% answered two folders (Recommendation-Letters and MScStudents). Apparently, some users tried to explore and find every folder with that name, most of them followed a logical path through the students folder and no users reported RecommendationLetters as the only parent folder. Also, there wasn't any user that had a difficulty to locate the wanted folder.

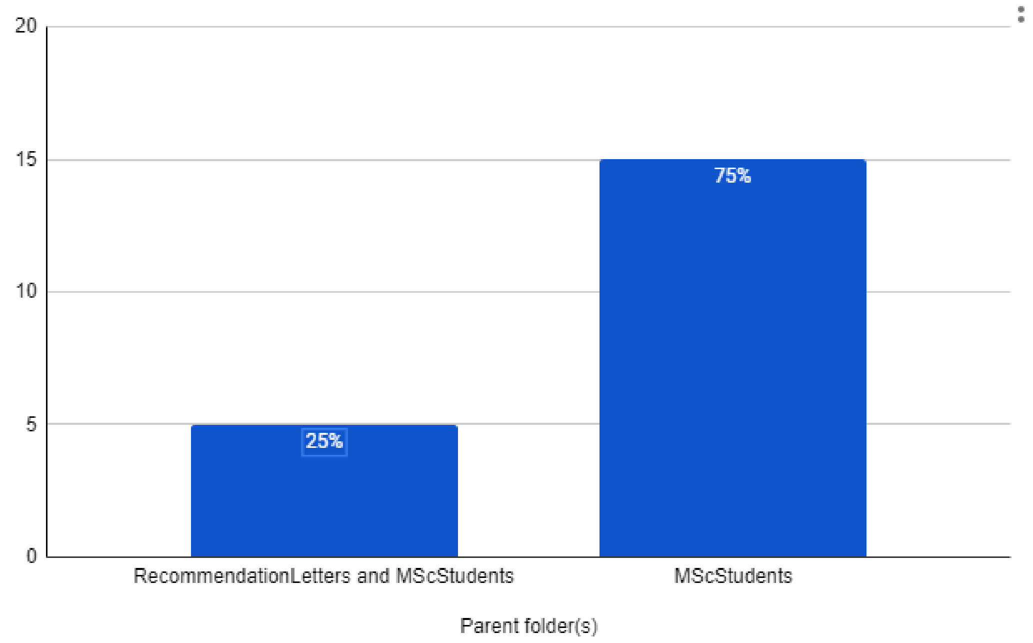


Figure 5.9: Scenario A: TA2 answers

TA3 *Provide the class of the TonyDavidon entity*: Student (**80%**), TonyDavinson (**5%**), Student, Alumni (**5%**), Smyrnakis (**5%**)

The third task, asked users to provide some info for the specific folder (Tony-Davidson). Precisely, users had to discover and note the class of the specific entity. Providing a detailed breakdown in 5.10, the majority of users (80%) identified a single class (Student) as the entity's class, a user provided two classes for the same entity 5.11 (5%) whereas the rest of the users gave inconclusive answers. An observation here is that most users answered with a single class following the question they were given to provide the class of Tony Davidson.

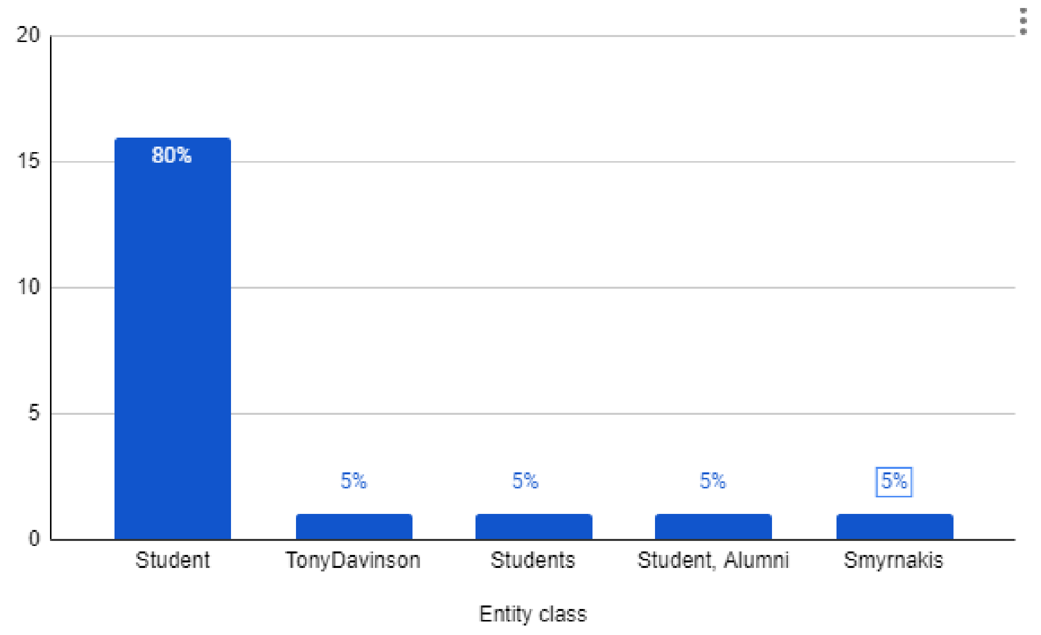
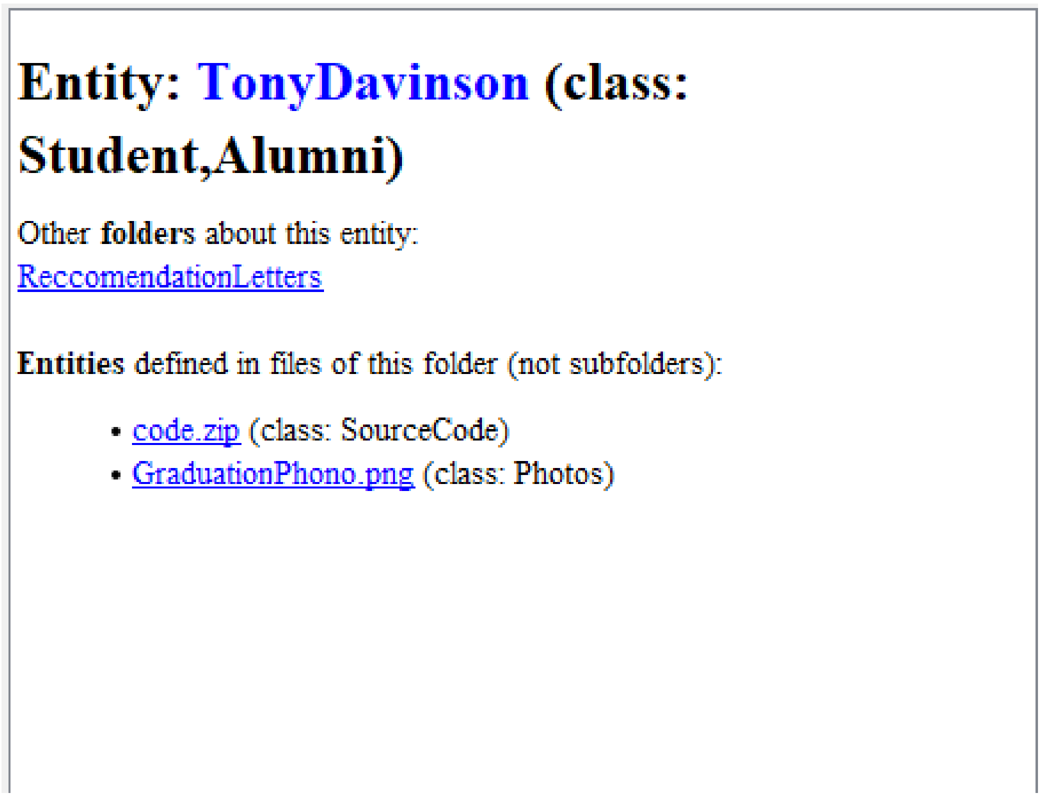


Figure 5.10: Scenario A: TA3 answers



Entity: TonyDavinson (class: Student,Alumni)

Other **folders** about this entity:
ReccomendationLetters

Entities defined in files of this folder (not subfolders):

- code.zip (class: SourceCode)
- GraduationPhono.png (class: Photos)

Figure 5.11: Scenario A: TA3 multiple classification entity

TA4 *How many folders about the entity TonyDavidson exist?*: 2 (**70%**), 1 (**25%**), 4 (**5%**)

Based on the second task, users were given the insight through the forth task to record how many folders of the specific entity (TonyDavidson) exist. In the analysis presented in 5.12, the primary trend reveals that 70% of users indicated that two folders exist with that entity, the 25% opted for one folder while the rest 5% recorded an erroneous number of folders. As is evident, users followed the question and tried to identify more than one folders with that entity defined. Compared to the second task on the users that answered two folders, we can see that there was a 45% raise.

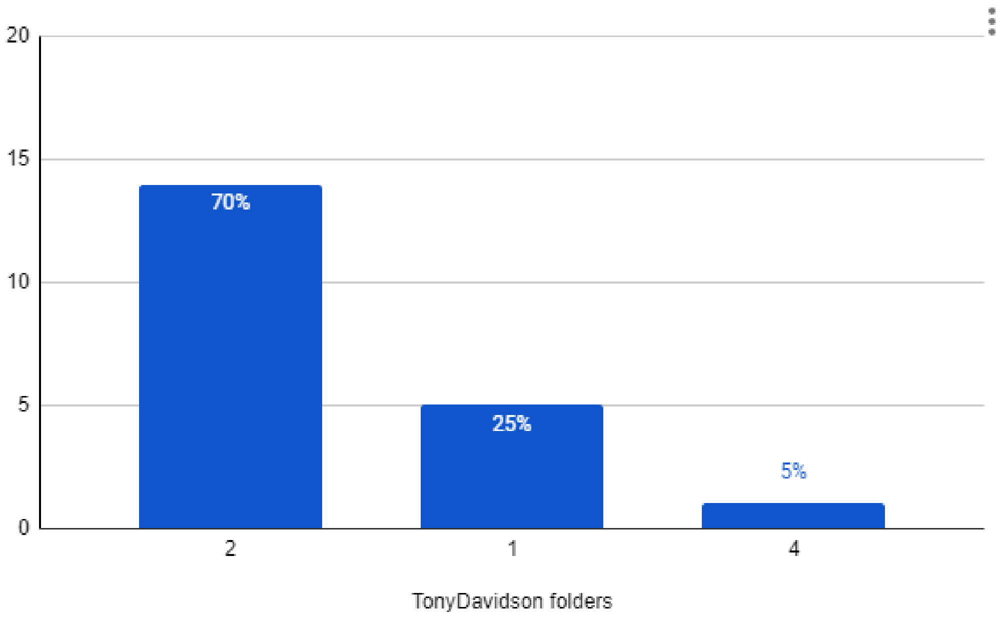


Figure 5.12: Scenario A: TA4 answers

TA5 *Which other entities did you find in the TonyDavidson directory, and which are their classes? (If there are any)*: code.zip (class: SourceCode) , GraduationPhono.png (class: Photos) (**80%**), none (**20%**)

The fifth task asked users to identify and list the entities found within the TonyDavidson directory, along with their corresponding classes if applicable. This task had a dual purpose as users had to explore the different folders that the entity was found and then record the entities that were stored there. In the chart in 5.13, the predominant pattern suggests that 80% of users acknowledged the presence of two entities defined in the TonyDavidson directory, recording their names and classes, while the rest 20% of users could not find any entities. We can safely say that there is an explanation for this finding as all of the users that recorded the

TonyDavidson in two folders also noted the entities defined in one of the folders (see 5.14).

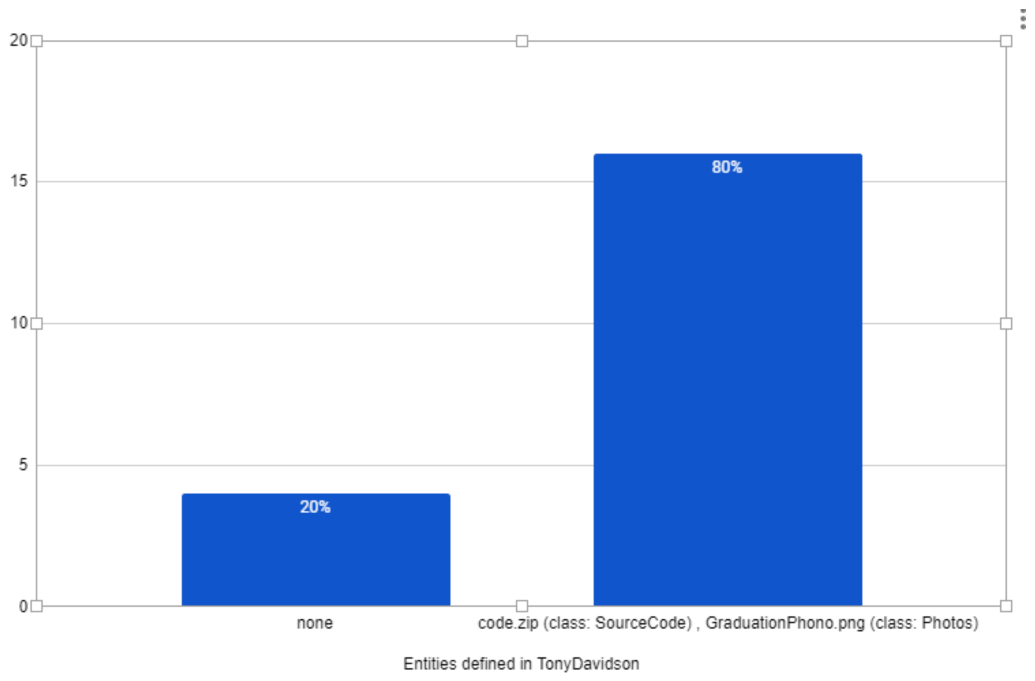
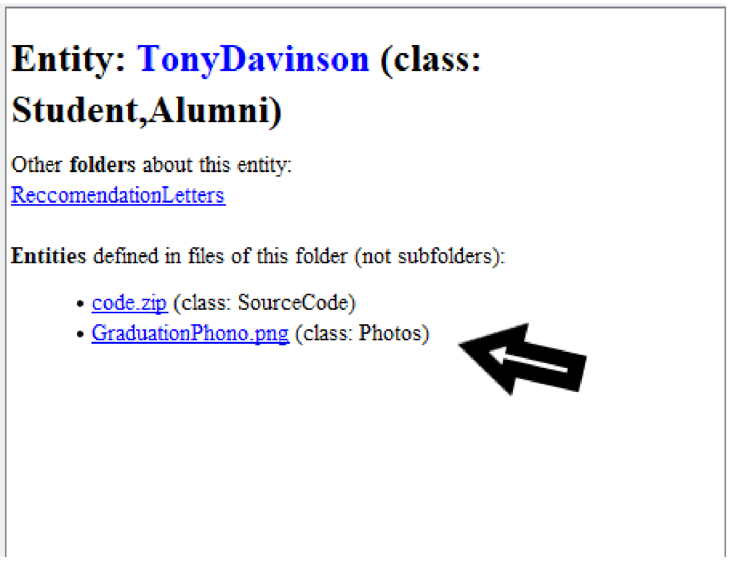


Figure 5.13: Scenario A: TA5 answers



Entity: TonyDavinson (class: Student,Alumni)

Other **folders** about this entity:
ReccomendationLetters

Entities defined in files of this folder (not subfolders):

- code.zip (class: SourceCode)
- GraduationPhono.png (class: Photos)

Figure 5.14: Scenario A: TA5 entities defined in TonyDavidson

TA6 *Navigate to the folder Data/2-LocationsTaxonomy and find how many European countries are listed* : 8 (**70%**), 14 (**15%**), 13 (**10%**), 12 (**5%**),

Finally the last task in scenario A was to navigate to a specific file (LocationsTaxonomy.txt) and explore the contents of this file. Particularly, users were asked to discover the definedInFile functionality and were asked to record how many countries are defined in this file. In the bar chart presented in 5.15, a prevailing trend emerges, indicating that 70% of users recognized the existence of two 8 countries defined within the LocationsTaxonomy file. These users successfully recorded the number of countries. Conversely, the 15% of users identified 14 countries, 10% identified 13 countries and 5% noted 12 countries. It seems that users that recorded more than 8 countries tried to record generally locations as the file is consisted of these entities (see 5.16).

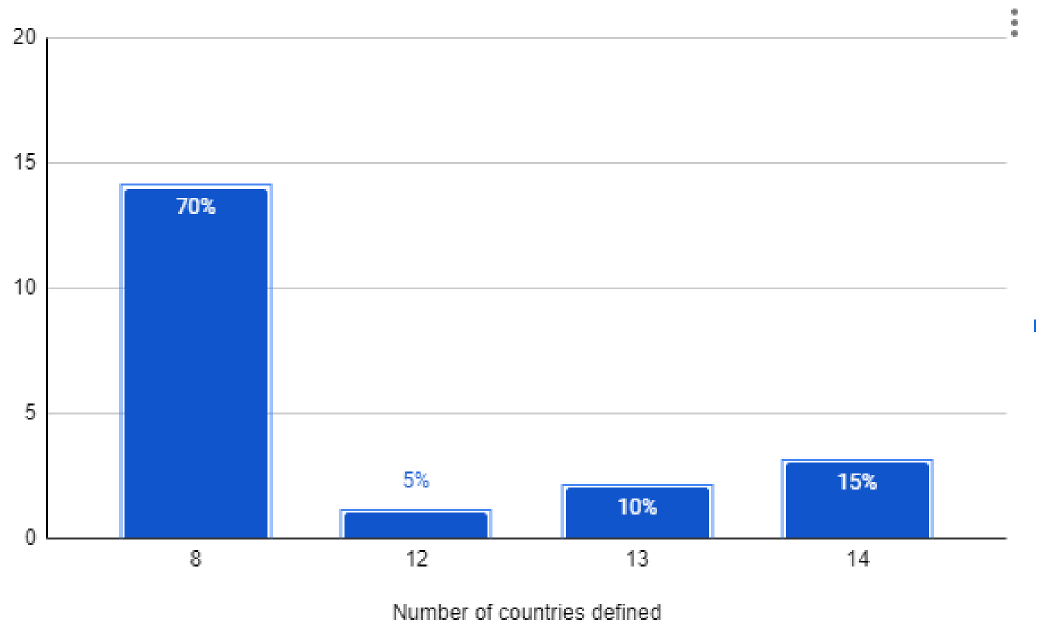
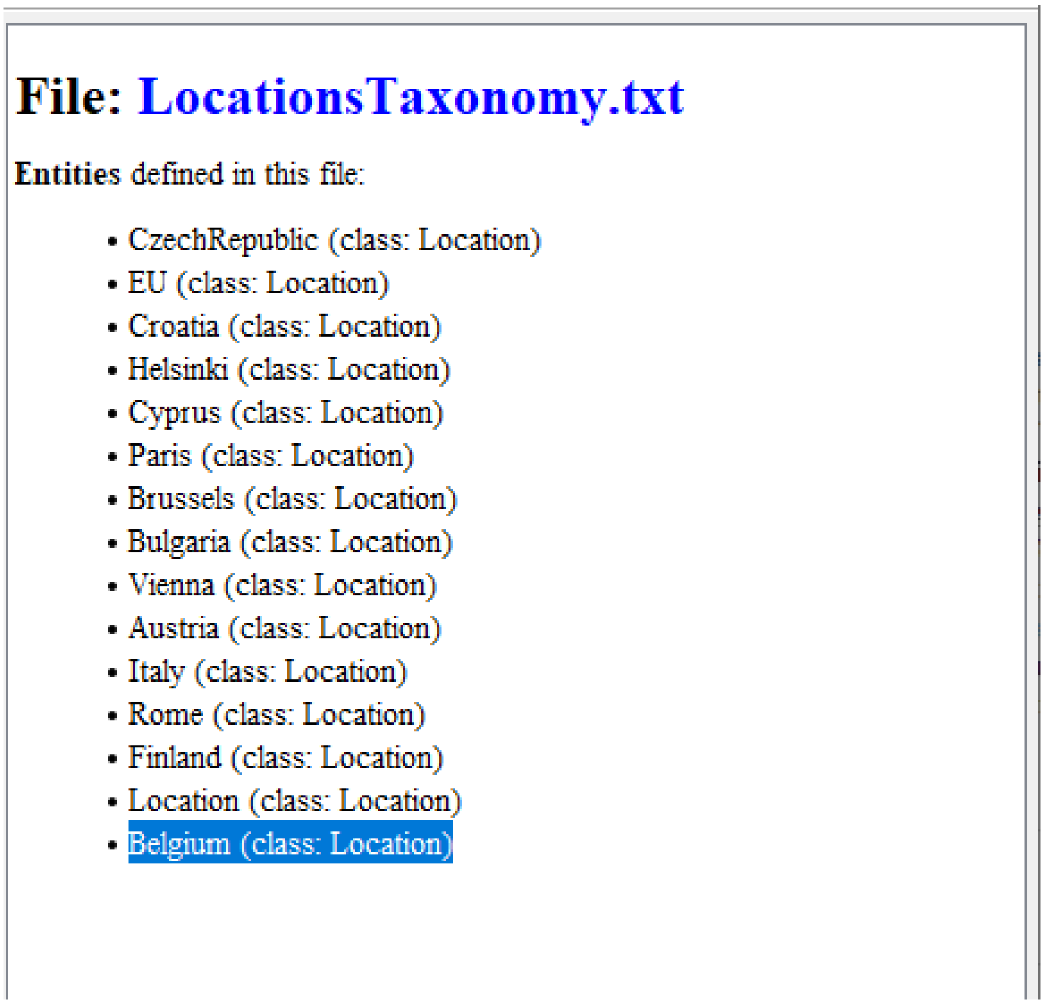


Figure 5.15: Scenario A: TA6 answers

5.2.2.1 Results summary

As regards task performance, all users responded to the questions of the tasks. Most of them were correct. There were some variations, mainly because of possible misunderstandings on the task questions. An observation is that many users provide comprehensive explanations on their answers. This is mainly caused by the fact that users are not confident enough about their answer as they have not experienced a similar application before. Apart from that, the metrics regarding the answers given show that transitioning from one question to the next, users become more familiar with the application.



File: LocationsTaxonomy.txt

Entities defined in this file:

- CzechRepublic (class: Location)
- EU (class: Location)
- Croatia (class: Location)
- Helsinki (class: Location)
- Cyprus (class: Location)
- Paris (class: Location)
- Brussels (class: Location)
- Bulgaria (class: Location)
- Vienna (class: Location)
- Austria (class: Location)
- Italy (class: Location)
- Rome (class: Location)
- Finland (class: Location)
- Location (class: Location)
- Belgium (class: Location)

Figure 5.16: Scenario A: TA6 locations taxonomy file

5.2.3 Scenario B

TB1-a *Navigate to the Data/2-LocationsTaxonomy and add Paris. What did you have to write and where?*: Paris;France France;EU (**60%**), Paris;EU (**30%**), Paris;France (**10%**),

The first task in scenario B asks users to try to locate a specific folder and add an entity inside the file. In detail, users had to add data inside the LocationsTaxonomy.txt about Paris. Most of the users, replied with the 'Paris;France France;EU' rows which is the most accurate answer following the pattern and the configuration tutorial they were given. Specifically 60% of the users answered correctly, 30% of the users answered 'Paris;EU' and 10% of the users answered 'Paris;France' (see 5.17). All of the answered questions can be valid as in every question the data will be linked correcty and Paris will be visible in the semantic info panel.

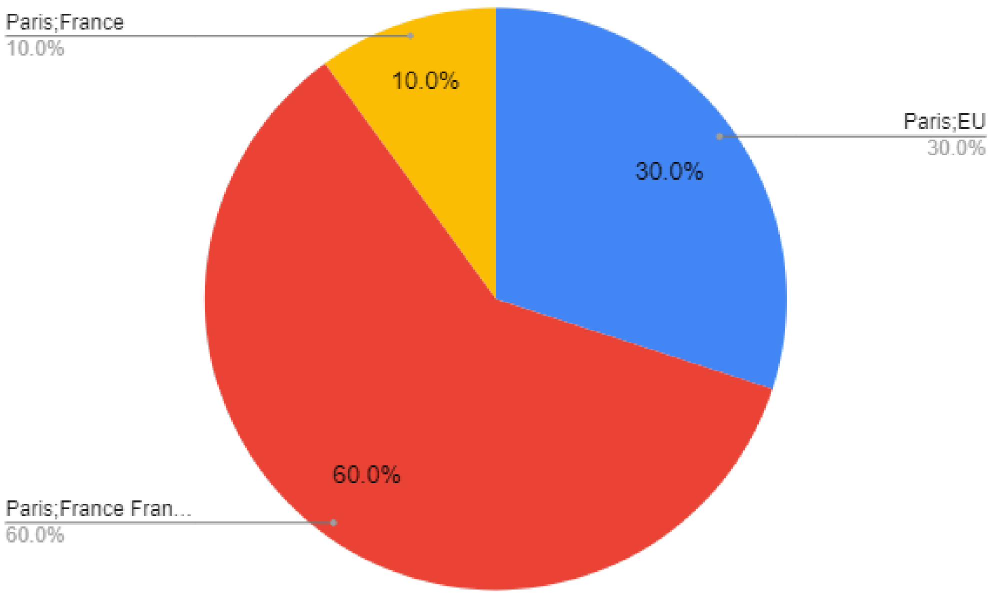
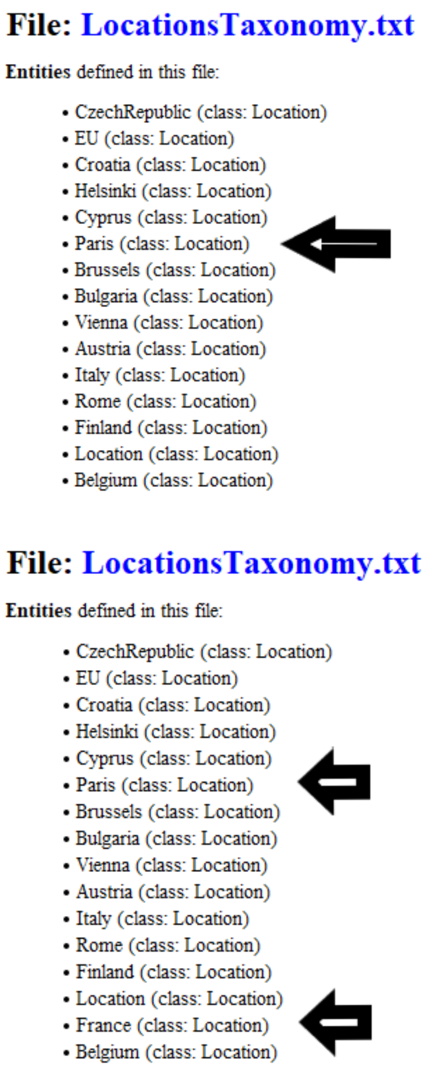


Figure 5.17: Scenario B: TB1-a answers

TB1-b *Using FS2KG-Explorer, create the KG and navigate to Data/2-LocationsTaxonomy. Can you see Paris as entity when clicking the LocationsTaxonomy.txt?*: Yes (**100%**),

Based on first task users had to navigate through the FS2KG-explorer to confirm that their answer is semantically correct. All of the users were able to successfully identify Paris as a defined entity in the file through the app. However, 70% of users could also identify France as an entity and a Location subclass as they added it separately (France;EU) or in a row (Paris;France) (see 5.18).



File: LocationsTaxonomy.txt

Entities defined in this file:

- CzechRepublic (class: Location)
- EU (class: Location)
- Croatia (class: Location)
- Helsinki (class: Location)
- Cyprus (class: Location)
- Paris (class: Location)
- Brussels (class: Location)
- Bulgaria (class: Location)
- Vienna (class: Location)
- Austria (class: Location)
- Italy (class: Location)
- Rome (class: Location)
- Finland (class: Location)
- Location (class: Location)
- Belgium (class: Location)

File: LocationsTaxonomy.txt

Entities defined in this file:

- CzechRepublic (class: Location)
- EU (class: Location)
- Croatia (class: Location)
- Helsinki (class: Location)
- Cyprus (class: Location)
- Paris (class: Location)
- Brussels (class: Location)
- Bulgaria (class: Location)
- Vienna (class: Location)
- Austria (class: Location)
- Italy (class: Location)
- Rome (class: Location)
- Finland (class: Location)
- Location (class: Location)
- France (class: Location)
- Belgium (class: Location)

Figure 5.18: Scenario B: TB1-b Difference between data given

TB2-a *Enrich the .kg file at the folder Sports so that all subfolders are classified to a class example:Sport. What did you have to write and where?* : subFoldersClass=example:Sport (**80%**), subFoldersClass=example:Sport readme=on (**20%**),

In the third task, users were instructed to write a specific configuration inside the Sports folder in the given Demo Folder. In more details, users were asked what is the configuration and where it should be written so every sub-folder inside the Sports folder to be classified as example:Sport. All of the users, answered that they had to create a .kg file inside the Sports folder and 80% of them wrote 'subFoldersClass=example:Sport' while the rest 20% wrote 'subFoldersClass=example:Sport readme=on' (see 5.19). Both the answers were right and users were provided the same semantic info.

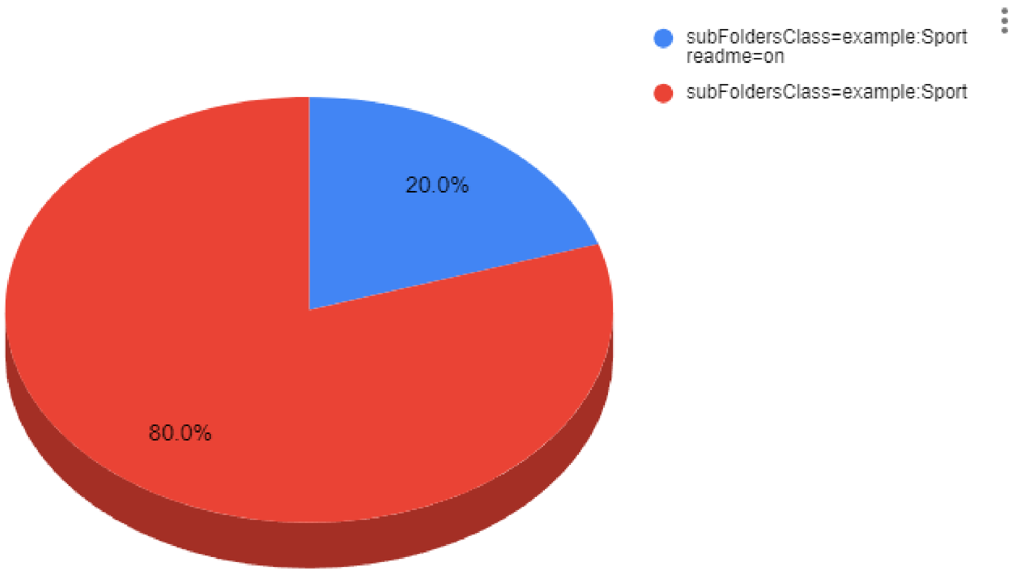


Figure 5.19: Scenario B: TB2-a answers

TB2-b *Using FS2KG-Explorer, create the KG. Open with Protege the produced ttl. Is the class example:Sport and its two instances defined properly? Write down the instances.* : Football and Tennis (**70%**), Baseball,Billiards,Football,Running,Tennis (**30%**),

Similarly with the first two questions, we asked users to make an observation based on the previous task. Users were instructed to create the Knowledge Graph after the configuration they added and to use the produced .ttl file as input in the knowledge management tool, Protégé. Precisely, the question required users to confirm that the classification is correct and to identify the two instances they noticed within the application. The 70% of the users indicated two instances 'Football and Tennis' while the rest 30% of them wrote down all of the instances ('Baseball,Billiards,Football,Running,Tennis') that were classified as example:Sport (see 5.20).

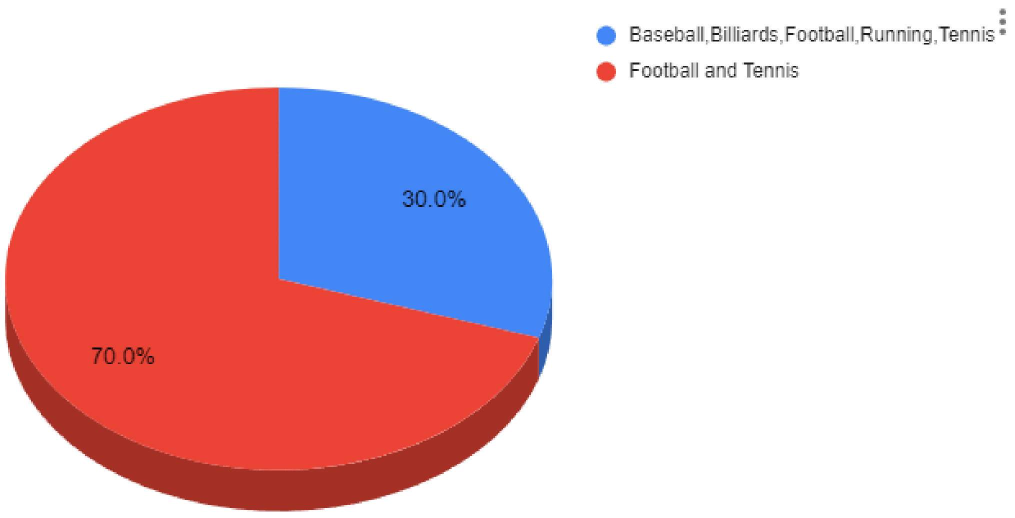


Figure 5.20: Scenario B: TB2-b answers

TB3-a *Enrich the folder Software with a subfolder "tool2", write inside that folder a Readme file, and create a .kg file so that the contents of the readme file become rdfs:comment of tool2. What did you have to write and where?*: Readme.txt and readme=on (**70%**), readme=on (**30%**),

The fifth task directed users to write a specific configuration inside the Software folder and to create a new entity. In greater detail, users were prompted to specify the configuration and indicate the location for its entry so a 'tool2' entity is created and has an rdfs:comment property. Every user, answered that they had to create a sub-folder inside the Software folder and .kg file inside that sub-folder with the 'readme=on' rule inside the file. However only the 70% of them were right as they created a README.txt file in which is the comment stored (see 5.21).

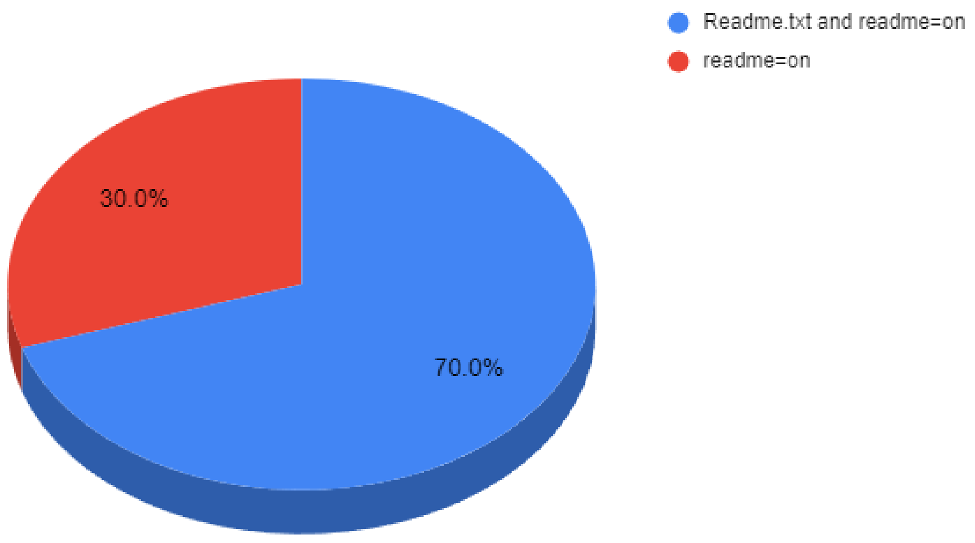


Figure 5.21: Scenario B: TB3-a answers

TB3-b *Using FS2KG-Explorer navigate to tool2 and ensure that the comment is visible at the right frame. Is that visible?*: Yes (**70%**), No (**30%**),

In reference to the fifth task, users were questioned if the comment is visible at the semantic info panel. Although users created the correct configuration rule inside the .kg ,as mentioned earlier, some of the users (30%) did not create a README file with the contents of the comment (see 5.22).

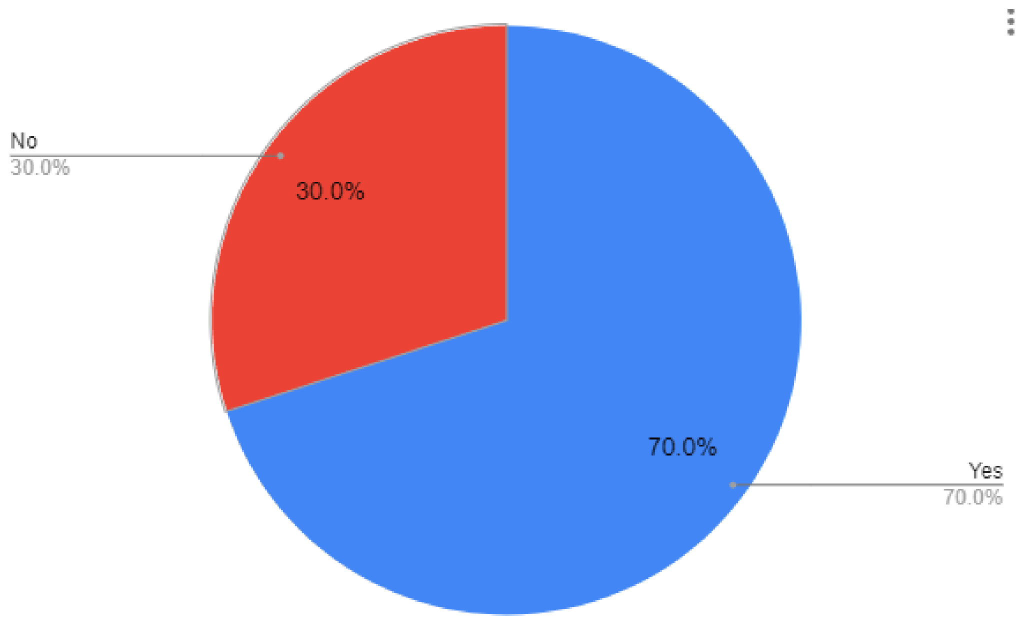


Figure 5.22: Scenario B: TB3-b answers

TB4-a *Place in the subfolder "tool2", a "main.java" file and classify it with the class "example:Java". What did you have to write and where?* : C0=example:Java inside a main.kg (**60%**), extraTriples = main.java rdf:type example:Java; (**30%**), C1=example:Java inside a main.kg (**10%**)

Another task users had to complete was to create a new file and the proper configuration to make this file an entity under a specific class. In particular, users were requested to create a new Java file inside the previously defined directory (tool2) and add the configuration needed so the file is classified as an 'example:Java' entity. A large portion of the users (60%), created a new main.kg file and inserted the 'C0=example:Java' rule. However, some users (30%) found a different way to classify this file as Java entity, adding a rule inside the existing .kg file using the extra triples rule: 'extraTriples = main.java rdf:type example:Java;'. The rest (10%) of the users inserted a faulty configuration rule (see 5.23).

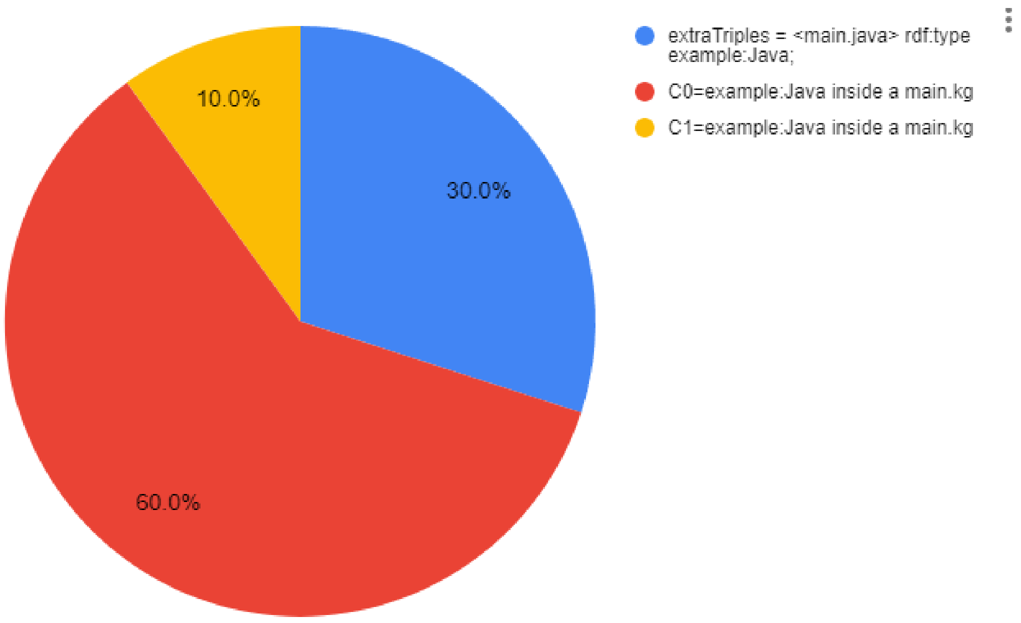


Figure 5.23: Scenario B: TB4-a answers

TB4-b *Using FS2KG-Explorer navigate to tool2. Is the file main.java and its class visible at the right frame?*: Yes (**90%**), No (**10%**)

Finally, users had to answer whether the class of the added java file is visible on the semantic info panel. Following the stats collected earlier on how users tried to classify the main.java file, we can see that both the ways work effectively and users observed the file as an example:Java entity (see 5.24).

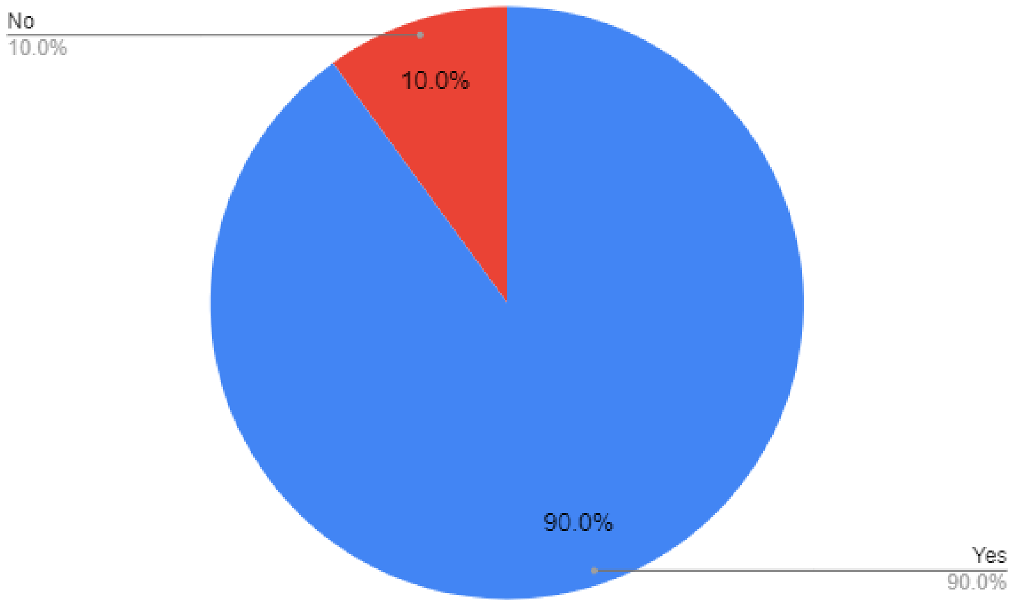


Figure 5.24: Scenario B: TB4-b answers

5.2.3.1 Results summary

Similarly, as in scenario A, regarding the task performance all users responded to the questions of the tasks. The majority were accurate, with a few discrepancies arising primarily due to potential misunderstandings regarding the task questions. However, the discrepancies were much more less than the first scenario as this task was delegated to users more proficient in Semantic Web, involving more intricate tasks. An noteworthy observation is the diversity in approaches that users employed while addressing the question. Other than that, following the same behaviour as in scenario A, the provided answers indicate that as users progress from one question to the next, they grow more acquainted with the application.

5.2.4 User Ratings

Each scenario was accompanied by a user rating questionnaire, in which users could express their opinion on the application and the configuration system.

Q1 *How would you rate the usability of FS2KG-Exlorer?*: Very user friendly (**40%**), User friendly (**60%**), Not user friendly (**0%**), Very difficult to use (**0%**)

In greater detail, the first scenario was combined with a questionnaire regarding the application's usability. Most of the users described the application user friendly and the rest very user friendly (see 5.25).

Q2 *How would you rate the process of the configuration?*: Very user friendly (**40%**), User friendly (**60%**), Not user friendly (**0%**), Very difficult to use (**0%**)

The second scenario included a questionnaire which asked about the configuration procedure. Most of the users described the application user friendly, some very user friendly while there were some users that found the whole procedure not user friendly (see 5.26).

Additionally, both of the scenarios provided a form for reporting any errors,problems and recommendations. In the first scenario, there were some suggestions to improve the user interface when listing entities defined in a file, while there were some comments on the functionality of the double click on the main grid. Most of the comments were positive and some users indicated that there was nothing to add or report.

Similarly, in the second scenario there were some improving comments on the user interface on the main grid. There were some users that indicated an improvement on how a user can run the whole procedure of the configuration and the Knowledge Base creation from the beginning. Finally, a user proposed a refresh/update button which would update the UI after any change.

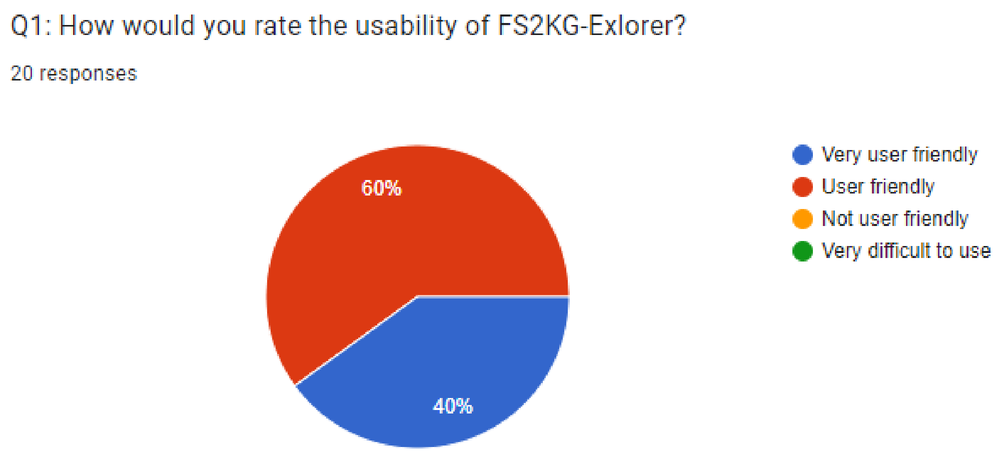


Figure 5.25: Scenario A: User ratings

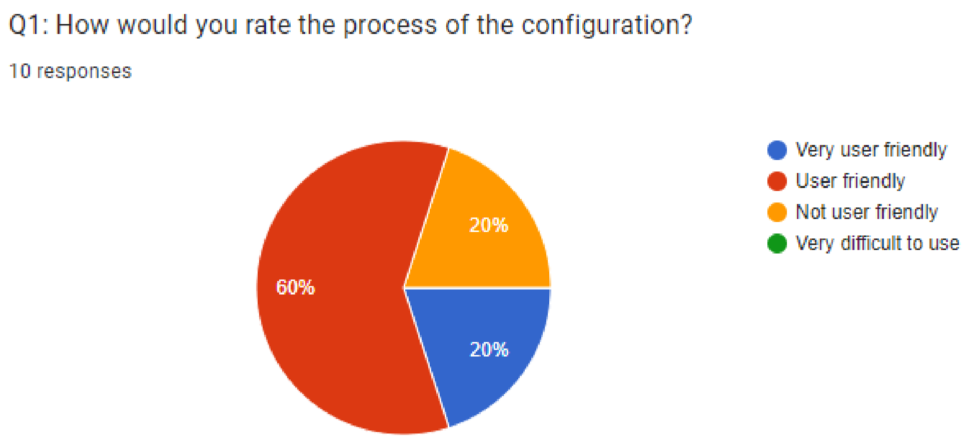


Figure 5.26: Scenario B: User ratings

Chapter 6

Concluding Remarks

Finding an effective method to conciliate freedom of file system usage, and Knowledge Graph integrity and usability, is a challenging task. We proposed a modular configuration approach relying on small configuration files in the folders, and KG reconstruction at any moment. We introduced a configuration language, and we have showcased its feasibility, and flexibility that it offers. The approach supports a default operation that requires no configuration, scope restriction directives, automatic creation and classification of entities corresponding to subfolders, leveraging of 'readme' files, easily configurable data extraction and transformation from the desired csv files, provenance of the mined entities, a convenient method to add arbitrary metadata, as well as a light weight query client.

We showcased the feasibility of the approach by an implementation through the tool `FS2KG` we have reported experimental and empirical results from using it (over real file systems). `FS2KG` effortlessly combines conventional file management features with cutting-edge semantic technologies. By allowing user configuration and leveraging a knowledge graph, the application enables semantic querying, creating an intelligent and context-aware tool for users.

We conducted a careful and intentionally limited task-based evaluation with end-users. The main goals of this evaluation were twofold: first, to gauge users' proficiency and comfort with the interaction paradigm introduced in our system, and second, to gather valuable feedback for improving both the graphical user interface (GUI) and the underlying procedural framework. The majority of ratings and comments were overwhelmingly positive, with users expressing a strong inclination to utilize our work for effectively organizing their file systems.

We encouraged users to apply the approach to their individual file systems to acquire a functionality tailored to their needs. They had the freedom to suggest and implement whatever they deemed useful. Most users utilized both the FS2KG explorer and the configuration procedure to construct the knowledge graph based on their needs while some users suggested enhancements or extensions. We collected many ideas that are worth further work and research, on possible extensions and improvements both on User Interface and functionality.

Returning to the questions of the introductory section, as regards $Q1$ we have seen that we can enrich the capabilities of file systems with methods that spot and connect entities in different folders. As regards $Q2$, we have seen that we can construct various forms of KGs (from taxonomies to arbitrary triples by extraction and transformation rules) very easily. Finally we proposed various possible extensions and combinations with other tools. One immediate next step is the implementation of an explorer that combines the functionality of the classical file explorer with `FS2KG`-Q. We wish OS and file systems to start incorporating this functionality in their forthcoming versions.

Bibliography

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[2] GA Barnard III and Louis Fein. Organization and retrieval of records generated in a large-scale engineering project. In *Papers and discussions presented at the December 3-5, 1958, eastern joint computer conference: Modern computers: objectives, designs, applications*, pages 59–63, 1958.

[3] Stephan Bloehdorn, Olaf Görlitz, Simon Schenk, Max Völkel, et al. Tagfs-tag semantics for hierarchical file systems. In *Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06), Graz, Austria*, volume 8, pages 6–8, 2006.

[4] Robert C Daley and Peter G Neumann. A general-purpose file system for secondary storage. In *Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I*, pages 213–229, 1965.

[5] Jesse David Dinneen and Charles-Antoine Julien. The ubiquitous digital file: A review of file management research. *Journal of the Association for Information Science and Technology*, 71(1):E1–E32, 2020.

[6] Jesse David Dinneen, Charles-Antoine Julien, and Ilja Frissen. The scale and structure of personal file collections. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[7] Laura Drăgan and Stefan Decker. Knowledge management on the desktop. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 373–382. Springer, 2012.

[8] Pavlos Fafalios, Konstantina Konsolaki, Lida Charami, Kostas Petrakis, Manos Paterakis, Dimitris Angelakis, Yannis Tzitzikas, Chrysoula Bekiari, and Martin Doerr. Towards semantic interoperability in historical research: Documenting research data and knowledge with synthesis. In *International Semantic Web Conference*, pages 682–698. Springer, 2021.

[9] Laura Faulkner. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35:379–383, 2003.

[10] Bernhard Haslhofer and Antoine Isaac. data. europeana. eu: The europeana linked open data pilot. In *International Conference on Dublin Core and Metadata Applications*, pages 94–104, 2011.

[11] Aidan Hogan. The semantic web: Two decades on. *Semantic Web*, 11(1):169–185, 2020.

[12] Charlotte Jenkins, Mike Jackson, Peter Burden, and Jon Wallis. Automatic RDF metadata generation for resource discovery. *Computer Networks*, 31(11-16):1305–1320, 1999.

[13] Christian Jilek, Markus Schröder, Sven Schwarz, Heiko Maus, and Andreas Dengel. Context spaces as the cornerstone of a near-transparent and self-reorganizing semantic desktop. In *European Semantic Web Conference*, pages 89–94. Springer, 2018.

[14] Mikko Koho, Esko Ikkala, Petri Leskinen, Minna Tamper, Jouni Tuominen, and Eero Hyvönen. Warsampo knowledge graph: Finland in the second world war as linked open data. *Semantic Web – Interoperability, Usability, Applicability*, 2020. In press.

[15] Yannis Marketakis, Makis Tzanakis, and Yannis Tzitzikas. Prescan: towards automating the preservation of digital objects. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, pages 404–411, 2009.

[16] Yannis Marketakis, Yannis Tzitzikas, Aureliano Gentile, Bracken van Niekerk, and Marc Taconet. On the evolution of semantic warehouses: The case of global record of stocks and fisheries. In *Research Conference on Metadata and Semantics Research*, pages 269–281. Springer, 2020.

[17] Syed Rahman Mashwani, Azhar Rauf, Shah Khusro, and Saeed Mahfooz. Linked file system: Towards exploiting linked data technology in file systems. In *2016 International Conference on Open Source Systems & Technologies (ICOSST)*, pages 135–141. IEEE, 2016.

[18] Franck Michel, Fabien Gandon, Valentin Ah-Kane, Anna Bobasheva, Elena Cabrio, Olivier Corby, Raphaël Gazzotti, Alain Giboin, Santiago Marro, Tobias Mayer, et al. Covid-on-the-web: Knowledge graph and services to advance covid-19 research. In *International Semantic Web Conference*, pages 294–310. Springer, 2020.

[19] Christos Nikas, Giorgos Kadilierakis, Pavlos Fafalios, and Yannis Tzitzikas. Keyword search over RDF: Is a single perspective enough? *Big Data and Cognitive Computing*, 4(3):22, 2020.

[20] Leo Sauermann, Ansgar Bernardi, and Andreas Dengel. Overview and outlook on the semantic desktop. In *Semantic Desktop Workshop*, volume 175. Citeseer, 2005.

[21] Leo Sauermann and Dominik Heim. Evaluating long-term use of the gnowsis semantic desktop for pim. In *International Semantic Web Conference*, pages 467–482. Springer, 2008.

[22] Leo Sauermann, Ludger Van Elst, and Andreas Dengel. PIMO - a framework for representing personal information models. *Proceedings of I-Semantics*, 7:270–277, 2007.

[23] Bernhard Schandl. SemDAV: a file exchange protocol for the semantic desktop. In *SemDesk'06: Proceedings of the 5th International Conference on Semantic Desktop and Social Semantic Collaboration*, November 2006.

[24] Bernhard Schandl. Representing linked data as virtual file systems. In *Proceedings of the WWW'2009 Workshop on Linked Data on the Web, LDOW 2009*, 2009.

[25] Bernhard Schandl and Niko Popitsch. Lifting file systems into the linked data cloud with tripfs. In *Proceedings of the WWW'2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA*, 2010.

[26] Bram Steenwinckel, Gilles Vandewiele, Ilja Rausch, Pieter Heyvaert, Ruben Taelman, Pieter Colpaert, Pieter Simoens, Anastasia Dimou, Filip De Turck, and Femke Ongenae. Facilitating the analysis of covid-19 literature through a knowledge graph. In *International Semantic Web Conference*, pages 344–357. Springer, 2020.

[27] Yannis Tzitzikas. Fs2kg: From file systems to knowledge graphs. In *Demo paper, Proceeding of ISWC 2022.*

[28] Ruben Verborgh and Miel Vander Sande. The semantic web identity crisis: in search of the trivialities that never were. *Semantic Web*, 11(1):19–27, 2020.

[29] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.