Standards Based Personal Knowledge Graphs

Omes Baltes

Bachelor's Thesis – 15. July, 2022
Ruhr University Bochum
Faculty of Mathematics

Supervisor: Prof. Dr. Maribel Acosta
Advisor: Prof. Dr. Ajsa Fischer

Abstract

Personal knowledge graphs as the foundation for personal knowledge management have gained increasing attention from the industry. The tools associated with these terms are based on proprietary technologies, resulting in vendor lock-in and limited functionality. This thesis explores how personal knowledge graphs can leverage open semantic web standards to increase their expressiveness, interoperability and capabilities.

My gratitude goes to

Prof. Acosta for the space to grow
Sebastian for the paths to wander
Hendrik & Lara for the company on the way

Contents

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], [11]. 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 [2], [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 users' 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.

Missing query language support. 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. Users' 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 scientists 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 scope. We argue that the data model for PKGs should be an RDF data graph based on semantic web technologies. Such a data model is described in detail. This enables applications implementing the model to benefit from all the features developed for the semantic web. They will be able to express semantics, use external schema, 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].

2.1 Knowledge Graphs

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

Vertices are also called nodes or points. Edges are sometimes called links or relationships.

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

Directed edges are also called arcs.

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 $\phi : E \to \{\{x, y\} \mid x, y \in V\}$ 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], [11]. 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 "type" label points from individuals (white nodes) to classes (blue nodes).

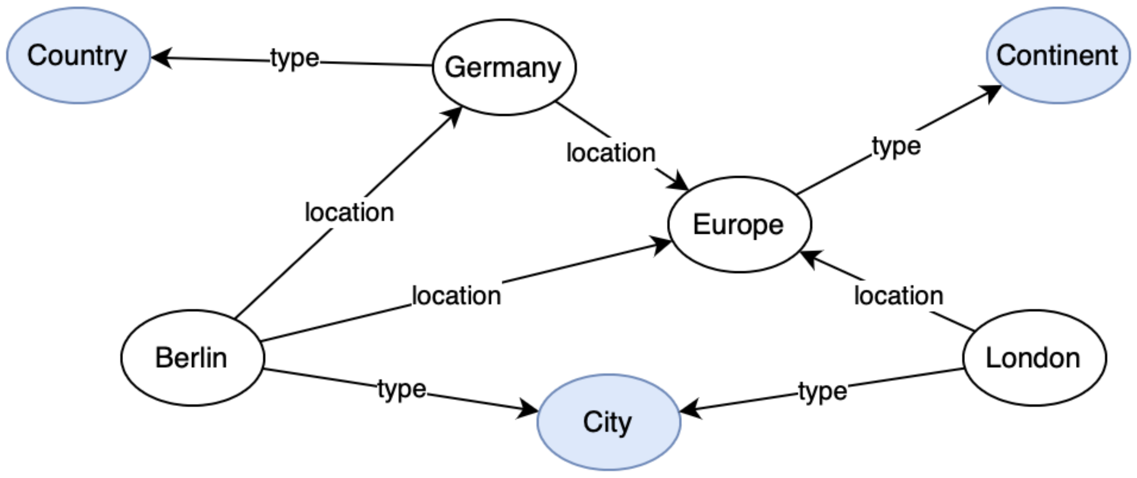


Figure 2.1: A Knowledge Graph

2.2 Semantic Web

The semantic web technologies that will be used extensively 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 $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup Ł)$ 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**.

A visual representation of an RDF graph can be seen in Figure 2.2. IRIs are simplified and abbreviated. Green rectangles represent literals. Notice how node and arc lables are not necessarily human readable, as they are IRIs.

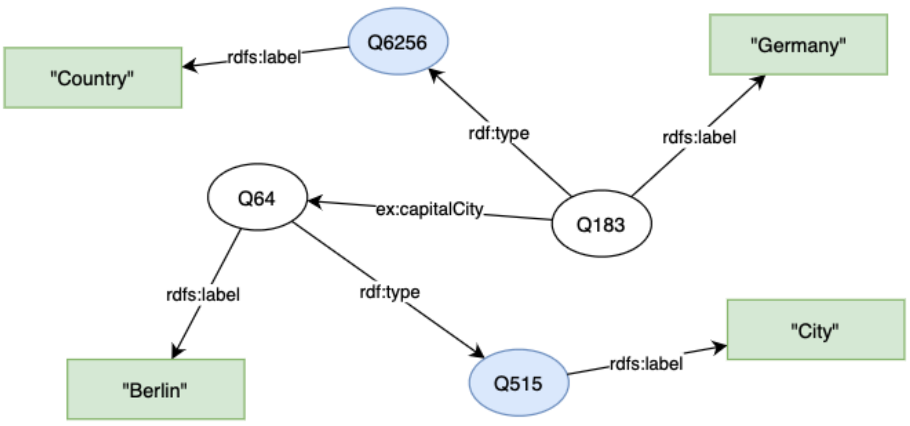


Figure 2.2: An RDF Graph

Turtle is a serialization and file format for RDF. It provides an easy-to-read syntax for RDF triples. It allows 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 [16], [17], [21], [22], [24], [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 digital 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.

Balog and Kenter established PKGs as a research area with their research agenda [1]. 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 users' interest [19]. 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 [2]. 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 [10].

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 [9], [12], [13], [15], [18]. 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. 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 [2], [4], [8], [10]. 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 system 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 schema 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.

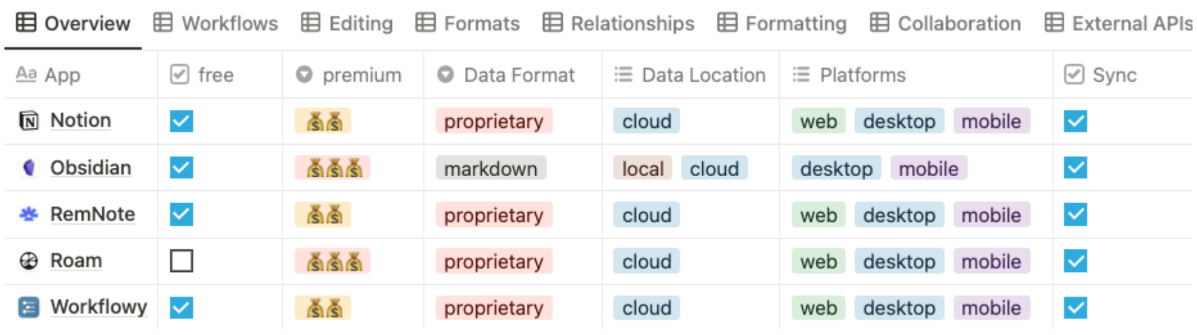
Cloud Drive for Files. Most current PKM tools only allow for import and export of files. Editing, annotating or opening the files in other programs is not possible, which is a severe limitation. A PKG needs to provide or integrate with a cloud drive to access and edit files in real-time. This way files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

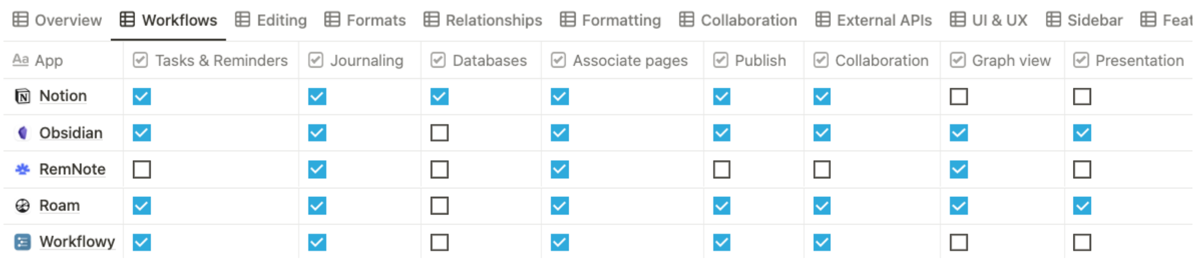
In this section, we provide a detailed comparison of select current personal knowledge management tools. We have looked at more than 50 tools and selected the most mature tools associated with PKGs [9], [12], [13], [15], [18]. We provide tables outlining the properties of these tools and analyze them in light of the requirements and limitations outlined previously.


Overview
Workflows
Editing
Formats
Relationships
Formatting
Collaboration
External APIs

App	free	premium	Data Format	Data Location	Platforms	Sync
Notion	✓	💰💰	proprietary	cloud	web desktop mobile	✓
Obsidian	✓	💰💰💰	markdown	local cloud	desktop mobile	✓
RemNote	✓	💰💰	proprietary	cloud	web desktop mobile	✓
Roam	☐	💰💰💰	proprietary	cloud	web desktop mobile	✓
Workflowy	✓	💰💰	proprietary	cloud	web desktop mobile	✓

Table 4.1: PKM tool overview – the software landscape

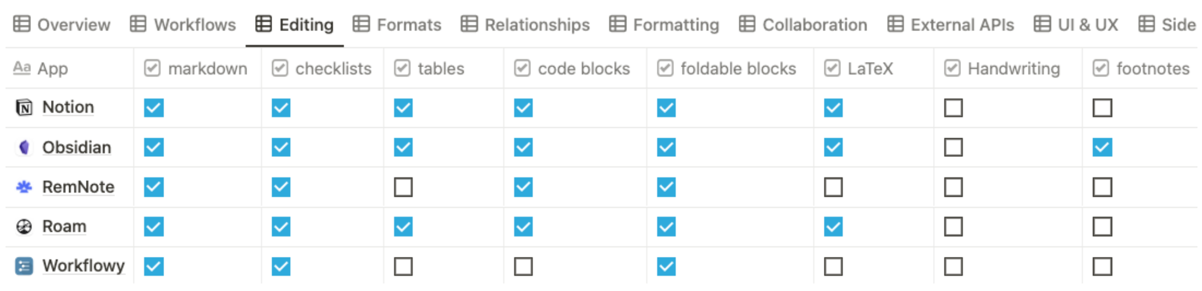
Figure 4.1 shows a general overview of the software landscape. We can see that all of these tools are based on proprietary data formats. Most of them store data in the cloud to enable features like multi-platform sync and collaboration.


Overview
Workflows
Editing
Formats
Relationships
Formatting
Collaboration
External APIs
UI & UX
Sidebar
Feat

App	Tasks & Reminders	Journaling	Databases	Associate pages	Publish	Collaboration	Graph view	Presentation
Notion	✓	✓	✓	✓	✓	✓	☐	☐
Obsidian	✓	✓	☐	✓	✓	✓	✓	✓
RemNote	☐	✓	☐	✓	☐	☐	✓	☐
Roam	✓	✓	☐	✓	✓	✓	✓	✓
Workflowy	✓	✓	☐	✓	✓	✓	☐	☐

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

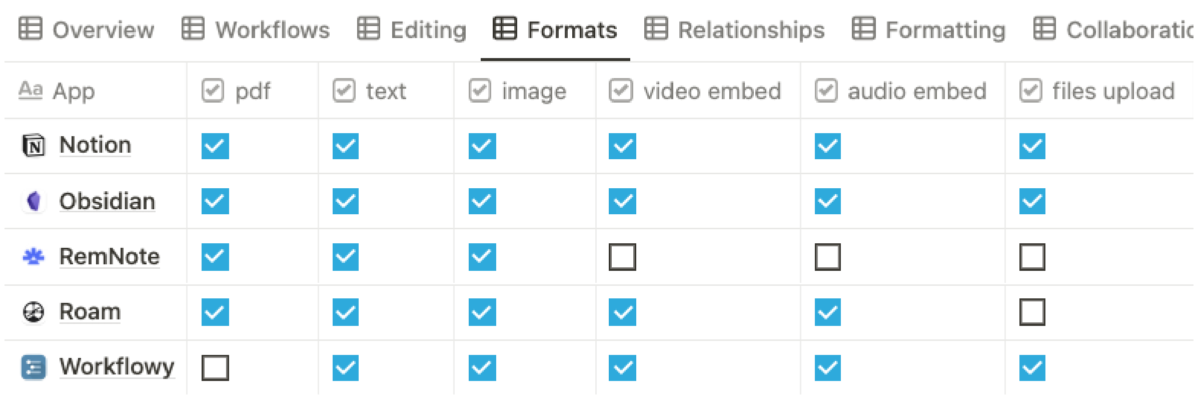
Figure 4.2 shows the workflows these tools enable. Most of the tools allow task management, journaling, and collaboration. Notably, only Notion includes databases with views and filters.


Overview
Workflows
Editing
Formats
Relationships
Formatting
Collaboration
External APIs
UI & UX
Side

App	markdown	checklists	tables	code blocks	foldable blocks	LaTeX	Handwriting	footnotes
Notion	✓	✓	✓	✓	✓	✓	☐	☐
Obsidian	✓	✓	✓	✓	✓	✓	☐	✓
RemNote	✓	✓	☐	✓	✓	☐	☐	☐
Roam	✓	✓	✓	✓	✓	✓	☐	☐
Workflowy	✓	✓	☐	☐	✓	☐	☐	☐

Table 4.3: PKM tool editing – text processing capabilities

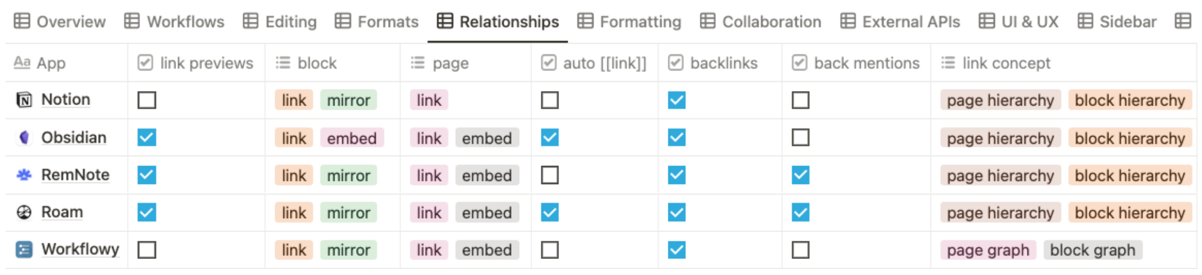
Figure 4.3 shows the text editing capabilities of the different tools.


Overview
Workflows
Editing
Formats
Relationships
Formatting
Collaboratic

App	pdf	text	image	video embed	audio embed	files upload
Notion	✓	✓	✓	✓	✓	✓
Obsidian	✓	✓	✓	✓	✓	✓
RemNote	✓	✓	✓	☐	☐	☐
Roam	✓	✓	✓	✓	✓	☐
Workflowy	☐	✓	✓	✓	✓	✓

Table 4.4: PKM tool formats – interoperability with file formats

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


Overview
Workflows
Editing
Formats
Relationships
Formatting
Collaboration
External APIs
UI & UX
Sidebar

App	link previews	block	page	auto [[link]]	backlinks	back mentions	link concept
Notion	☐	link mirror	link	☐	✓	☐	page hierarchy block hierarchy
Obsidian	✓	link embed	link embed	✓	✓	☐	page hierarchy block hierarchy
RemNote	✓	link mirror	link embed	☐	✓	✓	page hierarchy block hierarchy
Roam	✓	link mirror	link embed	✓	✓	✓	page hierarchy block hierarchy
Workflowy	☐	link mirror	link embed	☐	✓	☐	page graph block graph

Table 4.5: PKM tool linking – relationships between knowledge elements

Figure 4.4 shows how the tools manage the granularity of their pages, blocks or entities and how they link between them. This concept is explained in more detail in chapter 5.

4.3 Comparison

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. As the entire human experience is based on a linear flow of time, this seems to be a natural approach for arranging information too. It also allows every consumer of the information to experience it in the same way. However, information being constrained to a linear arrangement is not always optimal. Sometimes one wants to get an overview of a topic as quickly as possible or follow an information path based on relevance or personal interest. For cases like this spatial arranged information and interactive media are optimal, where you can instantly see the big picture and zoom in and out on information. Consider the example of diagrams, maps or websites like Wikipedia. This non-linear arrangement no longer results in consumers having the same experience and the sense of orientation might be lost. In this chapter, we will argue that with today's technology it is possible to combine the benefits of both these historical approaches by leveraging a flexible data model and interfaces that can provide different views and visualizations of information.

5.1 Interpreting Linear Text as a Graph

An important realization with a linear presentation of information is that it can be interpreted as an ordered tree. A text document 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. However, a tree is just a special case of a graph, so we can conclude that this type of linear unstructured information can also be modeled with a graph. Visual examples of this appraoch are shown in Figure 5.1, for the ordered tree. In Figure 5.2 we show the Interface of a PKG tool (the prototype we developed) side-by-side with how its data would be represented in an RDF graph.

In computer science, more powerful ways to store information have been developed. Structured and semi-structured information is stored with XML, HTML, 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, schemas and transclusion. These advanced data models can also be interpreted as a Graph.

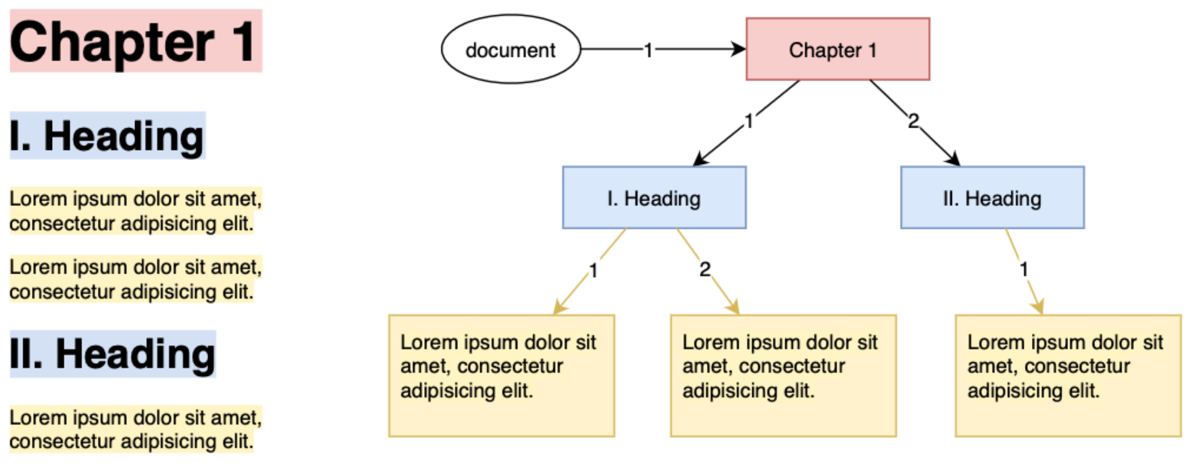

Chapter 1
I. Heading
Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Lorem ipsum dolor sit amet, consectetur adipisicing elit.
II. Heading
Lorem ipsum dolor sit amet, consectetur adipisicing elit.
document
1
Chapter 1
1
2
I. Heading
II. Heading
1
2
1
Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Figure 5.1: Representing linear information as text (left) and ordered tree (right)

A graph is thus a very flexible, powerful and interoperable data model for storing information and knowledge. If designed correctly, a graph could leverage the strength of both unstructured text documents and structured database technologies, while serving as the data model for the PKG ecosystem.

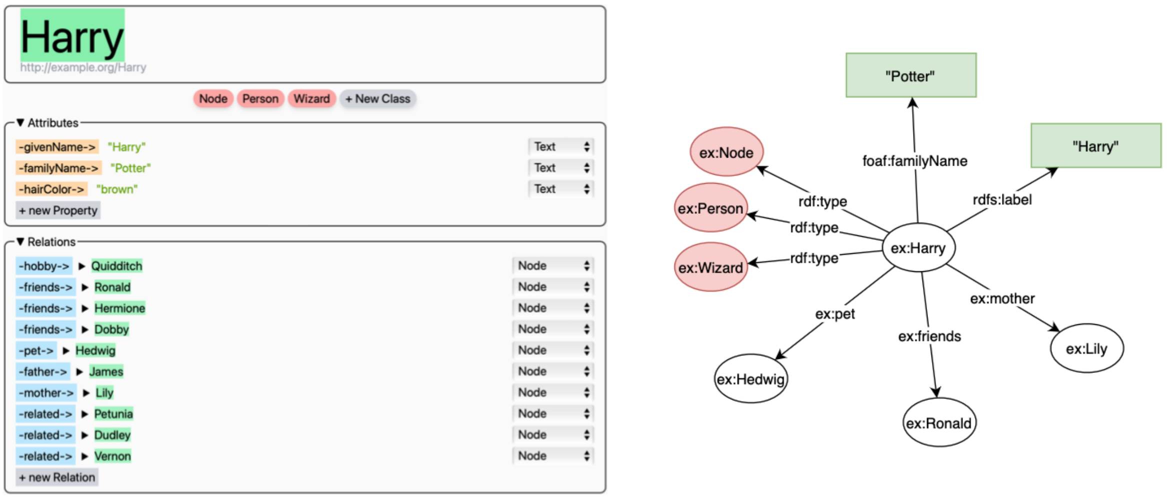
5.2 Data and Storage Model

An important consideration, when talking about data and storage models, is the granularity of the data that can be represented. This granularity is essential, because it defines what knowledge can be expressed and associated in a PKG. As outlined in chapter 4, we require a data model that is flexible and powerful, able to express atomic and complex concepts and freely associate them.

We investigate the granularity of the smallest entities that different data and storage models allow us to associate with each other. Depending on the context these entities have different names. For files, they are called documents, pages, paragraphs or blocks, for databases they are called rows or cells and for graphs they are called nodes, vertices, entities, or resources. We adopt the term knowledge element for the smallest linkable data element in this context [4].

In chapter 4 we saw that current PKG tools are based on proprietary formats, most often based on text files. The knowledge elements in this case are documents or pages. Even though these tools can link to paragraphs, these paragraphs are bound to the document. Each paragraph can only belong to one document, and each document can only belong in one folder in the file system. If the document or folder is deleted, all paragraphs inside are also deleted, turning all links to those elements into null pointers. This means that in practice text files are limited to only linking to other documents. To model atomic concepts, each concept would need to be a separate document. This approach would be similar to RDF graphs, except that instead of IRIs you were linking actual files. This would be a performance and maintenance nightmare. These limitations with links between file hierarchies rule out text files as the storage model for PKGs.

Considering this, we choose RDF graphs as the data model and a graph database as the storage model for PKGs. RDF graphs are an open standard and already support a wide array of mature technologies, like query languages, description logic, and external schema that will be of use for PKGs [21], [24]. Graph databases are chosen because they are more performant when doing path traversals or recursive querys to navigate associated knowledge. Structured knowledge will be implemented with the RDF data model. Semi- and unstructured Knowledge will be modeled with Notes using semantic markdown, as proposed in chapter 6.



Harry
http://example.org/Harry

Node Person Wizard + New Class

▼ Attributes

-givenName->	"Harry"	Text
-familyName->	"Potter"	Text
-hairColor->	"brown"	Text
+ new Property		

▼ Relations

-hobby->	▶ Quidditch	Node
-friends->	▶ Ronald	Node
-friends->	▶ Hermione	Node
-friends->	▶ Dobby	Node
-pet->	▶ Hedwig	Node
-father->	▶ James	Node
-mother->	▶ Lily	Node
-related->	▶ Petunia	Node
-related->	▶ Dudley	Node
-related->	▶ Vernon	Node
+ new Relation		

Figure 5.2: Pkg tool interface (left) and RDF graph (right)

5.3 RDF Dataset Layers

The PKG data model is an RDF graph dataset with several Layers:

Static external schema. 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 mergeable 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 produced by notes, reasoning, inverted links, etc.

Metadata. This data layer can be used by applications to save metadata like the order of notes or display properties like highlights, alignment, etc.

5.4 CRUD Operations and Effects

In this section, we outline the effect of create, read, update and delete (CRUD) operations on the data model.

These operations are illustrated by pseudocode similar to the RDFJS spec [14]. Quads are RDF triples with an optional graph IRI. Dataset is a quad store. Function calls have their parameters indented. Quads are in a syntax similar to Turtle, but with an additional graph term and separated by newlines.

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, we 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 their display order. 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 became infamous 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 became infamous 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 markdown 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.

In the next chapter, we look at how this markdown attached to the RDF graph can be used to mix structured and unstructured knowledge into the PKG data layers. We will propose an extension of markdown called semantic markdown.

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 standardized, 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 [9], [12], [13], [18]. 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
- `$$` gets converted to LaTeX
- `^^` highlights text

A similar appraoch to hypertext markdown could be used as a syntax for creating RDF triples in markdown, which we will propose in the next section.

6.2 A Semantic Markdown Extension

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, we looked at special characters easily reachable on both a physial and digital ANSI keyboard. we 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, like this:

```
[[
"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.

6.3 Outlook

The semantic markdown proposed in this chapter could be used to infer RDF statements from plain text through the use of a special syntax. These RDF statements would always connect to all the parents of the `<:note>` (recursively), by using the parents IRI as the subject for the inferred RDF triples. This would enables the triples to be present on all parent IRIs.

Semantic markdown would need to be standardized to be useful in combination with RDF or other data models.

7 A Prototype PKG Tool

While many challenges of PKGs can be addressed with the data model and architecture, the User Experience (UX) aspect needs to be considered separately. Semantic web technology is hard to understand even for Computer Scientists, as even the W3C acknowledges [5]. Expecting non-technical people to grasp all the concepts semantic web standards rely on is unrealistic. For personal use by non-technical people, these concepts need to be abstracted away with a self-explanatory User Interface (UI) aided by tutorials.

In this chapter, we present a prototype that aims to be user-friendly and not rely on previous knowledge of semantic web technologies.

7.1 Tech-Stack

The prototype is implemented as a web application using HTML, CSS, TypeScript and the frameworks TailwindCSS and Svelte.

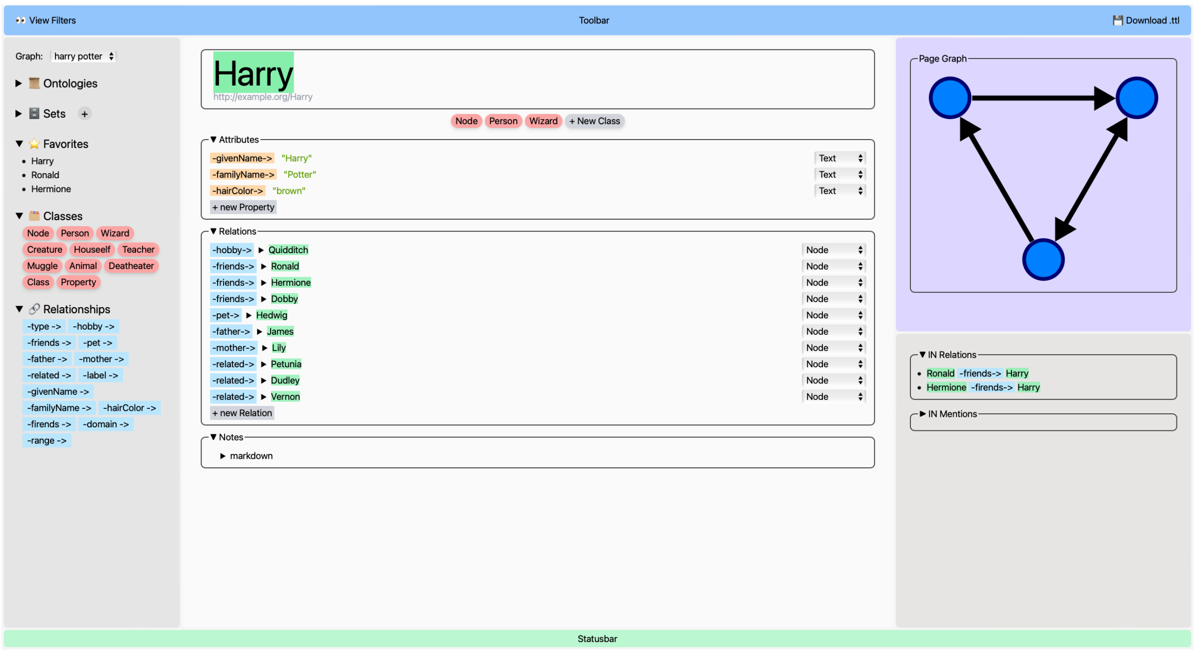
The RDF data is stored in memory by using our implementation of the RDFJS specification called RDFun found in appendix A [14]. RDFJS is a specification for working with RDF in JavaScript. A host of Libraries exist that provide functionality for RDF graph datasets that are compliant with RDFJS. We used libraries for fetching and querying external RDF data [3].

7.2 Usability Considerations

The prototype is implemented as a single-page web application. This enables a frictionless setup without needing to download or configure a client.

No knowledge of semantic web technologies should be necessary to use the app, even though it is based on an RDF graph.

7.3 The User Interface



Graph: harry potter

Favorites
- Harry
- Ronald
- Hermione

Classes: Node, Person, Wizard, Creature, Houseelf, Teacher, Muggle, Animal, Deatheater, Class, Property

Relationships: -type ->, -hobby ->, -friends ->, -pet ->, -father ->, -mother ->, -related ->, -label ->, -givenName ->, -familyName ->, -hairColor ->, -firends ->, -domain ->, -range ->

Harry
http://example.org/Harry

Node Person Wizard

Attributes

-givenName->	"Harry"	Text
-familyName->	"Potter"	Text
-hairColor->	"brown"	Text

Relations

-hobby->	Quidditch	Node
-friends->	Ronald	Node
-friends->	Hermione	Node
-friends->	Dobby	Node
-pet->	Hedwig	Node
-father->	James	Node
-mother->	Lily	Node
-related->	Petunia	Node
-related->	Dudley	Node
-related->	Vernon	Node

IN Relations
- Ronald -friends-> Harry
- Hermione -firends-> Harry

Figure 7.1 shows a screenshot of the UI. It is divided into several areas, 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.

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

Open and enterprise KGs are established research areas. Their construction, enrichment, schema and context are heavily based on consensus. For open KGs, a consensus is needed to make them interoperable with other KGs from the same domain, and for adhering to linked open data principles. We think it is no coincidence that the maintenance of open knowledge graphs is closely connected to other consensus-based efforts like Wikipedia. Enterprise KGs are proprietary and instead of being built from the established consensus in a domain, they define what the consensus for data representation in their application area is. This approach is closer to strict database schema.

We believe that PKGs will take a different direction regarding consensus. The use case for consensus in PKGs will probably be about smaller communities that share consensus about a certain topic. For example, a hobbyist community could build a KG that individuals can clone, integrate into their PKG and personalize to their liking. We also regard PKGs as a promising approach in education. Rather than the linear and chronological information representations students are currently served, teachers might hand out knowledge graphs. These can be traversed based on interest and relevance to the individual student, which we think might result in more enthusiastic interaction with the material. Students could even be split into groups to work on small knowledge graphs that would be merged at the end of the course.

All of this hinges on everyday people having better access to personal knowledge management and structured knowledge. We think that pursuit of the ideas presented in this thesis might not only help people with their personal knowledge management, but also cause them to take more interest in opening up knowledge proliferation. A similar effect can be observed with software developers, who are required to work and think collaboratively regarding large codebases. They developed incredibly strong solutions for data reuse and composition which resulted in the most amazing open source projects.

9 Conclusions

In this thesis, we have investigated PKGs and the PKG ecosystem as approaches to personal knowledge management and the information overload problem. We have outlined the functionalities required of the PKG ecosystem and established that PKM tools based on proprietary formats or text files are not viable candidates for the PKG ecosystem. We have proposed RDF as the data model for PKGs and outlined how to store both structured and unstructured data with it. To achieve this we proposed the development of a markdown extension that can express RDF statements. As a proof of concept, we implemented the prototype of a web application that can store structured and unstructured knowledge as an RDF graph.

given more time, we would have developed the prototype with more of a focus on usability by including user tests.

Version control and access rights for PKGs are also topics that we would have liked to investigate in more detail.

Limitations in our approach are that we could not create standards for the RDF data model or semantic markdown. This would require more resources and consensus.

The SOLID project is an initiative by Tim Berners Lee to provide individuals with online data stores, so-called "Pods". These pods could potentially serve as the backend we envisioned for the PKG ecosystem, because they use the RDF data model, and are compatible with several graph query languages [20].

A The RDFun Javascript Library

RDFun is a JavaScript Library we wrote that implements the RDFJS spec [14].

```
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

List of Figures

List of Tables

List of Code

References

[1] K. Balog and T. Kenter, “Personal knowledge graphs: A research agenda,” *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, 2019.

[2] V. Bush *et al.*, “As we may think,” *The atlantic monthly*, vol. 176, no. 1, pp. 101–108, 1945.

[3] “Comunica: A knowledge graph querying framework for javascript.” (), [Online]. Available: https://github.com/comunica/comunica (visited on 07/15/2022).

[4] S. Davies, J. Velez-Morales, and R. King, “Building the memex sixty years later: Trends and directions in personal knowledge bases ; cu-cs-997-05,” 2005.

[5] “Easier RDF.” (), [Online]. Available: https://github.com/w3c/EasierRDF (visited on 07/15/2022).

[6] D. C. Engelbart, “Augmenting human intellect: A conceptual framework,” 1962.

[7] A. Hogan, E. Blomqvist, M. Cochez, *et al.*, “Knowledge graphs,” *Communications of the ACM*, vol. 64, pp. 96–104, 2021.

[8] W. P. Jones and J. Teevan, *Personal information management*. University of Washington Press Seattle, 2007, vol. 14.

[9] “Logseq.” (), [Online]. Available: https://logseq.com (visited on 07/15/2022).

[10] A. Matuschak and M. A. Nielsen, “How can we develop transformative tools for thought,” 2019.

[11] MichaelK.Bergman. “A common sense view of knowledge graphs.” (2019), [Online]. Available: http://www.mkbergman.com/2244/a-common-sense-view-of-knowledge-graphs/ (visited on 07/15/2022).

[12] “Notion.” (), [Online]. Available: https://www.notion.so (visited on 07/15/2022).

[13] “Obsidian.” (), [Online]. Available: https://obsidian.md (visited on 07/15/2022).

[14] “Rdf javascript libraries.” (), [Online]. Available: http://rdf.js.org (visited on 07/15/2022).

[15] “Remnote.” (), [Online]. Available: https://www.remnote.com (visited on 07/15/2022).

[16] “Resource description framework.” (), [Online]. Available: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (visited on 07/15/2022).

[17] "Resource description framework schema." (), [Online]. Available: https://www.w3.org/TR/rdf-schema/ (visited on 07/15/2022).

[18] "Roam research." (), [Online]. Available: https://roamresearch.com (visited on 07/15/2022).

[19] T. Safavi, C. Belth, L. Faber, D. Mottin, E. Müller, and D. Koutra, "Personalized knowledge graph summarization: From the cloud to your pocket," *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 528–537, 2019.

[20] "Solid project." (), [Online]. Available: https://solidproject.org (visited on 07/15/2022).

[21] "Sparql protocol and rdf query language." (), [Online]. Available: https://www.w3.org/TR/sparql11-query/ (visited on 07/15/2022).

[22] "Terse rdf triple language." (), [Online]. Available: https://www.w3.org/TR/turtle/ (visited on 07/15/2022).

[23] I. Velitchkov. "Ontologies for roam research." (2022), [Online]. Available: https://github.com/kvistgaard/roamo (visited on 07/15/2022).

[24] "Web ontology language." (), [Online]. Available: https://www.w3.org/TR/owl2-overview/ (visited on 07/15/2022).

[25] "Wikipedia: Graph theory." (), [Online]. Available: https://en.wikipedia.org/wiki/Graph_theory (visited on 07/15/2022).

[26] "Xml schema definition." (), [Online]. Available: https://www.w3.org/TR/xmlschema11-1/ (visited on 07/15/2022).