Standards Based Personal Knowledge Graphs

Omes Baltes

Abstract

Knowledge graphs have been leveraged in enterprise and open data for more than a decade, but for personal use, they are just emerging. Most of the tools for personal knowledge graphs are based on proprietary tech and data formats. This Paper explores how data-centric PKGs can be modeled using Semantic Web Standards and how to create usability focused Interfaces for them.

My gratitude goes to

Contents

List of Figures

List of Tables

List of Source Code

1 Introduction

The never-ending growth of data and information is overwhelming the natural capacities of the human brain. This phenomenon is known as the information overload problem. Personal Knowledge Graphs (PKGs) have recently garnered increasing attention from the industry as a promising solution to information overload on the individual level. The need for Individuals to structure and organize sources of knowledge has long been recognized. Over the last decades, a multitude of productivity tools have been developed to augment the abilities of the human mind. Applications for text processing, file and task management, vocabulary training, note-taking, and mind-mapping, all try to help humans manage the information and knowledge at their disposal.

In more recent years there has been a wave of tools focusing on the need to have all of this knowledge centralized and interlinked. This freely associative approach to knowledge is supposed to mimic the way humans think. Some of these tools take this as motivation to market themselves as "Tools for Thought", claiming to enable maintenance of a lifelong "Second Brain", "Personal Knowledge Graph", or "Digital Garden".

In Research, PKGs are a mostly uncharted area, based on Knowledge Graphs (KGs), and associated with Personal Knowledge Management (PKM) and Intelligence Amplification. As even the definition of Knowledge Graphs is contentious, PKGs are not clearly defined yet [7, 13]. In broad terms, PKGs are data graphs of structured Knowledge that is relevant or related to the person in question. The Notion of a PKG is an evolving concept that combines ideas from Computer Science, Information Science, Data Science, Mathematics, Psychology and Philosophy. The focus of this thesis is on the practical implications of PKGs serving as the foundation for Personal Knowledge Management.

1.1 Vision

The vision of personal knowledge management based on PKGs can be realized by building a PKG Ecosystem. This ecosystem would consist of a data model for PKGs and all the tools and technologies that interact with it. To motivate this idea, we present

examples of how the PKG Ecosystem might impact the way we interact with information, knowledge and technology. We hope this will develop an understanding for the importance of this technology, and the paradigm shifts it can enable, as well as provoke thoughts on the status quo of data ownership in technology. Consider the following examples:

Data centralization, ownership and binding. ~~Imagine~~ after changing your address you just have to update the address in one place: Your PKG. From this point onwards, anyone you shared your address with previously, will automatically retrieve the most up-to-date address. In your PKG tool, you can always see who has access to your information and revoke it at any time. Contrast this with the current situation of having to update a copy of your address across dozens of external data silos which you have limited control over.

Free association of knowledge. ~~Imagine~~ you are visiting a Wedding in Spain with some wonderful people and dancing to a Song. In a PKG you can freely associate/link all of this information, just like your brain does. You then have multiple associative retrieval paths to remember the song, because it is connected to the people, the wedding, the location and the Spain trip. You can also freely associate any information with the song, for example, make notes, which is not possible on vendor lock-ined data, that is coupled with the app, like Spotify.

Transclusion and bi-directional links. ~~Imagine~~ you have a file or website that you frequently quote from. Traditionally you would copy and paste the content, and have duplicates all over your workspace, without necessarily knowing where the information came from. In your PKG you can attach links to this information, specifying the location of the original. If the link is a pointer within your PKG, you can even update the content in any location and it will prompt you, asking if you want to update it in all other locations in real-time. Information in your PKG that is linked towards will show the locations referencing it.

1.2 Current Limitations

The increasing difficulty of Personal Knowledge Management was already identified early in the 20th century [3, 6]. The suggested approaches at the time were hindered by unsurmountable technological hurdles, which have all but disappeared by now [4]. Still, even today's Tools for PKM are not powerful enough to address information overload. Even worse, these tools have limitations that make them unfit candidates for lifelong Personal Knowledge Management:

Proprietary technology and vendor lock-in. They are based on proprietary technology. This causes vendor lock-in, which means the data stored can not be used in other tools without considerable switching costs. The user's data is sometimes even stored in data silos, meaning they do not have control over their data.

Lack of structure and expressivity. They are based on unstructured plain text and do not support structured knowledge, schema or semantics. This means integrating or collaborating with external knowledge is not possible. These characteristics have proven indispensable for Knowledge Management in Industry and Academics, however.

Query and search algorithms. A direct consequence of the previous points. Only structured information, like data stored in databases, supports query languages. These tools do not support query languages and rely on rolling their own search algorithms. This limits them to a single Information retrieval path: full-text search, with the user required to remember the exact keywords of what they are looking for.

Lack of open standards Even though these tools want to enable lifelong Personal Knowledge Management, they are not based on open standards. Interoperability with other data formats and services is not guaranteed. User's data might not be portable to other formats if the tool shuts down.

Open Standards for semantic Knowledge Graphs, called Semantic Web Technologies, are developed by the W3C. Theoretically, one could use Tools for Semantic Web Technologies to create PKGs and solve most of these limitations. Unfortunately, the tools based on Semantic Web Standards (SWSs) are notoriously hard to understand and use, even for computer scientist and programmers [5]. Expecting non-technical users to manage their personal knowledge with them is unrealistic.

1.3 Approach

To address the problems and limitations outlined in this chapter, we embrace the vision of a PKG Ecosystem. We establish that the creation of such an ecosystem is possible and beneficial, by outlining its structure and ~~desirable features~~ scope. We argue that the data model for PKGs should be a data graph based on Semantic Web Technologies. Such a data model is described in detail. Applications implementing the model will benefit from all the features developed for the Semantic Web. They will be able to express semantics, use external schema, and reasoning and have support for mature query languages.

Finally, we develop a prototypical implementation of the data model in the form of an interactive web application.

2 Preliminaries

This chapter provides definitions for concepts used throughout the thesis. These definitions were influenced by related literature [7, 25, 1]. For descriptions of terms and phrases please refer to the glossary.

2.1 Knowledge Graphs

Definition 1 (Graph). A **graph** is a tuple $G = (V, E)$, where V is a set of **vertices** and $E \subseteq \{\{x, y\} \mid x, y \in V \text{ and } x \neq y\}$ is a set of **edges**.

Definition 2 (Directed graph). A **directed graph** is a tuple $G = (V, E)$, where V is a set of vertices and $E \subseteq \{(x, y) \mid (x, y) \in V^2 \text{ and } x \neq y\}$ is a set of **directed edges**.

Definition 3 (Labeled graphs). An **edge-labeled graph** is a graph that has a labelling function $l_E : E \to X$, where X is a set of edge labels. A **vertex-labeled graph** is a graph that has a labelling function $l_V : V \to Y$, where Y is a set of vertex labels. A **labeled graph** is a graph that is both edge-labeled and vertex-labeled.

Definition 4 (Multigraph). A **multigraph** is a tuple $G = (V, E, \phi)$, where V is a set of vertices, E is a set of edges and $\phi : E \to \{\{x, y\} \mid x, y \in V \text{ and } x \neq y\}$ is an incidence function mapping every edge to an unordered pair of vertices.

Definition 5 (Knowledge Graph). All of the above definitions can be combined to get a directed labeled multigraph. A **Knowledge Graph** is such a directed labeled multigraph with a special typing label $t \in X$ that connects vertices from I with vertices from C, where $I \subset V$ corresponds to individuals or entities, and $C \subset V$ corresponds to classes or types.

The definition of KGs remains contentious [7, 13]. To put the formal definition above in words, consider a KG as a data graph that represents knowledge as entities and the relationships between them.

We define a **Personal Knowledge Graph** as a KG that represents knowledge relevant or related to the person in question.

A visual representation of a KG can be seen in Figure 2.1. The vertices and edges have labels, and the "is" label represents the special typing label.
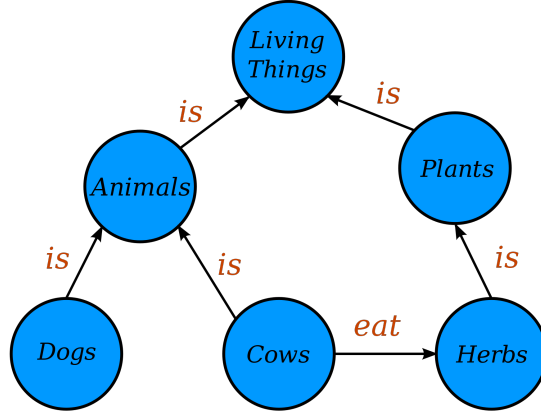

Figure 2.1: A Knowledge Graph

2.2 Semantic Web

The semantic web technologies that will be extensively used in this thesis include the Resource Description Framework (RDF) and the Terse RDF Triple Language (Turtle).

RDF is a data model standard designed by the W3C. It can be used to represent and exchange graph data on the web. RDF is a directed edge-labeled graph composed of triple statements. A triple consists of a subject, predicate and object. Three different RDF Terms can appear in a statement: Internationalized Resource Identifiers (IRIs), blank nodes and literals. IRIs refer to globally unique resources, blank nodes represent resources without identifiers, and literals are plain data values. The position of these terms inside of an RDF statement is limited as follows:

- The subject can contain an IRI or a blank node.

- The predicate can only contain an IRI.

- The object can contain an IRI, a blank node or a literal.

Multiple RDF Statements thus form a graph. This motivates the following

Definition 6 (RDF Graph). Let I be a set of IRIs, B a set of blank nodes, and L a set of RDF literals. A tuple $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an **RDF triple**, where s is the subject, p is the predicate and o is the object. A set of RDF triples is called an **RDF Graph**.

5

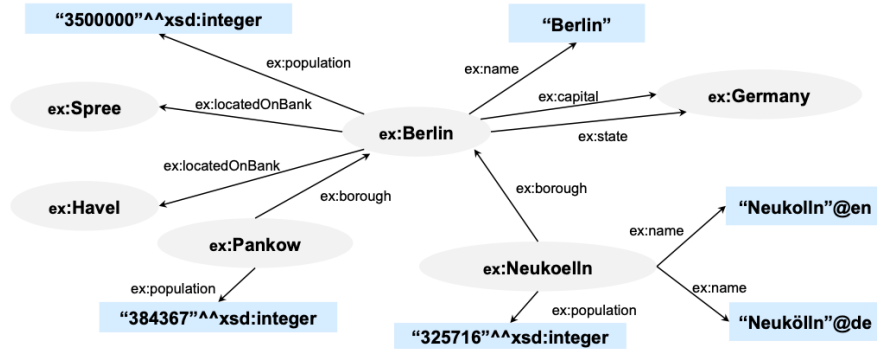A visual example of such an RDF Graph can be seen in Figure 2.2.

Figure 2.2: An RDF Graph

Turtle is a serialization and file format for RDF. It provides an easy-to-read syntax for RDF statements. Listing 2.1 shows an example of turtle's syntax.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://www.example.org/> .

ex:Pizza rdf:type ex:Meal .
ex:Pizza rdfs:label "Pizza" .
```

Listing 2.1: turtle example

For more Information about Semantic Web Technologies, we refer the reader to the official W3C specifications [17, 18, 24, 22, 21, 26].

3 Related Work

A search in academic search engines yields about a dozen results with "Personal Knowledge Graph" in the title; They are a novel research area. We observed different perceptions of PKGs in related work, depending on the perspective and context of the authors. These perceptions can roughly be grouped into the following categories:

PKGs as special cases of Knowledge Graphs. Here PKGs are conceived as KGs about an Individual, or personalized KGs. The focus lies on knowledge "about" the person and its relationships to other entities. This knowledge could for example be used by personal assistants to great effect. The PKG is mostly generated and maintained without intervention by the user.

PKGs as the basis for Personal Knowledge Management. The focus lies on knowledge "relevant" to the person. The Knowledge is in part generated and maintained by the user. Tools for interacting with PKGs are a logical consequence.

In the following sections, we will look at related work in each of these categories.

3.1 Knowledge Graphs

Hogan et al provide a comprehensive overview of Knowledge Graphs and related concepts and technologies [7]. They notably only distinguish two types of knowledge graphs in practice: open knowledge graphs and enterprise knowledge graphs. They explain that consensus-based schema, identity and context play a key role for knowledge graphs. If PKGs were implemented the same as the knowledge graphs described by Hogan et al, everything covered in their paper would also apply to them. In contrast to open and enterprise KGs, however, PKGs will probably not be maintained by large organizations or communities. This is an important difference to keep in mind because it will likely result in schemas and contexts that are not consensus-based and thus less expressive, harder to merge and not in line with linked open data principles [9].

Balog and Kenter established PKGs as a research area with their research agenda [2]. They define PKGs as "a source of structured knowledge about entities and the relation between them, where the entities and the relations between them are of personal, rather than general, importance. The graph has a particular "spiderweb" layout, where every node in the graph is connected to one central node: the user". They differentiate general

KGs from PKGs by stating that KGs concern public or global entities. This rules out non-public entities that Individuals might find relevant for their PKGs. Their definition is quite broad, but we would like for it to not enforce a "spiderweb" layout and also include entities that are just relevant and not related to the user. Broadening the definition in this way would connect the perceptions of PKGs about an Individual and PKGs for personal knowledge management.

Safavi et al describe a methodology to summarize KGs personalized to the user's interest [20]. This highlights other differences between PKGs and KGs: Data in a PKG should be relevant to its user, and it should focus on being human readable. Many KGs are primarily composed of knowledge relevant to machines.

3.2 Personal Knowledge Management

Bush wrote about his vision of the memex, an external device acting as a supplement to an individual's memory [3]. This vision was held back by technical limitations at the time, but highly influential. Davies et al were motivated by this vision and wrote about personal knowledge bases 60 years later, noting that the technical hurdles were gone and suggesting semantic web technologies, which were new at the time, as a promising approach. [4]. These papers were highly influential to this thesis. Even though the terms have changed from "memex" and "personal knowledge base" to "second brain", "personal knowledge graph" etc., their vision remains in large parts intact and unfulfilled as discussed in this thesis.

Velitchkov writes about Ontologies for his PKM tool Roam Research [23]. He emphasizes that ontologies for personal knowledge graphs are important and beneficial.

Matuschak writes about the limitations of Text and Books and analyses the difficulty of building intelligence augmenting tools for thought [11].

3.3 PKG Tools

As mentioned in chapter 1, several tools associate themselves with PKGs. The most relevant ones at the time of writing are Notion, Obsidian, Roam Research, LogSeq and RemNote [14, 19, 15, 10, 16]. These tools will be discussed in chapter 4.

4 PKG Ecosystem

Kickstarting the development of a PKG ecosystems is a prerequisite for realizing the vision of PKGs. The PKG ecosystem would consist of all the technologies, protocols and models that enable individuals to interact with their PKGs. Apart from enabling personal knowledge management, a PKG ecosystem could also address other problems in our technology landscape. Concerns about data silos, vendor lock-in and privacy are increasingly brought up in research and politics [12]. Considering the examples in chapter 1 we believe a PKG ecosystem is a promising approach to these problems.

We group the PKG ecosystem into three areas, like tech stacks in the industry:

- Data model

- Backend and Protocols

- Applications

Data and storage models will be discussed in chapter 5.

The backend will have the typical responsibilities of a backend server for modern applications. It will store an implementation of the data model, provide protocols for accessing the data, handle requests, manage access rights, etc. We will only describe functionalities expected from this backend, the implementation will not be a part of this thesis.

Once the data model and backend exist, they will provide a software platform that developers can target with their applications. For applications to work consistently across backends provided by different vendors, it would be beneficial for the data model and protocols to be open standards. We present a prototype of an application like this in chapter 7.

The PKG ecosystem is a large-scale undertaking similar in complexity to online cloud services. In this chapter, we analyze required functionalities this ecosystem needs to be capable of and compare it with the current software lanscape of personal knowledge management tools.

4.1 Functionalities

To gather the requirements for the PKG ecosystem, we assembled ideas present in research and industry [3, 4, 11, 8]. Some of these ideas are already established and implemented across products. Others are theoretical or might even pose to be major implementation hurdles. Nonetheless, all of them are possible to implement with current technological capabilities.

For each of the proposed categories, we list the essential ideas that we deem requirements for the successful development of a PKG ecosystem.

Data model

Centralization. All of the data in a PKG needs to be accessible in one place. This approach corresponds to a trend in operating systems search engines. Originally files, mails, calendars, bookmarks, and notes were coupled with their apps data silos and search. The modern approach is to have a single quick search that accesses data from all types of apps present on your PC, providing a centralized retrieval path.

Free association of knowledge elements. Our brain is able to freely associate any thought, without worrying about the type, category or hierarchy. We can associate the song we hear at a wedding, with the people, location, dance and food. Technology we use today, creates artificial partitions on our knowledge because it is stored in data silos. Interoperability is hindered by vendor lock-in. We are unable to connect our media, contacts, location and calendars. This limits our ability to associatively organize and express knowledge, and the number of paths available to retrieve it.

Knowledge types and semantics. A PKG needs to be able to store structured, semi- and unstructured Knowledge. Unstructured knowledge is data without metadata, e.g., text or images. Semi-structured knowledge includes metadata and structured knowledge is typed and based on schemata, e.g., database entities. Unstructured knowledge is quickly generated, like a "thoughtsream" captured as a note or voice memo. Structured knowledge takes time to generate, organize and enrich, because types and schmema need to be considered for every entity.

Transclusion. Also known as embeds or components, transclusion describes the ability for the same data to appear in several places. This is enabled by including data by reference, rather than duplicating it. For example, transclusion enables multiple documents that use the same paragraph to be updated as soon as the paragragh is changed in any location. It is a very powerful concept, that is barely used by non-technical people, where copy and pasting data is the norm.

Based on open standards. A PKG data model based on open standards provides portability and interoperability. We argue this is essential for knowledge work because it enables integrating and extracting data from diverse data sources.

Applications

These requirements apply to applications for personal knowledge management.

Text editing. PKG tools need to allow free text processing. As we will see in the next section, all current PKG Tools include either a plain text markdown editor, an outliner markdown editor, or a WYSIWYG text processor.

Views and filters. PKGs will potentially store massive amounts of knowledge, so apps working with them need to provide filters to limit the complexity and volume of data shown. Different visualizations like interactive graphs, tables, galleries, and boards aided by filters can be used to present and navigate relevant information in a digestible way.

Flexible refactoring. changes of any magnitude to the schema, entities, relationships, and views in a PKG need to be frictionless and quick. This allows smooth refactoring of unstructured Knowledge into structured Knowledge.

External knowledge. Apps should provide features for integrating external knowledge. Structured external knowledge should be searchable and queryable. For semi- and unstructured knowledge, source capture and transclusion are considerations. When importing external knowledge, prompts and filters should be available to filter relevant knowledge.

Bi-directional Links The internet enables unidirectional links between websites. With bi-directional links, any site would show all sites that link to it. They have not been included, due to vandalism concerns. For moderated networks like wikis or personal knowledge graphs, vandalism is not a concern and bi-directional links are established as useful features.

Backend

The requirements for the backend are similar to the backends in use today.

Multiplatform Sync PKGs should be available on any platform and sync changes between devices.

Cloud Drive for Files Most current PKM tools only allow for import and export of files. Editing, annotating or opening the files in other programs is not possible, which is a severe limitation. A PKG needs to provide or integrate with a cloud drive to access and edit files in real-time. This way files can also be freely associated with other data in the PKG.

Version History changes need to be backed up and reversible.

Protocols and APIs The backend should be compliant with protocols that enable access and querying of the storage model.

4.2 PKM Tool Software landscape

In this section, we provide a detailed comparison of select current personal knowledge management tools. We have looked at more than 50 tools and selected the most mature tools associated with PKGs [14, 15, 19, 10, 16]. We provide tables outlining the properties of these tools and analyze them in light of the requirements and limitations outlined previously.

Table 4.1: PKM tool overview – the software landscape

Figure 4.1 shows a general overview of the software landscape. We can see that all of these tools are based on proprietary data formats. Most of them store data in the cloud to enable features like multi-platform sync and collaboration.

Table 4.2: PKM tool workflows – what they can be used for

Figure 4.1 shows the workflows these tools enable. Most of the tools allow task management and journaling. features like password protected pages and flash cards are only supported by a select few. Notably Notion includes databases with views and filters.

⊞ Overview	⊞ Features	⊞ Editing	⊞ Formats	⊞ Relationships	⊞ PLACEHOLDER	⊞ Formatting	⊞ UI & UX	⊞ Sidebar	⊞ Export form
Aa App	☰ markdown	☰ tables	☰ code blocks	☰ foldable blocks	☰ footnotes	☰ checklists	☰ LaTeX	☰ Handwriting	
🟨 Notes									
🔴 Bear									
🟦 Workflowy									
Ⓝ Notion									
⚫ Obsidian									
🔶 Craft									
🌐 Roam									
[A] Athens									
🟢 Logseq									
❇ RemNote									

Table 4.3: PKM tool editing – text processing capabilities

Figure 4.3 shows the text editing capabilities of the different tools.

⊞ Overview	⊞ Features	⊞ Editing	⊞ Formats	⊞ Relationships	⊞ PLACEHOLDER	⊞ Formatting	⊞ UI & UX	⊞ Sidebar	⊞ Export formats	⊞ Collaboration	⊞ External A
Aa App	☰ link previews	☷ block	☰ page	☰ autocreate [[mentions]]	☰ linked [[mentions]]	☰ unlinked \`mentions\`	☰ outgoing mentions	⊙ link concept			
🟨 Notes											
🔴 Bear											
🟦 Workflowy											
Ⓝ Notion											
⚫ Obsidian											
🔶 Craft											
🌐 Roam											
[A] Athens											
🟢 Logseq											
❇ RemNote											

Table 4.4: PKM tool linking – relationships between knowledge elements

Figure 4.4 shows how the tools manage the granularity of their pages / blocks / entities and links between them. this concept is explained in more detail in chapter 5.

Aa App	≡ pdf	≡ text / md	≡ image	≡ video	≡ audio	≡ files
🟨 Notes						
🔴 Bear						
🟦 Workflowy						
N Notion						
◗ Obsidian						
◖ Craft						
🌐 Roam						
[A] Athens						
🟢 Logseq						
✳ RemNote						

Table 4.5: PKM tool formats – interoperability with file formats

Figure 4.5 shows the different file formats the tools are interoperable with.

4.3 Conclusion

When we compare the features of these PKG tools, we notice that they already realize some of our outlined requirements. Almost all of them are centralized and support text editing, refactoring, bi-directional links, associating entities and transclusion. Notion even supports databases with views and filters. Multi-platform sync and version history are also either implemented or on their roadmap. What is missing then, are cloud drives, support for structured knowledge and the ability to integrate and query external knowledge. This is a consequence of these tools relying on proprietary data and storage models, instead of developing or adopting open standards.

The data and storage model are fundamental for PKGs, because they limit what you can store and express in it. The data model needs to be flexible; able to express and associate everything from atomic entities (Terms, Concepts) to complex composite entities (Projects, Sciencific Work) [4]. The proposal of such a model will be the topic of the next chapter.

5 RDF as a Data Model for PKGs

Information is predominantly stored as a combination of language, symbols and images, presented in a linear arrangement. Examples include books, videos and digital documents. This linear arrangement provides a sense of chronology and orientation for information retrieval. An important realisation with this kind of presentation, is that it can be interpreted as an ordered tree. A book can be seen as the root node, with edges to the chapters, headings, pages and finally the content as leaf nodes. Similar a TV Show is separated into Seasons, Episodes, Scenes and Cuts.

In Computer and Information Science more powerful ways to store knowledge have been developed. Structured and semi-structured Information is stored with XML, JSON, Databases etc. to great effect. There is no longer a sense of chronology or orientation, but these technologies enable powerful features like querying, search, (insert examples. . .) which can also be used for information retrieval. These advanced Data Models can be also be interpreted as a Graph.

A Graph is thus a very flexible, powerful and interoperable way to store Information and Knowledge (conjecture?). If designed correctly, a graph could leverage the strenghts of text documents and database technologies, while acting as the base for the PKG Ecosystem.

5.1 Granularity of Entities

An important consideration is the granularity of the Knowledge represented in a Knowledge Graph.

An Entity is... It can be represented as a file, node, block, row, paragraph, Word, Page, etc.

This granularity of Entities is what defines what can be said, what can be linked, and what can be queried.

We will sometimes also refer to these entities as knowledge elements, elements or nodes.

This is the most fundamental consideration. We use the definition

5.2 Data and Storage Models

Let us first compare the Storage Models

Documents and Files. While providing excellent orientation and navigation, this approach is too limited. Free association of Knowledge Elements is not possible, as every piece of content belongs to a single Document, and every File belongs to a single folder.

relational Databases. Querying, Freely associationg Knowledge Elements and even Transclusion is possible, but Database Schemata and Management are too restrictive for creating semi- or unstructured Knowledge. The Visualization of Databases also tends to be too complex for non-technical people.

Graph Databases. Graph databases provide the benefits of Relational Databases, while providing intuitive visualizations, flexible schemata and more natural query evaluations for following associated Knowledge.

Many Graph Frameworks could serve as the foundation for the PKG Ecosystem, but the Semantic Web Standards developed by the W3C are especially fitting. As the basis for open Knowledge Graphs and Vocabularies, they have a rich background in Academia and Industry. They provide mature features through Standards like RDF, OWL, SHACL and SPARQL. Because they are open standards, they are the best bet for PKGs as the basis for lifelong Personal Knowledge Management.

RDF Knowledge Graphs are **directed labeled graphs.** This Model is also fitting for Personal Knowledge Graphs. Because PKGs are for humans, there is however the additional requirement of making the Data Model human readable. At the very least, all Elements of a PKG need a label.

We will go with a **directed labeled RDF Graph** as the basis for our PKG model. Structured Knowledge will be implemented as closely as possible to Semantic Web Standards. Semi- and unstructured Knowledge will be modeled with Notes using semantic markdown, proposed in chapter [semantic markdown].

5.3 RDF Dataset Layers

The PKG Model consists of a Graph Dataset with several Layers:

Static external Schema (Ontologies) Basic Ontologies like RDF, RDFS and OWL will be included by default. Advanced users can add Ontologies and enforce their usage, to make their PKGs mergable with external KGs.

Adaptive personal Schema The User can freely edit Classes and Properties in his personal Schema, which will be saved for future use.

Structured Data and Notes This is the main Data Layer of the Personal Knowledge Graph.

Temporary Data A temporary Data Layer for external RDF Data and runtime generated data gained from Notes, reasoning, inverted links, etc.

Metadata this data Layer can be used by applications ****to save access rights, order of Notes, or display properties like highlights, alignment, etc.

5.4 CRUD operations and effects

- Quads are RDF Triples with an optional Graph IRI (subject, predicate, object, graph?) spog - Pseudo Code for clarification, a Javascript Implementation can be found in the appendix. - Dataset is a triple store / RDF Database - function calls are blue and their parameters indented - Quads are in Turtle syntax and separated by newlines - Each node has an existential triple 'node rdf:type rdfs:resource' - ¡IRI¿ represents dereferencable IRI parameters

Create

```
// create Class "Idea" (into users personal schema layer)
// inserts the quads in this array into the dataset
Dataset.addQuads
  :IRI rdf:type rdfs:Resource :UserSchema
  :IRI rdf:type rdfs:Class :UserSchema
  :IRI rdfs:label "Idea" :UserSchema

// create Relationship "author" (into users personal schema layer)
Dataset.addQuads
  :IRI rdf:type rdfs:Resource :UserSchema
  :IRI rdf:type rdfs:Property :UserSchema
  :IRI rdfs:domain rdfs:Resource :UserSchema
  :IRI rdf:range rdfs:Resource :UserSchema
  :IRI rdfs:label "author" :UserSchema

// create Node "Hypothesis" (into data layer)
Dataset.addQuads
    :IRI rdf:type rdfs:Resource :Data
    :IRI rdfs:label "Hypothesis" :Data
```

Read

```
// get Quad with matching subject, predicate, object, graph
// returns: Array of matching Quads
Dataset.match
  subject
  predicate
  object
  graph

// get Quads with matching subject and wildcards for  predicate, object, graph
Dataset.match
  subject      //e.g. :IRI or Literal "Hypothesis"
  *
  *
  *
```

Delete

```
// delete Class or Node <:IRI>
// deletes all quads that include this node
Dataset.deleteQuads
  Dataset.match
    <:IRI>
    *
    *
    *
Dataset.deleteQuads
  Dataset.match
    *
    <:IRI>
    *
    *
Dataset.deleteQuads
  Dataset.match
    *
    *
    <:IRI>
    *

// delete Relationship <:IRI>
// deletes all quads that include this relationship
Dataset.deleteQuads
  Dataset.match
    *
    <:IRI>
    *
    *

// delete Quad s p o g
// deletes this quad
Dataset.delete
  s
```

```
    p
    o
    g
```

Update

can be implemented either as a combination of delete and create operations or as more performant in-place operations. The in-place operations need to be optimized for performance, based on the exact tech stack.

```
// update Label to "Hypothesis"
Dataset.deleteQuads
  Dataset.match
    <:IRI>
    rdfs:label
    *
    *

Dataset.addQuads
  :IRI rdf:type rdfs:Resource :Data
  :IRI rdfs:label "Hypothesis" :Data
```

5.5 Markdown Outliner in RDF

In this section I propose an implementation for storing semistructured Text in RDF.

Note that because RDF Graphs are unordered, Text nodes need to be coupled with metadata defining display order, etc.

Let us start with a simple example:

```
:IRI rdf:type :Person, :Artist, :Musician
:IRI rdfs:label "Lady Gaga"
:IRI foaf:givenName "Stefani Joanne Angelina"
:IRI foaf:familyName "Germanotta"

\# attaches the note to <:IRI> Lady Gaga entity
:IRI :note :note\_000
:note\_000 rdf:type :SemanticMarkdown
:note\_000 rdfs:label "Here is some Trivia about Lady Gaga:"
:note\_000 :note :note\_001
:note\_001 rdf:type :SemanticMarkdown
:note\_001 rdfs:label "Lady Gaga is not her real name."
:note\_000 :note :note\_002
:note\_002 rdf:type :SemanticMarkdown
:note\_002 rdfs:label "She is sometimes confused with [Gwen Stefani]"
:note\_000 :note :note\_003
:note\_003 rdf:type :SemanticMarkdown
```

```
:note\_003 rdfs:label "She got a lot of attention for extravagant dresses,
    like her [Meat Dress]"
```

Could be rendered like this by a PKG Tool:

```
- rdfs:label -> "Lady Gaga"
- rdfs:type -> [Person], [Artist], [Musician]
- foaf:givenName -> "Stefani Joanne Angelina"
- foaf:familyName -> "Germanotta"

- Here is some Trivia about Lady Gaga:
    - Lady Gaga is not her real name.
    - She is sometimes confused with [Gwen Stefani]
    - She got a lot of attention for extravagant dresses, like her [Meat Dress]
```

The RDF Property ':note' is special and signals to the PKG Tool that this Entity should be treated as a Text Note / Semantic Markdown on the Entity. Note that all of the unstructured text notes are still their own IRIs. This means SemanticMarkdown can be associated with or rendered on an unlimited number of other Entities. They have 'rdf:type :SemanticMarkdown' , which signals the PKG Tool to render it in outliner mode.

Tweets & Antworten   **Medien**   Gefällt mir

roa...
Draft documentation at
roamresearch.com/#/app/nodebook...

And their story in this thread 🧵
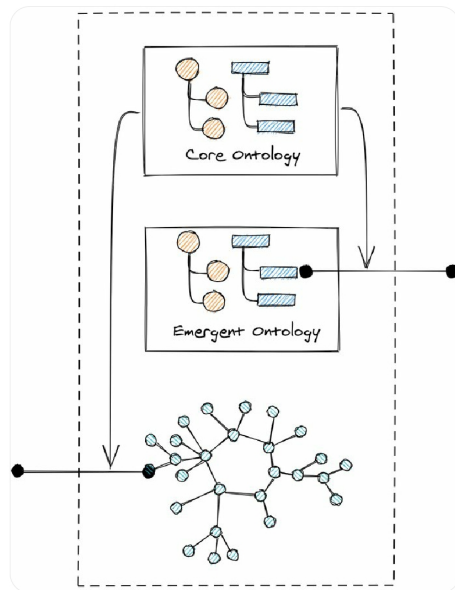
💬 1   ⟲ 14   ❤ 55   ⤴

Diesen Thread anzeigen

Ivo Velitchkov @kvistgaard · 05. Apr. ⋮

6 Semantic Markdown

Markdown is a lightweight markup language for creating formatted text using a plain-text editor. Text gets transformed by prepending or enclosing it in special Characters like '# * -' etc.

6.1 Markdown Flavors

Markdown although not standardised, is embraced by the Web Community and continuously extended. There are several Flavors (Github, CommonMark, etc.), which can be roughly separated into the following expressivity levels:

Basic Markdown Includes mostly Text formatting:

- # Headings, ****Bold****, **Italic**,   Strikethrough   , 'Code'

- Numbered and bulleted Lists

- Images and Links as URL references

Extended Markdown Includes advanced Formatting options:

- Tables

- Heading ID's (for in-document navigation)

- Syntax highlighted Code Blocks

- Footnotes

- Todos

- Emoji

- Highlighting

- Sub- and Superscript

- Table of Content

- Callouts

- Comments

- Captions

"Hypertext Markdown" Recently Note-Taking Tools have adopted an extension to Markdown Syntax, enabling toolwide Hyperlinks [Bear, Notion, Roam, LogSeq, Obsidian]. Text enclosed in certain special characters is automatically indexed as special connections between markdown files. These are rendered as links, mentions or embeds.

- `[[` gets converted to links to documents

- `((` links to a paragraph

- `{{` embeds documents or paragraphs

- `$$` LaTeX code

- `^^` Highlight Text

A similar appraoch to "Hypertext Markdown" could be used as a syntax for creating RDF triples in Markdown, which I will propose in the next section.

6.2 A Semantic Markdown extension (.mds?)

This takes the approach of Hyptertext markdown and adds semantic relationships to it. These semantic relationships are then manifested into the temporary Data Layer during runtime.

The Syntax is invoked by wrapping the expressions in special characters. These Characters are typed twice, so they don't get accidentaly triggered in the application. For the characters used, I looked at special characters easily reachable on both a physial and digital ANSI Keyboard. I substracted the ones that are frequently used in other contexts, like `**` `~~` `\\` `!!` `??`, to come up with the following list:

- `[[` link to IRI, similar to rdfs:seeAlso

- `((` embed title of a Note, similar to mention (without children)

- `{{` embed a Note or IRI (with children)

- `>>` link to external IRI

- `<<` embed of external IRI

- `&&`

- `%%`

- `::`

- `;;`

The syntax to create RDF Properties with these characters could use a label for the link to render in markdown, followed by turtle triples separated by a delimiter, a comma in this example. These Turtle Triples would automatically use the IRI to which the '¡:note¿' is attached to as their subject.

```
[[
"This is the link label",
rdf:type rdfs:Resource,
rdfs:seeAlso <http://www.w3.org/2000/01/rdf-schema\#>
]]
```

Let us consider an Example of how this could look like in a Note:

```
# Notes on the RDF IRI "Sun"
The Sun is the [["Star", rdf:type ex:Star]] at the center of
the [["Solar System", ex:location ex:Solar\_System]].
It is a nearly perfect ball of hot plasma.


# Notes on the RDF IRI "Earth"
Earth is the third [["planet", rdf:type ex:Planet]]
from the [["Sun", rdfs:seeAlso ex:Sun]] and
the only astronomical object known to harbor life.
While large volumes of water can be found
throughout the [["Solar System", ex:location ex:Solar\_System]],
only Earth sustains liquid surface water.
```

Would result in the following RDF inserted into the temporary data Layer:

```
ex:Sun rdf:type ex:Star
ex:Sun ex:location ex:Solar\_System

ex:Earth rdf:type ex:Planet
ex:Earth rdfs:seeAlso ex:Sun
ex:Sun ex:location ex:Solar\_System
```

Embeds and Title Mentions can only be used as separate Paragraphs, and are rendered as the content or title of their reference.

Conclusion

I demonstrated how Semantic Markdown can be used to infer RDF Statements from plain Text through the use of a special Syntax. These RDF statements always connect to

all the parents of the '¡:note¿' (recursively), by using the parents subjects for the inferred triples. This enables the triples to be present on all parent IRIs

- great candidate for standardisation - which special characters to use - syntax inside the characters and delimiters - syntax for OWL inverseProperties etc. ?

7 A Prototype PKG Tool

While many challenges of PKGs can be tackled by choosing a good model and architecture, the User Experience (UX) aspect needs to be considered seperately. Semantic Web Technology is hard to understand even for Computer Scientists [???]. Expecting non-technical people to grasp all the Concepts SWS rely on is unrealistic. For personal use by non-technical people these concepts need to be abstracted away with self-explanatory User Interface (UI) aided by Tutorials.

In this Chapter I present a Prototype that aims to be user friendly and to not rely on previous knowledge of Semantic Web Technologies [ARC repo].

7.1 Tech-Stack

The Prototype is implemented as a Web Application using HTML, CSS, TypeScript and the Frameworks TailwindCSS and Svelte.

The RDF Data is stored in memory by using my own implementation of the RDFJS specification called RDFun [appendix, RDFJS]. RDFJS is a Specification for working with RDF in JavaScript. A host of Libraries exist that provide functionality for RDFJS compliant Graph Datasets. I used libraries for fetching external RDF Data, querying and manipulating Datasets [rdfetch, comunica, N3].

7.2 Usability Considerations

The prototype is implemented as a single page web app. This enables a frictionless no-install usage.

The Design is similar to what people are used to from Note-Taking Apps like Notion, Obsidian or LogSeq and Browsers.

No knowledge of Semantic Web Technologies like RDF or OWL is necessary to use the App, even though it is based on RDF Data.
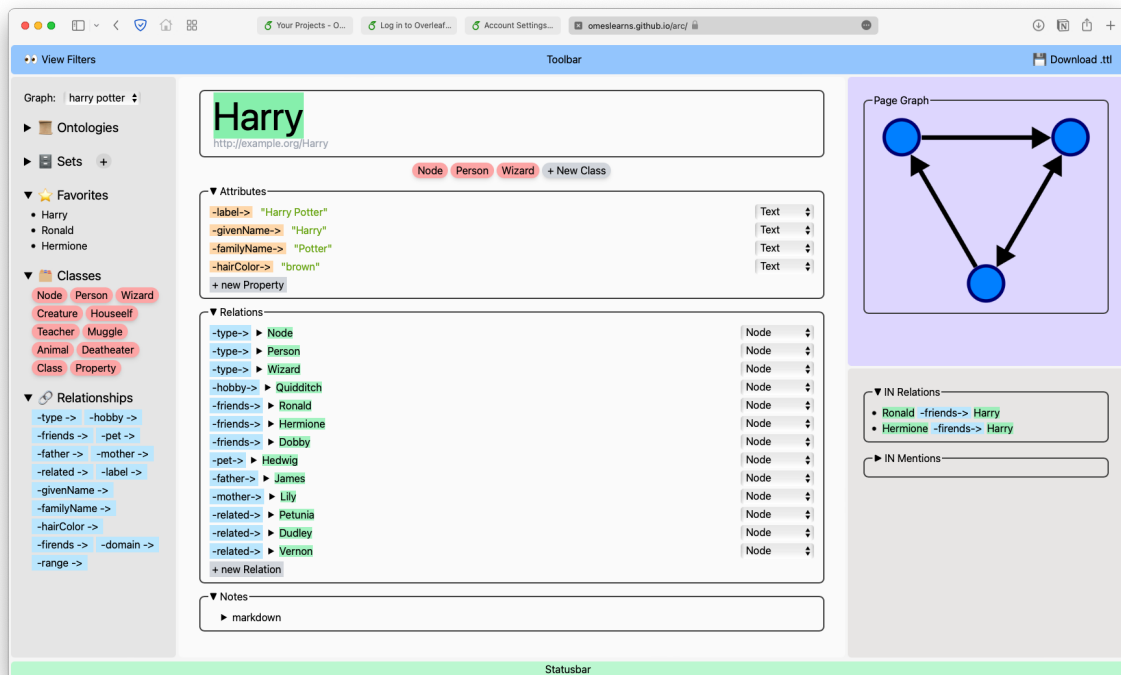
7.3 The User Interface

Figure 7.1 shows a screenshot of the UI. It is divided into several Sections, some of them similar to sections found in Apps people frequently use, like note-taking apps, Wikipedia or email clients. From top left to bottom right:

Toolbar The Toolbar is similar to the ones found in modern browsers. It displays open Nodes in Tabs with the IRI appearing as the Title for the Tab. In the outermost corners are shortcuts for accessing prominent functionality.

Sidebar The Sidebar serves as navigation and orientation. Users can navigate to Classes, Relationships and their favorite Nodes in the PKG. There are also overviews of used Schema and Sets / Views.

Editor The Editor renders all the relevant Information about the currently active Node. The User can toggle which Information is relevant to him and navigate to related Nodes. He can also create, update and delete related Entities and Text.

Graph panel The Graph Panel serves as an overview of the context of the current and associated Nodes in visualized form. This provides further orientation and a spatial sense of connectedness in the PKG.

Relation panel The Relation Panel shows inferred relations and mentions. (Meaning arcs from other nodes to the current node.

The label and classes of the current node are placed in special positions and hidden from the Relations. The label is prominently displayed as the title, reminiscent of most Text Processors. Classes are presented right below it, similar to tags or categorys in other Applications. This is done to build on the familiarity with similar UIs the User might have interacted with.

Notice how Classes, Properties, Entities and Literals are all visually distinguished in the UI, highlighting their different nature and providing easy navigation of the content.

The Semantic Markdown is displayed in a outliner UI, as though it was a hierarchical attachment to the current Node. The structured semantics in it are displayed on the relevant Nodes in an opaque fashion, to hightlight the fact that they are not manifested in the data layer, but rather inferred from Notes. A button enables the user to transform them into explicit statements from the note layer into the data layer. Similar functionality is present for integrating external structured Data (like wikidata, dbpedia, etc.)

7.4 Functionality

The following is an overview of Features and Functionalities of the Prototype.

All of this functionality manipulates the Dataset as outlined in Chapter [Data Model], by creating, updating or deleting RDF triples.

Create Node Pressing the button labeled "+ New Node" in the botton left corner or using the keyboard shortcut "ctrl + n" creates a new Node in the Graph. The User is automatically navigated to the new Node and can edit the Title / Label.

Attach Class A new Class can be attached to the current Node by clicking the "+ add Class" button right below the Title / Label.

Attach Attribute / Relationship A RDF Triple with the Current Node in subject position can be created by pressing the "+ add Attribute / Relation" Button at the bottom of the respective Sections. This prompts the user to enter a label for the predicate (Relationship) and Object (Target Entity / Literal) of the triple.

Write Semantic Markdown In the Notes Section, Semantic Markdown can be edited in an Outliner Text Processor Interface (This feature is not fully implemented yet).

Switch RDF Graph Dataset The user can select the RDF Dataset in the "Graph" labeled select input in the Sidebar, to switch between Graph Datasets.

Import RDF Graphs from URL The user can import RDF Data from external Sources, by selecting "+ import from URL" from the "Graph" labeled select input in the Sidebar.

8 Discussion

use cases:

- government - public institutions like university and school - teachers instead of giving out information about "what" is important, - also give out structured information that shows associations - this structured knowledge could even be collaboratively maintained across severel schools / universitys

apart from the obvious use cases for personal knowledge graphs, is see the following use cases:

government structured information and databases government semi-structured information and knowledge, public education

schools and universities teaching materials ...

differentiate wikis / PKGs pkgs don't HAVE to be consensus based, but they can.

just like in normal life people with similar ideas will group and build consensus built kgs, like teachings,

9 Conclusions

- Problem with PKGs: Schema, Context, Identity - The Missing Global Ontology standard for Personal Knowledge - How tightly should PKGs integrate concensus based open ontologies? - people have been struggling to develop better mediums than text, for mainly x reasons: - people learn to read and write in school, they don't learn information science or databases

- Semantic Web also promises global consistency of terminology and unambiguous identifiers for shared concepts. This latter capability is what makes Semantic Web technologies an enticing component to consider for a PKB system. [memex60]

How will the tool be used by real users?

Why Graphs for personal Knowledge Bases?

comparison of Text Documents / PDFs / Books to the Graph structure (hierarchy, chronology, sense of orientation / navigation)

- Use Cases of KG

- Open

- Enterprise

- Personal

- Collaborative PKGs?

For the Prototype, I would have liked to be able to do user tests and integrate with a backend like described in [pkg ecosystem]

This thesis has explored the PKG Ecosystem that would develop from PKGs serving as the foundation for lifelong Personal Knowledge Management.

We have shown that Information and Knowledge from hierarchical Documents and Databases can be mapped to a directed labeled Graph Model. Implemented with Semantic Web Standards, we can levererage Semantic Web Technologies for PKGs and PKM. The Model provides great flexibility, expressivity and interoperability.

Semantic Markdown was proposed as a Syntax extension for Markdown, to mix RDF Relationships in plain text.

PKGs are still a mostly uncharted research area with basic research questions regarding definition, generation and enrichment.

For PKGs to serve as the foundation for PKM, we need standards for Data formats for Media, Health, Contacts, etc. and for Semantic Markdown or similar Hypertext.

We also need to consider the Backends that manage security, query evaluation and access rights.

The SOLID Project pioneered by Tim Berners Lee already supports RDF and SPARQL and might be a promising approach for PKGs.

with more time, I would have liked to

develop the prototype more in depth

do usability tests

have a more detailed look at some of the requirements, like version control and access rights for RDF graphs

A RDFun Javascript Library

RDFun is a JavaScript Library we wrote that implements part of the RDFJS spec
[RDFJS]. A.1

```
import type { NamedNode , Quad , Term } from "./Datafactory";

export class Store {
  size: number;
  #memory: Quad [];
  constructor() {
    this.#memory = [];
  }

  add(quad: Quad) {
    if (!this.has(quad)) this.#memory.push(quad);
    return this;
  }
  delete(quad: Quad) {
    const quadIndex = this.#memory.findIndex((q) => q.equals(quad));
    this.#memory.splice(quadIndex , 1);
    return this;
  }
  has(quad: Quad) {
    return this.#memory.some((q) => q.equals(quad));
  }
  match(
    s: Term | null ,
    p?: Term | null ,
    o?: Term | null ,
    g?: NamedNode | null
  ) {
    const sMatches = s
      ? this.#memory.filter((quad) => quad.subject.equals(s))
      : this.#memory;
    const pMatches = p
      ? sMatches.filter((quad) => quad.predicate.equals(p))
      : sMatches;
    const oMatches = o
      ? pMatches.filter((quad) => quad.object.equals(o))
      : pMatches;
    const gMatches = g
      ? oMatches.filter((quad) => quad.graph.equals(g))
      : oMatches;
```

```
    const matchesStore = new Store();
    gMatches.forEach((quad) => matchesStore.add(quad));
    return matchesStore;
  }

  [Symbol.iterator]() {
    let index = 0;
    return {
      next: () => {
        if (index < this.#memory.length) {
          return {
            value: this.#memory[index++],
            done: false,
          };
        } else {
          return { done: true };
        }
      },
    };
  }

  // non standard
  addQuads(quadArray) {
    quadArray.forEach((quad) => this.add(quad));
    return this;
  }
}
```

Listing A.1: Dataset.ts

```
export const namedNode = (value) => new NamedNode(value);
export const blankNode = (value?) => new BlankNode(value);
export const literal = (value, langOrDataType?) =>
  new Literal(value, langOrDataType);
export const variable = (value) => new Variable(value);
export const defaultGraph = () => new DefaultGraph();
export const quad = (s, p, o, g?) => new Quad(s, p, o, g);

export class NamedNode {
  termType: string;
  value: string;
  constructor(value: string) {
    this.termType = "NamedNode";
    this.value = value;
  }

  equals(other: Term) {
    if (
      typeof this === typeof other &&
      this.termType === other.termType &&
      this.value === other.value
    ) {
      return true;
    }
    return false;
  }
}
export class BlankNode {
  termType: string;
  value: string;
  constructor(value?: string) {
    this.termType = "BlankNode";
    this.value = value;
  }

  equals(other: Term) {
    if (
      typeof this === typeof other &&
      this.termType === other.termType &&
      this.value === other.value
    ) {
      return true;
    }
    return false;
  }
}
export class Literal {
  termType: string;
  value: string;
  language: string;
  datatype: NamedNode;
  constructor(value: string, langOrDataType?: string | NamedNode) {
```

```
      this.termType = "Literal";
      this.value = value;
      if (typeof langOrDataType == "string") this.language = langOrDataType;
      else this.datatype = langOrDataType;
    }
    // TODO: fix copy pasta
    equals(other: Term) {
      if (
        typeof this === typeof other &&
        this.termType === other.termType &&
        this.value === other.value
      ) {
        return true;
      }
      return false;
    }
  }
  export class Variable {
    termType: string;
    value: string;
    constructor(value: string) {
      this.termType = "Variable";
      this.value = value;
    }

    equals(other: Term) {
      if (
        typeof this === typeof other &&
        this.termType === other.termType &&
        this.value === other.value
      ) {
        return true;
      }
      return false;
    }
  }
  export class DefaultGraph {
    termType: string;
    value: string;
    constructor() {
      this.termType = "DefaultGraph";
      this.value = "";
    }

    equals(other: Term) {
      if (
        typeof this === typeof other &&
        this.termType === other.termType &&
        this.value === other.value
      ) {
        return true;
      }
```

```
      return false;
    }
}
export class Term {
  termType: string;
  value: string;
  constructor() {}

  equals(other: Term): boolean {
    throw new Error("equals() not implemented");
  }
}

export class Quad {
  subject: Term;
  predicate: Term;
  object: Term;
  graph: Term;
  constructor(s, p, o, g?) {
    this.subject = s;
    this.predicate = p;
    this.object = o;
    this.graph = g;
  }
  equals(other: Quad) {
    if (
      typeof this === typeof other &&
      this.subject.equals(other.subject) &&
      this.predicate.equals(other.predicate) &&
      this.object.equals(other.object) &&
      this.graph.equals(other.graph)
    ) {
      return true;
    }
    return false;
  }
}
```

Listing A.2: DataFactory.ts

Glossary

vendor lock-in also known as proprietary lock-in or customer lock-in, makes a customer dependent on a vendor for products, unable to use another vendor without substantial switching costs.. 2

References

[1] Maribel Acosta. *Knowledge graph lecture*. 2021.

[2] Krisztian Balog and Tom Kenter. "Personal Knowledge Graphs: A Research Agenda". In: *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval* (2019).

[3] Vannevar Bush et al. "As we may think". In: *The atlantic monthly* 176.1 (1945), pp. 101–108.

[4] Stephen Davies, Javier Velez-Morales, and Roger King. "Building the Memex Sixty Years Later: Trends and Directions in Personal Knowledge Bases ; CU-CS-997-05". In: 2005.

[5] *Easier RDF*. URL: https://github.com/w3c/EasierRDF (visited on 07/15/2022).

[6] Douglas C. Engelbart. "AUGMENTING HUMAN INTELLECT: A CONCEPTUAL FRAMEWORK". In: 1962.

[7] Aidan Hogan et al. "Knowledge graphs". In: *Communications of the ACM* 64 (2021), pp. 96–104.

[8] William P Jones and Jaime Teevan. *Personal information management*. Vol. 14. University of Washington Press Seattle, 2007.

[9] *Linked open data principles*. URL: https://www.example.org (visited on 07/15/2022).

[10] *Logseq*. URL: https://logseq.com (visited on 07/15/2022).

[11] Andy Matuschak and Michael A. Nielsen. "How can we develop transformative tools for thought". In: 2019.

[12] Me. "YOU HAVE MISSING CITATIONS, FIX THEM!" In: ().

[13] MichaelK.Bergman. *A Common Sense View of Knowledge Graphs*. 2019. URL: http://www.mkbergman.com/2244/a-common-sense-view-of-knowledge-graphs/ (visited on 07/15/2022).

[14] *Notion*. URL: https://www.notion.so (visited on 07/15/2022).

[15] *Obsidian*. URL: https://obsidian.md (visited on 07/15/2022).

[16] *RemNote*. URL: https://www.remnote.com (visited on 07/15/2022).

[17] *Resource Description Framework*. URL: https://www.example.org (visited on 07/15/2022).

[18] *Resource Description Framework Schema*. URL: https://www.example.org (visited on 07/15/2022).

[19] *Roam Research*. URL: https://roamresearch.com (visited on 07/15/2022).

[20] Tara Safavi et al. "Personalized Knowledge Graph Summarization: From the Cloud to Your Pocket". In: *2019 IEEE International Conference on Data Mining (ICDM)* (2019), pp. 528–537.

[21] *SPARQL Protocol and RDF Query Language*. URL: https://www.example.org (visited on 07/15/2022).

[22] *Terse RDF Triple Language*. URL: https://www.example.org (visited on 07/15/2022).

[23] Ivo Velitchkov. *Ontologies for Roam Research*. 2022. URL: https://github.com/kvistgaard/roamo (visited on 07/15/2022).

[24] *Web Ontology Language*. URL: https://www.example.org (visited on 07/15/2022).

[25] *Wikipedia: Graph Theory*. URL: https://en.wikipedia.org/wiki/Graph_theory (visited on 07/15/2022).

[26] *XML Schema Definition*. URL: https://www.example.org (visited on 07/15/2022).