future internet

MDPI

Article

A Graph DB-Based Solution for Semantic Technologies in the Future Internet

Stefano Ferilli *, Eleonora Bernasconi, Davide Di Pierro and Domenico Redavid

Department of Computer Science, University of Bari, 70125 Bari, Italy; eleonora.bernasconi@uniba.it (E.B.); davide.dipierro@uniba.it (D.D.P.); domenico.redavid1@uniba.it (D.R.)
* Correspondence: stefano.ferilli@uniba.it; Tel.: +39-080-544-2293

Abstract: With the progressive improvements in the power, effectiveness, and reliability of AI solutions, more and more critical human problems are being handled by automated AI-based tools and systems. For more complex or particularly critical applications, the level of knowledge, not just information, must be handled by systems where explicit relationships among objects are represented and processed. For this purpose, the knowledge representation branch of AI proposes Knowledge Graphs, widely used in the Semantic Web, where different online applications may interact by understanding the meaning of the data they process and exchange. This paper describes a framework and online platform for the Internet-based knowledge graph definition, population, and exploitation based on the LPG graph model. Its main advantages are its efficiency and representational power and the wide range of functions that it provides to its users beyond traditional Semantic Web reasoning: network analysis, data mining, multistrategy reasoning, and knowledge browsing. Still, it can also be mapped onto the SW.

Keywords: graph databases; ontologies; Knowledge Graphs; Semantic Web; knowledge representation and reasoning; graph mining; network analysis; knowledge exploration

Citation: Ferilli, S.; Bernasconi, E.; Di Pierro, D.; Redavid, D. A Graph DB-Based Solution for Semantic Technologies in the Future Internet. *Future Internet* **2023**, *15*, 345. https://doi.org/10.3390/fi15100345

Academic Editors: Vincenza Carchiolo and Marco Grassia

Received: 11 September 2023
Revised: 12 October 2023
Accepted: 16 October 2023
Published: 20 October 2023

1. Introduction

Artificial Intelligence (AI) is becoming more and more pervasive. The subsymbolic (mathematical/statistical) approach to AI provides efficient and effective solutions that are not easily interpretable or explainable. Symbolic approaches, based on formal logic, are required to improve trustworthiness and ensure uniform knowledge interpretations. Concerning Internet applications, the Semantic Web (SW) solutions and standards ensure system interoperability, automatic service composition, and automated reasoning facilities. Such a high level of data storage and exploitation determines the so-called Knowledge Bases (KBs), where 'data' (simple values) and 'information' (interpreted data) are raised to the level of 'knowledge' (interconnected information, where the whole is more valuable than just the sum of the parts).

The SW focuses on ontologies as the key to providing formal definitions of the available information items and develops specific formalisms and knowledge storage systems based on graph representations of knowledge, the so-called 'Knowledge Graphs' (KGs). Sometimes, KGs are intended to encompass both the ontology and the instances, and sometimes, they are intended to be limited to the instances only. Thanks to the ontology, some automated reasoning tasks (such as inheritance, consistency checks, etc.) can be applied to the KG, and various tools have been developed for this. We believe that, to support increases in the power, effectiveness, and reliability of AI solutions, the future of SW solutions should also rely on DB technologies and especially on graph DB ones to exploit their efficiency, especially when instance-based information exploitation is required, as opposed to batch information processing (satisfactorily supported by traditional relational DBs). An obvious link between these two technologies is their basis on graph models, albeit on

different and partly incompatible ones: the SW adopts the RDF graph model, and the most prominent graph DB adopts the Labeled Property Graph (LPG) model. The latter allows more expressiveness and produces more compact graphs than the former.

This paper proposes a framework that covers the whole range of layers, from back-end machinery to end-user interfaces, and the whole range of functions, from knowledge representation (defined by the GraphBRAIN framework), base creation, population, and maintenance to network mining and analysis (carried out by the GraphBRAIN system), to interoperability among different SW applications (guaranteed by a purposely developed GraphBRAIN API), to Knowledge Base querying, browsing, and mining (provided by the SKATEBOARD system). Moreover, we propose a mapping mechanism to port data expressed in the LPG into the RDF to create a bridge between GraphBRAIN and the SW.

GraphBRAIN uses an LPG-based knowledge representation that can be mapped directly onto the Neo4j [1] graph DB for instant storage and manipulation and acts as a schema for the DB content. An API, which can be used by any application that adopts the GraphBRAIN framework, wraps the DB and ensures that all operations carried out on it are compliant with a given ontology. A Web Application is available for form-based information entry and retrieval and for user-friendly execution of the basic CRUD operations. The GraphBRAIN framework acts as an intermediate format that can be easily mapped onto different technologies:

- The Neo4j libraries for graph manipulation, querying, and mining;
- Advanced MultiStrategy Reasoning tools that combine different inference strategies to tackle more complex problems and closely simulate human reasoning (e.g., as provided by GEAR [2]);
- Standard SW representation techniques and reasoning tools to be interoperable with the existing SW world.

One of the end-user applications powered by GraphBRAIN is SKATEBOARD, which was purposely developed to support semantic-based, interactive, and dynamic browsing of the KG. It provides various filters for the information to be displayed, tools for expanding the portion of the graph under consideration and looking into the single graph components for their information content, and additional facilities, such as maps, that allow easy location-based analysis and browsing of the data.

The novel contributions of this paper are

1. A description of the current version of the GraphBRAIN knowledge representation structure and formalism, which extends and refines that proposed in [3];
2. A description of a strategy to map the GraphBRAIN knowledge representation into the standard language for ontologies used on the Web (OWL);
3. A description of the API with its latest functions in more depth than that presented in [4];
4. An introduction to the SKATEBOARD interface and the end-user-related functions of GraphBRAIN.

In essence, the key contribution of GraphBRAIN lies in its ability to bridge the gap between graph databases and ontologies, enabling efficient data handling while also facilitating advanced reasoning and analysis, making it a versatile tool for managing complex Knowledge Graphs with rich semantics. Furthermore, it offers modular ontology handling, allowing for the integration of existing ontologies or the creation of new ones, enhancing the adaptability of the framework to various domains and specific requirements.

This paper is structured as follows: After reviewing related work in Section 2, Section 3 describes the GraphBRAIN framework and its features and functionalities. The relationship between GraphBRAIN and standard Semantic Web technology is discussed in Section 4, while Section 5 presents the general interfaces within GraphBRAIN. Then, in Section 6, we evaluate the framework's performance and capabilities before drawing our conclusions in Section 7.

2. Related Work

In this section, we delve into various facets of related research and technologies that have paved the way for the development of GraphBRAIN.

2.1. RDF and LPG Models for Knowledge Graphs

Concerning the possibilities for cooperation between ontologies and graph DBs, there is a recognized need for, but limited adoption of, logic-based KR for the development of KGs [5]. Most theoretical works specifically deal with the mapping from OWL- to LPG-based graph DBs, but are always biased towards ontologies without leveraging the extended structure and expressiveness of LPGs with respect to RDF graphs: G2GML [6], OWL2LPG [7], SciGraph [8], and VirtualFlyBrain [9]. GraphBRAIN takes the opposite perspective and, considering that DB technology is more mature and widespread than the ontology one, aims to preserve the DB structure and organization, superimposing the ontology.

OWLStar [10] exports Neo4j to OWL using RDF*, but still with an OWL-centric perspective. While there is a formal mapping between LPGs and RDF* that would allow us to keep the data in the DB and render them in RDF* [11], RDF* and its query language SPARQL* have not yet been proposed as W3C recommendations. GraphBRAIN overcomes this limitation by defining a mapping of LPGs onto a standard RDF. Some solutions, as discussed in the Neo4j community blog [12], export Neo4j instances to the RDF. For example, they do this upon the request of an ontological reasoner. However, these solutions may not guarantee that only data compliant with the intended ontology are inserted into the database.

The GraphBRAIN technology tackles all of the above issues by superimposing a schema to Neo4j DBs in the form of an ontology to enable high-level reasoning on the available knowledge and still benefit from the advantages provided by graph DBs (scalability, storage optimization, efficient handling, mining and browsing of the data, etc.) and LPGs (flexibility, expressive power) and to exploit ontological reasoners. By referring to a schema, the applications commit to compliance with it, as in traditional DBs. Like in Triplestores and RDF*, this will ensure tight integration between the data and the schema. As opposed to Triplestores, the RDF* and most of the cited works, where the ontology is ingested in the graph, the data/instances (stored in the graph DB) are kept apart from the schema/ontology (specified in a file external to the DB using an ontological representation format). This separation between the data repository and the schema leads to another peculiarity: different (but compatible) schemas can be applied to one DB. Again, this is not even thinkable in Triplestores.

Since the RDF and LPG are graph models, their interoperability has been investigated [13], but all proposals for mapping LPG DBs to or from an RDF offer predefined, non-customizable mapping. This is a significant limitation, as the structure and naming conventions adopted in an LPG schema can differ from those that are desirable for the RDF view of the data. GraphBRAIN proposes an extension to RML that allows structural mapping between LPG and RDF elements and provides an ontological alignment between local resources and others known to refer to specific standard concepts. This proposal bridges the gap to the SW perspective and allows us to use of graph DB facilities, algorithms, and knowledge representation tools. When local resources are directly mapped onto OWL/RDF KBs, they can be managed as regular SPARQL endpoints [14] or by ontological reasoners [15].

2.2. Linked Data Interfaces

Here, we explore the tools available for knowledge extraction, information retrieval, and semantic data visualization and the use of semantic technologies at large to draw a comparison of the state-of-the-art tools with the SKATEBOARD interface for Linked Data exploration of the GraphBRAIN platform.

A concise overview of the main distinctions between the GraphBRAIN framework and the analyzed works is provided in Table 1.

Table 1. Advantages of GraphBRAIN compared to other frameworks/initiatives.

Related Work	Key Features	GraphBRAIN
AIDA [16]	Specific focus on Named Entity Recognition (NER) and semantic annotation	Broader range of knowledge extraction and visualization capabilities
Apache Stanbol [17]	Knowledge extraction and semantic enrichment	Extended capabilities for semantic data retrieval and exploration
DBpedia Spotlight [18]	Entity recognition and linking to DBpedia	More comprehensive knowledge extraction and visualization
Open Calais [19]	Text analytics and entity recognition	Advanced semantic browsing and visualization features
Semiosearch Wikifier [20]	Entity recognition and semantic annotation	Robust semantic data exploration tools
GLOBDEF [21]	Comprehensive knowledge extraction and enrichment	Broader range of visualizations and semantic browsing
L'ERMA [22]	Traditional visual data retrieval tools	Enhanced semantic faceted browsing and tailored visualizations
TORROSSA [23]	Traditional visual data retrieval tools	Innovative semantic exploration capabilities

The focus of knowledge extraction is on using advanced techniques to transform unstructured text into meticulously structured knowledge representations. The significance of knowledge extraction in the proficient management of semantic data is unmistakable as it encompasses both information extraction and semantic enrichment. To achieve this, various tools harness the capabilities of natural language processing, machine learning, and knowledge representation. Among the prominent contenders in this field are AIDA [16], Apache Stanbol [17], DBpedia Spotlight [18], Open Calais [19], Semiosearch Wikifier [20], and the GLOBDEF system [21]. Each excels in specific facets of knowledge extraction. However, SKATEBOARD distinguishes itself by offering a comprehensive solution. It adeptly transforms unstructured text into structured data, identifies entities through Named Entity Recognition (NER), establishes connections with Knowledge Bases like DBpedia [24], and enhances data through semantic annotation. SKATEBOARD has emerged as a pragmatic choice for managing Linked Data, providing an inclusive solution that spans the entire knowledge extraction pipeline.

In the arena of traditional visual information-seeking tools, venerable names such as L'ERMA [22] and TORROSSA [23] have long played pivotal roles in visual data retrieval. However, the landscape underwent a remarkable transformation with the introduction of SKATEBOARD, empowered by cutting-edge semantic technology. SKATEBOARD transcends traditional paradigms by enabling semantic faceted browsing, fostering serendipitous discovery, and providing intelligent recommendations. These capabilities significantly enhance the user experience, redefining the search boundaries.

The domain of visualizing semantic data presents distinct challenges due to its diverse and dynamic nature. Prior research has diligently categorized visualization tools based on interaction paradigms and the types of information they represent. Interaction paradigms encompass a range of options:

- The Tabular Interaction Paradigm: This format organizes information about a single resource in a tabular layout, allowing users to explore specific attributes such as media files, descriptions, and links to related resources.
- The Node-Link Interaction Paradigm: In this paradigm, resources are represented as nodes or boxes connected by arcs symbolizing relationships. Users navigate the data by following these connections.
- The Visual Query Composition: These interfaces simplify the creation of SPARQL queries through graphical elements.

Regarding the types of information represented, the distinctions are clear:

- Data Visualization: These tools leverage graphical representations to enhance data comprehension.
- Model Visualization: This category specializes in illustrating data models, including schemas and ontologies, providing insights into underlying data structures.
- Data to Model Visualization (Schema Extraction): These tools deduce ontology schemas from RDF triples via SPARQL queries.

However, SKATEBOARD goes beyond these conventions and classifications by introducing a flexible approach tailored to specific types of entities. For example, it provides interactive maps for geographical locations, unveils intricate details like birthdates, birthplaces, and essential biographical information for individuals, and offers extensive publication data and book author profiles. SKATEBOARD's capacity to tailor visual representations according to entity types enriches the exploration of semantic information, providing a more user-friendly experience. SKATEBOARD is a versatile solution, featuring a comprehensive approach to knowledge extraction, seamless conversion of unstructured textual data into structured formats, and dynamic visualizations.

2.3. Bridging LPG and RDF

The Semantic Web (SW) and RDF were conceived for different purposes than the traditional uses of local DBs. DBs focus on efficiency and scalability, while the SW focuses on interoperability and data availability. The integration between these two technologies, designed to take advantage of the strengths of both, has attracted interest since the early days of the WWW because, while the SW provides a means to share and integrate data, various factors often prevent the public distribution of data in many contexts. However, problems are posed by the structural differences between the graph models used in the SW and in DBs.

Some approaches [25–27] have investigated how to collect the data available in the SW and import them into local graph DBs. This is especially relevant when there is a need to create shared knowledge on a specific topic that end-users can query. When mapping is carried out in this direction, the primary advantage is that queries in graph DBs are optimized for quicker data retrieval in a more centralized and controlled domain. However, this approach may limit the possibility of integrating this knowledge with external sources and supporting automated reasoning. Additionally, the Open-World Assumption, a fundamental aspect of the SW, may not apply in such cases.

In the opposite direction (from DBs to RDF), the problem has been extensively addressed, initially focusing on relational DBs [28,29], often using rule-based mappings [30,31]. Entities are often represented as classes, keys as relationships, and properties as attributes. While natural, this translation lacks generality and does not apply to NoSQL models. The initial attempts at personalization, such as those in [32], used XML to express the mapping between concepts. This effort produced two W3C standards:

- Direct mapping [33], i.e., the conversion of the relational model to the RDF through a set of fixed rules without the need for configuration;
- R2RML [34,35] (https://www.w3.org/TR/r2rml/, accessed on 9 September 2023), requiring a detailed configuration of the mapping and thus enabling highly customized

behavior. R2RML mappings consist of one or more triple maps, each with one logical table, one subject map, and zero predicate object maps.

Another straightforward strategy is available when the fixed structure of tables provides fewer mapping rules to RDF structures [36]. Ref. [37] tried to generalize the problem for graphs, but it does not allow us to customize the translation according to specific needs. The RML language [38] generalizes R2RML to the most commonly used file formats for structured data (CSV, JSON, and XML) but not to non-relational DBs. Inspired by works on R2RML [39–41] and RML [42,43], we propose an approach to map the concepts of the LPG. Given the need to personalize the mapping to avoid strict rules, our approach relies on a customizable language that can specify how to map concepts from one perspective to the other.

With the introduction of graph structures, research on the integration of different models has gained momentum, facilitated by both LPG and RDF being graph-based.

A good example of an overall integration infrastructure is OpenBiodiv [44], which combines semantic publishing workflows, text and data mining, common standards, ontology modeling, and graph DB technologies to manage biodiversity knowledge. It is presented as Linked Open Data (LOD) [45] generated from the scientific literature. It falls short in the integration (and disambiguation) of knowledge coming from many sources. In [46], a framework for the projection and validation of a portion of an RDF graph using the graph DB formalism is described. It allows a SPARQL query to be written, translating the result into JSON and parsing it into the LPG model. It acts more as a wrapper than as a bridge between the two technologies. In [6], a mapping language between Cypher and SPARQL is proposed based on static concept mapping. However, it does not really map classes and relationships in the graph with those available in the SW.

In many cases [47,48], different graphs have been dealt with by building many SPARQL endpoints in a federated architecture and then querying and fusing the results. Others [49,50] use graphical visual interfaces in which the SPARQL endpoints become transparent to the end user, like in SKATEBOARD, but differently from us, they do not have external connections to the graph DBs.

3. GraphBRAIN Framework

GraphBRAIN [3,51] is a framework for the management of KGs that has the vision of joining the efficiency in data handling provided by LPG-based graph DBs (specifically, Neo4j) with the expressive power of ontologies. One further objective of GraphBRAIN is to provide more handling possibilities than those provided by standard SW reasoners. These include automated multistrategy reasoning based on the First-Order Logic, Data Mining, and Network Analysis functions, interactive knowledge browsing, and human-understandable descriptions of the processing results.

In GraphBRAIN KGs, labels usually represent classes, nodes represent class instances, types represent relationships, and arcs represent relationship instances. Each node or arc is associated with the label representing the most specific class or relationship it belongs to. Nodes are also labeled with all domains for which the instance they represent is relevant. The ontologies provide high-level and formal interpretations of the data and enable semantic-aware automated reasoning. When used as DB schemes, they allow us to keep the graph consistent with user-defined constraints. For this reason, we use the terms *schema* and *ontology* interchangeably. As in traditional DBs, the schema is stored apart from the data and based on a different representation. GraphBRAIN proposes a new LPG-based ontological formalism, expressed in XML due to its simplicity, human interpretability, flexibility, and easy mapping onto different languages.

Ontology handling in GraphBRAIN is modular. Existing ontologies can be imported and extended in new ontologies defined by the knowledge engineers in charge of modeling a given domain (or they can be defined from scratch). GraphBRAIN already provides a general ontology with top-level, highly reusable classes and relationships, plus several

domain-specific ontologies. In fact, each schema represents a different, partial view of the same data, which can be limited or expanded depending on specific needs.

3.1. Architecture

The GraphBRAIN framework consists of several interconnected components (see Figure 1) that work together to manage Knowledge Graphs (KGs) efficiently and provide advanced reasoning and analysis capabilities. Here is an outline of the key components and their functions:

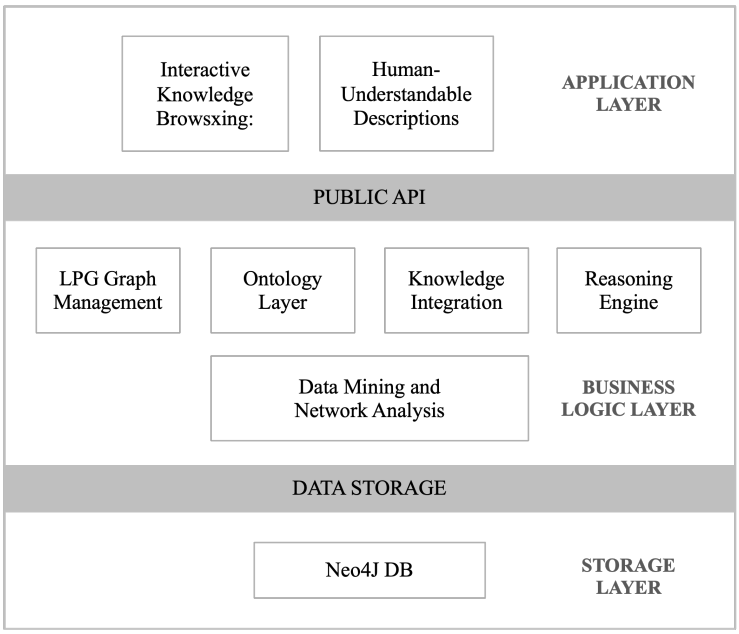

Figure 1. Architecture.

Data Storage: GraphBRAIN leverages Labeled Property Graph (LPG)-based graph databases, such as Neo4j, for efficient data storage. The data storage component manages the underlying KGs, where labels represent classes, nodes represent class instances, and arcs represent relationship instances.

Ontology Layer: The ontological layer provides formal interpretations of data and supports semantic reasoning. On the one hand, it manages the schema for KGs, ensuring that the data adhere to user-defined constraints. The ontological formalism, expressed in XML, allows flexibility and easy mapping to various languages. On the other hand, the Semantic Web technologies used, particularly the OWL and RDF standards, enable semantic interoperability with the WWW.

Knowledge Integration: This component integrates existing ontologies and allows knowledge engineers to create new domain-specific ontologies. It offers a modular approach, enabling the import and extension of ontologies, as needed.

Reasoning Engine: GraphBRAIN goes beyond standard Semantic Web (SW) reasoners by incorporating automated multistrategy reasoning based on First-Order Logic. This component enables advanced logical inference and reasoning capabilities on the KGs.

Data Mining and Network Analysis: The framework includes data mining and network analysis functions, which allow users to extract patterns, insights, and meaningful connections from the KGs. These analytical tools enhance the understanding of the data.

Public API: GraphBRAIN functionalities are also accessible as a Java library. This makes it possible to build applications that meet specific requirements or simply customize the usability according to specific domains. Two example are

- **Interactive Knowledge Browsing:** GraphBRAIN offers an interactive knowledge browsing interface that enables users to explore the KGs intuitively. This component

provides a user-friendly way to navigate and query the knowledge stored within the system.

- **Human-Understandable Descriptions:** To enhance the usability, GraphBRAIN generates human-understandable descriptions of processing results. This feature aids users in interpreting and making informed decisions based on the KGs.

3.2. Summary of the GraphBRAIN Schema Formalism

The GraphBRAIN framework allows us to specify ontologies by providing an LPG-based structure and an associated formalism. The LPG-based structure allows a more intuitive description of the ontological components with a clear distinction among entities (or classes), relationships, and their attributes. Various options can be specified for the different components. It is formally specified as an XML file with the extension GBS (GraphBRAIN schema). In the following text, we provide brief descriptions of the latest version of this structure and the formalism.

The root tag of the XML file, enclosing all of the ontology elements, is **domain**, where the ontology name and attributes, such as creator and date, can be specified. This tag encloses four sections, each specifying a different ontological element.

The first schema fragment is identified by the tag **imports** and allows us to include other existing ontologies in the current ontology, in this way supporting reuse and interoperability. Each imported ontology is specified in an **import** tag. Imported ontologies must be compliant with each other and with the ontology being defined in the file, meaning that the hierarchical structure of the entities and relationships and the type of attributes must be consistent (e.g., if class C' is a subclass of class C'' in one ontology, C'' cannot be a subclass of C' in the other). The same entities, relationships, and attributes of different ontologies are identified by the same name.

The next fragment, identified by tag **types**, encloses user-defined datatypes, each of which can be a list of values (datatype Select) or a hierarchical organization of values (datatype Tree).

The third fragment, identified by the tag **entities**, defines the entities (whose instances will be the nodes in the graph), each enclosed in a tag **entity** with a *name* XML attribute (which will be the node label in the graph). The last fragment, identified by the tag **relationships** defines the relationships (whose instances will be the arcs in the graph), each enclosed in the tag **relationship** with *name* and *inverse* XML attributes, defining the name of the relationship (which will be the arc label in the graph, associated with the arc direction) and of its inverse (implicitly associated with the opposite arc direction in the graph). Additional Boolean XML attributes allow us to specify the relationship properties (e.g., if it is transitive). The relationship is implicitly assumed to be symmetric if its name is the same as the inverse name.

Each **entity** or **relationship** tag may enclose an **attributes** tag (mandatory for entities), where the various attributes of the entity or relationship can be defined (note that the RDF graph model used in the standard SW does not natively allow for relationship attributes) and, optionally, a **taxonomy** tag, where subclasses or subrelationships can be specified using nested occurrences of the **entity** tag. The tag **taxonomy** is recursive to allow the definition of hierarchies of classes or relationships, for which the inheritance of properties is supported. So, each subclass/relationship must only specify the additional attributes it introduces with respect to all of its ancestors (if any). **relationship** tags also include a **references** tag to specify the possible Subject–Object pairs for the relationship, each specified by a **reference** tag with XML attributes *subject* (a possible entity for the arc source) and *object* (a possible entity for the arc sink).

Each entity or relationship attribute is introduced by an **attribute** tag with the XML attributes **name** and **datatype**. The datatype can be a predefined one (GraphBRAIN provides Boolean, Integer, Real, String, Text, Select, Tree, Date, or Entity) or a user-defined one (defined in the **types** section of the XML file). All datatypes take a value, except for Entity and Date attributes, which actually establish an arc between the entity or relationship

instance and another node in the graph. The Entity datatype requires a further *target* XML attribute of the **attribute** tag, specifying the type allowed for the sink node of the arc. The Date datatype implicitly refers to the predefined entity 'TemporalSpecification' with its subclasses (e.g., Day, Month, Year, etc.). The datatypes Select and Tree specify a list or tree of values, such that the instances in the graph will be bound to take one of those values, or 'Other' if none of those options is satisfactory.

The last section, identified by the tag **axioms**, allows us to specify axioms in the form of logical formulas, each defined in an **axiom** tag. Axioms are constraints that the instances of the KG must fulfil at any time. Axioms determine the complexity, or even computability, of the ontology. The default reasoner provided by GraphBRAIN is GEAR, so the operators provided by GEAR will be allowed for sure. This part is currently still under development, so no more details can be provided at the time of writing.

3.3. GraphBRAIN Data Analysis & Mining Services

GraphBRAIN offers many services for handling data too. They can run several graph mining algorithms to obtain relevant indications of the graph content. Examples of the tools provided by GraphBRAIN are

- **The assessment of the relevance of nodes using several centrality assessment approaches (currently Closeness, Betweenness, PageRank, Harmonic, and Katz).** This function can be used to extract the most relevant nodes from the whole graph or from a selected portion. In turn, the most relevant nodes can be used as a starting point to select an initial portion of the graph to display when no specific indication is provided.
- **The extraction of a portion of the graph that is relevant to some specified starting points (nodes and/or arcs).** Given the starting points, different algorithms can be applied to select a subgraph (currently PageRank, Spreading Activation, and a proprietary algorithm that can take into account user profiles to determine a personalized subgraph that can include more relevant and interesting items based on the user background, interests, goals, etc.).
- **The extraction of frequent patterns and associated subgraphs.** Currently the g-Span and Spreading Activation algorithms are used for this purpose and specifically for clustering that returns relevant groups of nodes.
- **The prediction of links between nodes** based on different approaches (currently Resource Allocation, Common Neighbors, AdamicAdar, and Katz). This is useful for identifying possible connections between items that are not currently reported in the KG. One possible use of this feature is to suggest research directions to be investigated or to propose relevant knowledge that would not be found by simple graph browsing.
- **Pattern Mining** based on automated reasoning, which provides more expressive power than LPG-based query languages such as Cypher. For this, GraphBRAIN is connected to the GEAR [2] Multistrategy Reasoning engine, currently providing deduction, abduction, abstraction, induction, probabilistic, and similarity-based inferences.
- **Graph Understanding** via a logic-based formalization of the claims represented by the given portion of the graph. NLP generative techniques are used to translate them into natural language.

3.4. The GraphBRAIN API

GraphBRAIN is a general framework and technology that can be used by any application sharing its vision that is interested in taking advantage of its features. To connect GraphBRAIN to these applications, we developed a GraphBRAIN API that can be imported into any application to have all its tools and functions described in the previous sections: from ontology definitions to ontology-based knowledge access and manipulation and from semantic-aware data mining and network analysis to automated human-like reasoning on the available knowledge. It also provides import/export functionality from/to standard SW representations and exports functionality to several other formalisms. It is currently available for Java applications and is being ported to Python.

Through the API, any individual or organization can apply the GraphBRAIN framework to their own data and KGs. It takes the URL of a Neo4j instance (local or online) that stores the KG instances as the input, along with the associated access credentials and a GBS ontology and wraps the DB to enforce compliance of the data with the intended schemas in both building and consulting the DB. After establishing a connection to the KG, CRUD operations and queries can be requested and expressed in Cypher language, and the API delivers the corresponding results, just as for the underlying DB, while ensuring compliance with the selected ontology. Or, the exposed methods providing the other advanced functions can be run.

4. Connection between the LPG and SW

Exporting or exposing the resources in GraphBRAIN KGs, especially to standard SW applications, would enrich the global knowledge on the Web.

4.1. OWL/RDF Mapping

Since the SW setting requires every element in the KB to be uniquely identified by a URI, we defined the prefix `http://graphbrain.it/`*`ontologyName`* (for which the abbreviation used in this paper is *gb*) and obtained the URI of a resource as the concatenation of such a prefix with its element identifier in GraphBRAIN. Then, we defined a set of translation rules to map GBSs onto OWL ontologies based on the strategy reported in the following paragraphs.

(1) Entities, Entity Attributes, and Relationships

The entities, attributes, and relationships in a GBS are mapped onto classes, data properties, and object properties (respectively) in an OWL ontology using the obvious corresponding OWL elements *owl:class*, *owl:objectProperty* and *owl:dataTypeProperty*. Some examples extracted from a concrete-generated OWL ontology are reported in Listing 1.

Listing 1. Example of GraphBRAIN-generated export in OWL.

```
<!-- OWL Classes -->
<owl:Class rdf:about="gb:Place">
   <owl:disjointWith rdf:resource="gb:ProcessComponent"/>
   ...
</owl:Class>

<owl:Class rdf:about="gb:Administrative">
   <rdfs:subClassOf rdf:resource="gb:Place"/>
  ...
</owl:Class>

<!-- OWL Object properties -->
<owl:ObjectProperty rdf:about="gb:pertainsTo">
   <owl:inverseOf rdf:resource="gb:pertainedBy"/>
   <rdfs:domain rdf:resource="gb:Word"/>
   <rdfs:range rdf:resource="gb:Word"/>
</owl:ObjectProperty>

<!-- OWL Datatype properties -->
<owl:DatatypeProperty rdf:about="gb:codeISO_Administrative">
   <rdfs:domain rdf:resource="gb:Administrative"/>
   <rdfs:range rdf:resource="xsd:string"/
</owl:DatatypeProperty>
```

Other components or features of GBSs are not available in OWL. Specifically, relationship mapping deserves some explanation as it may generate inconsistencies in the ontology if not performed correctly.

(2) Relationships with multiple domain–range pairs

As previously mentioned, relationships are translated as object properties. While each object property must have a single domain and range, GBS allows several Subject–Object pairs for each relationship. To handle this, for each Subject–Object pair available for a

relationship 'rel', we introduce a different *owl:objectProperty* 'rel_subject_object', assigning its subject as its *owl:domain* and object as its *owl:range*. We then state that all of these newly added object properties are specializations of the *owl:objectProperty* 'rel', with the domain given by the union of the OWL classes identified by the subjects and the range given by the union of the OWL classes identified by the objects, e.g., suppose the GBS relationship 'rel' can be established between Subject–Object pairs (ClassA,ClassB) and (ClassC,ClassD), it translates to

```
ObjectProperty: rel
Domain: ClassA or ClassC
Range: ClassB or~ClassD

ObjectProperty: rel_ClassA_ClassB
Domain: ClassA
Range: ClassB

ObjectProperty: rel_ClassC_ClassD
Domain: ClassC
Range: ClassD
```

where *rel_ClassA_ClassB* and *rel_ClassC_ClassD* are specializations of 'rel'. You can see an example of this strategy in Figure 2. Note that 'rel' alone in OWL would allow the establishment of the relationship between an individual of ClassA and one of ClassD in the KB, which would be not allowed by the original GBS relationship. By adding the other two relationships, the OWL KG becomes inconsistent.

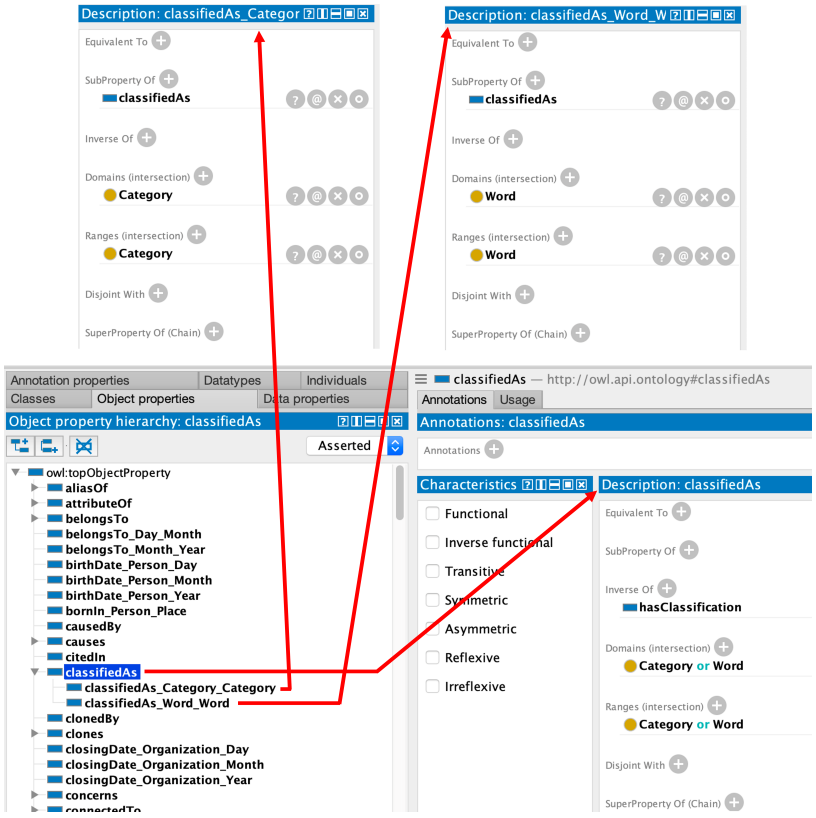

Figure 2. Protégé view of the classifiedAs object property and its specializations.

(3) Relationship attributes

For relationship attributes, we use an implicit reification: we introduce a class for each relationship in the schema and the **Relationship** class, which generalizes all of them. This allows us to represent the attributes of relationships as datatype properties of these newly reified classes. This is compliant with the official W3C guidelines [52] by applying the pattern relating to Use Case 1, where the requirement is to represent an additional

attribute describing an instance of a relation. Then, for each occurrence of the relationship, an individual representing the instance of the relation is created for the corresponding class with links toward the subject and object of the relation (*owl:subject* and *owl:object*, respectively) and other links to all items representing the original relationship attributes.

As an example, let us consider the object property 'wasIn' between the domain *gb:Collection* and the range *gb:Place*. To be able to specify why the collection was in that place, we create the class *gb:Reified_wasIn* as a subclass of the class *gb:Relationship*. Listing 2 these classes are used to specify all subproperties derived by applying the solution to the problem in the previous point (e.g., from the wasIn relationship in the XML schema, as many *wasIn_X_Y* properties are generated as there are pairs (domain X, range Y) specified for that relationship.)

Listing 2. Object property 'wasIn'.

```
<owl:ObjectProperty rdf:about="gb:wasIn">
   <owl:inverseOf rdf:resource="gb:hosted"/>
   <rdfs:domain>
      <owl:Class>
         <owl:unionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="gb:Artifact"/>
            ...
            <rdf:Description rdf:about="gb:User"/>
        </owl:unionOf>
     ...
   <rdfs:range>
      <owl:Class>
         <owl:unionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="gb:Collection"/>
            ...
            <rdf:Description rdf:about="gb:User"/>
         ...
</owl:ObjectProperty>
```

Finally, an *owl:datatypeProperty* is created. Its type is 'xsd:string', due to the fact that the filler is a string (which will be the textual description of the motivation).

The strategy was implemented using the OWLAPI library [53] (a free Java-based API for creating and managing OWL-based information), since it is fully compliant with the OWL specifications. By means of this API, we can check both the consistency of the generated ontology as well as its expressive level. If the ontology is not consistent or is OWL Full, a warning is raised by GraphBRAIN and its use is frozen. In such a case, the ontology can be opened in the well-known OWL ontology editor Protégé [54] to check what caused the problem and act in GraphBRAIN to solve it.

An ontology is inconsistent if there are instances of unsatisfiable classes or if there are instances of two or more classes that are declared as disjoint. GraphBRAIN checks for inconsistencies in two different phases: when the ontology is generated (satisfiability) and each time an RDF graph is extracted from the LPG graph (consistency).

On the other hand, OWL Full uses all primitives of the OWL languages (https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/ (accessed on 1 September 2023)) and allows arbitrary combinations of these primitives with RDF and RDF schemas. Although OWL Full is fully compatible with RDF, both syntactically and semantically, it is undecidable (i.e., it does not have full (or efficient) support for reasoning). For the purposes of this paper, it is not a problem if the ontology is OWL Full, because SW reasoning is not currently used in the applications presented, but rather, SPARQL is used as a query language for RDF. But, in general, the proposed strategy ensures that the mapped KB is decidable for SW reasoning purposes.

4.2. Alignment with Existing Semantic Web Resources

After discussing how to map ontologies from GBS to OWL, we now describe how to map instances in the LPG graph to the RDF. One of the main issues in merging external resources with local DBs is that one cannot know that their ontology definition of a

concept corresponds to another available on the Web. More specifically, to the best of our knowledge, there is no solution in the literature for automatically aligning such vast and (possibly) different ontologies. So, we propose a semi-automatic mapping in which the user must specify how GBS concepts are to be mapped onto those available in the SW. We propose the use of RML by differentiating the type of values of some parameters. In fact, RML is used to map various languages for relational DBs but not graph DBs. For this purpose, we use the *rml:logicalSource* directive by specifying a Cypher query to be applied to all nodes of a certain type as *rml:referenceFormulation* formally (using turtle format):

rml:logicalSource
[*rml:referenceFormulation* `ql:CypherPattern`; *rml:iterator* `'label name in GraphDB'`];

Next, we apply R2RML directives (*rr:subjectMap and rr:predicateObjectMap)* to formalize the mapping with subject, predicate, and object of the ontology with which we want to map the DB graph entities found by the query execution.

Let us introduce a small example that is useful for understanding how each single element is mapped. Consider an LPG-based KG about movies (node label `:Movie`) and people (node label `:Person`), where movies have a title (property `title`) and a release date (property `release`), and persons have a name (property `name`) and a birth date (property `born`). Both kinds of nodes have a numeric code (property `id`) to uniquely identify their instances. There is a relationship between people and the movies they have acted in (type `:actedIn`).

Assuming that we want to map the LPG fragment in Listing 3 to the RDF using the classes and properties from the schema.org (https://schema.org/, accessed on 9 September 2023) vocabulary. Listing 4 shows the desired result. To obtain this mapping, we must specify that LPG attribute 'name' is mapped onto 'schema:name', 'born' onto 'schema:birthDate', and so on.

Listing 3. Example of simple graph data.

```
(:Person{id:1, name:"Al Pacino", born:date("1940-04-25")})
(:Person{id:2, name:"Robert De Niro", born:date("1943-08-17")})

(:Movie{id:1, title:"The Godfather Part II", release:date("1975-06-20")})

(:Person{id:1})-[:actedIn]->(:Movie{id:1})
(:Person{id:2})-[:actedIn]->(:Movie{id:1})
```

Listing 4. Example of mapping based on an extension of RML.

```
@prefix person: <http://loc.example.com/person/>.
@prefix movie: <http://loc.example.com/movie/>.

person:1 schema:name "Al Pacino"; schema:birthDate "1940-04-25"^^xsd:date.
person:2 schema:name "Robert De Niro"; schema:birthDate "1943-08-17"^^xsd:date.

movie:1
  schema:name "The Godfather Part II";
  schema:datePublished "1975-06-20"^^xsd:date;
  schema:actor person:1, person:2.
```

We assume that the class and property names are unique. For the movie example, we specify the 'PersonMapping', in which all elements related to the label 'Person' (in the LPG graph) must be specified. In this section, we specify how to map all the attributes of nodes of type 'Person' ('name' and 'born', among the others). After doing the same for the other class in the example ('Movie'), we may specify how to map the relationship 'actedIn'. We understand that it has the same meaning as the standard object property 'schema:actor'. In the mapping, we obviously also specify the mappings for the subject and the object. The full code of the mapping is shown in Listing 5. Let us briefly analyze it line by line.

R2RML and RML mappings consist of a set of 'triple maps', each representing a source of triples for the target RDF graph. Listing 5 defines 3 triple maps: `<#PersonMapping>` (lines 8–20), `<#MovieMapping>` (lines 22–33), and `<#ActedInMapping>` (lines 35–46). As per the RML specification, each triple map is based on a different *logical source* which, in our case, is a Cypher pattern, i.e., an expression that can be used in the `MATCH` clause of a Cypher query. `<#PersonMapping>` maps any LPG node with the label `:Person` to an IRI of the type `schema:Person` built from the property `id` with a template mechanism. Its LPG properties `name` and `born` are mapped to the RDF properties `schema:name` and `schema:birthDate`, respectively. Similarly, `<#MovieMapping>` maps LPG nodes with the label `:Movie` onto IRIs of the type `schema:Movie` and their LPG properties onto the corresponding RDF properties. Finally, `<#ActedInMapping>` maps LPG arcs labeled `:actedIn` to triples connecting the two IRIs corresponding to the movie and the actor with the property `schema:actor` (note that the RDF property has the opposite direction with respect to the `:actedIn` LPG relationship).

Listing 5. Example of RML-based mapping.

```
1  @prefix rr: <http://www.w3.org/ns/r2rml#>.
2  @prefix rml: <http://semweb.mmlab.be/ns/rml#>.
3  @prefix ql: <http://semweb.mmlab.be/ns/ql#>.
4  @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
5  @prefix schema: <http://schema.org/>.
6  @base <http://example.com/ns#>.
7
8  <#PersonMapping> a rr:TriplesMap;
9    rml:logicalSource [rml:referenceFormulation ql:CypherPattern; rml:iterator "(p:
         Person)"];
10
11   rr:subjectMap
12     [rr:template "http://loc.example.com/person/{p.id}"; rr:class schema:Person];
13
14   rr:predicateObjectMap [
15     rr:predicate schema:name;
16     rr:objectMap [rml:reference "p.name"; rr:datatype xsd:string]
17   ], [
18     rr:predicate schema:birthDate;
19     rr:objectMap [rml:reference "p.born"; rr:datatype xsd:date]
20   ].
21
22 <#MovieMapping> a rr:TriplesMap;
23   rml:logicalSource [rml:referenceFormulation ql:CypherPattern; rml:iterator "(m:Movie
         )"];
24
25   rr:subjectMap [rr:template "http://loc.example.com/movie/{m.id}"; rr:class schema:
         Movie];
26
27   rr:predicateObjectMap [
28     rr:predicate schema:name;
29     rr:objectMap [rml:reference "m.title"; rr:datatype xsd:string]
30   ], [
31     rr:predicate schema:datePublished;
32     rr:objectMap [rml:reference "m.released"; rr:datatype xsd:date]
33   ].
34
35 <#ActedInMapping> a rr:TriplesMap;
36   rml:logicalSource
37     [rml:referenceFormulation ql:CypherPattern; rml:iterator "(p)-[a:actedIn]->(m)"];
38
39   rr:subjectMap [rml:reference "m"; rr:parentTriplesMap <#MovieMapping>];
40
41   rr:predicateObjectMap [
42     rr:predicate schema:actor;
43     rr:objectMap [rml:reference "p"; rr:parentTriplesMap <#PersonMapping>]
44   ].
```

4.3. SPARQL Mapping

Publishing a GraphBRAIN KG in the SW by entirely exporting it would require the translation to be run again after each modification of the KG. An approach that is more suitable for dynamic KGs would be to expose the KG as an online SPARQL endpoint, from which data can be fetched at need by any SW-compliant application. Of course, the mapping technicalities between the different underlying representations should be transparent to the users. In the following text, we identify the resources by generic property

values, as is done in traditional DB settings, but not necessarily by URIs. In the following text, we refer again to the movie KG and to the mapping in Listing 5 and to the excerpt of a possible movie dataset shown in Figure 3. Two actors, 'Al Pacino' and 'Robert De Niro', both acted in 'The Godfather Part II', but 'Al Pacino' also acted in 'The Godfather', whose sequel is 'The Godfather Part II'. The two actors are labeled as 'Person', and the two films as 'Movie'. Note that we defined an equivalent concept for the relationship 'actedIn', but not for 'sequel' (see Listing 5).

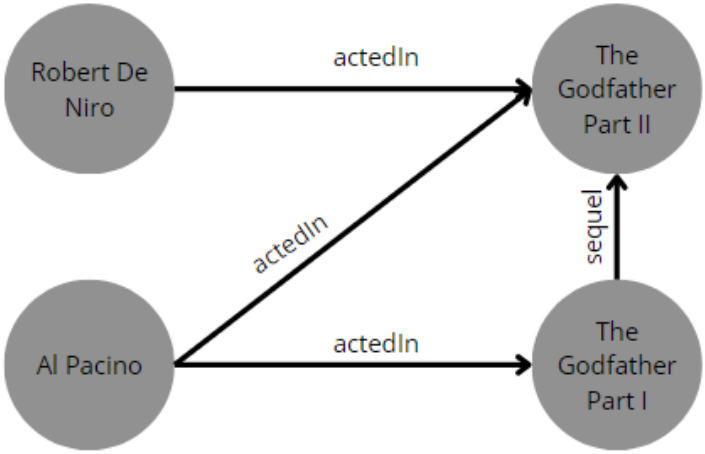

Figure 3. Excerpt of KG about actors starring in films.

Since one of the possible users of our SPARQL endpoint is GraphBRAIN's SKATEBOARD interface for KG browsing and exploration (see Section 5), here, we focus on three requests representing some of the most common SKATEBOARD queries.

(1) Return the type of node having a property.

This corresponds to the following SPARQL query:

```
SELECT DISTINCT ?object ?objectclass WHERE
{?object [property] [value].
?object rdf:type ?objectclass}
```

Our parsing strategy recognizes the query type thanks to the keyword **rdf:type**, and then, using simple pattern matching of specific tags in the query, it extracts the property [property], and the value [value], e.g., suppose the query asks for the type of node whose name (http://schema.org/name) matches the string 'The Godfather'. We check in the mapping file whether 'http://schema.org/name' corresponds to some property in the graph, and indeed, it has two correspondences: 'p.name' in PersonMapping, and 'm.title' in MovieMapping. PersonMapping has as a logical source 'Person', while MovieMapping has as a logical source 'Movie'. So, we check the graph for all 'Person' nodes with the value 'The Godfather' as the property 'name' and all 'Movie' nodes with the value 'The Godfather' as the property 'title' . To do this, when the parser recognizes that we are looking for the type of resource given a property, we prepare a general parametric Cypher query to retrieve this information from the graph. In our case, we execute this query, which is the result of the interpretation of the SPARQL query after the mapping process:

```
MATCH (n:Movie {title:'The Godfather'}) return labels(n)
UNION ALL
MATCH (n:Person {name:'The Godfather'}) return labels(n)
```

As expected, the result is the film named 'The Godfather'. Clearly, we do not return the type extracted from the graph, but obtain its reverse mapping according to Listing 5. In the graph shown in Figure 3, the label of 'The Godfather' is 'Movie', but by reversing the mapping procedure, we return 'http://schema.org/Movie'. This allows SW-based applications to infer new knowledge about the specific instance provided by the graph DB.

(2) Return all links directly connected to a given node.

This corresponds to the following SPARQL query:

```
SELECT DISTINCT ?link WHERE {
{
?object [property] [value] .
?object ?link ?outObject .
}
UNION
{
?inObject ?link ?object .
?object [property] [value] .
}
}
```

As shown in the previous scenario, we used a property value to identify the starting node. Since we do not care about the direction of the object property, the nodes we are looking for can be connected by an incoming or an outgoing arc to the given node, and we must return the union of these two cases. In the former, the property is associated with the object—in the latter to the object of the relationship.

Our approach to handling this case separates the two pieces of the query when a **UNION** is encountered and then applies the same procedure as for the first scenario, but this time by considering any object property, not only the **rdf:type**. The output consists of pairs of object properties and the corresponding subjects (for incoming arcs) or objects (for outgoing arcs). Finally, since the **SELECT DISTINCT ?link** specifies that only the object property must be returned and it must not be repeated, the output will be a list of distinct relationships to which the starting node is connected, regardless of whether it plays the role of the subject or of the object. Again, the properties are mapped back according to Listing 5. However, it may happen that the mapping document does not provide a mapping for a specific resource, i.e., the designer has not specified an external class (resp. object property) that can be considered to be equivalent to a class (resp. relationship) in the GraphBRAIN ontology. In these cases, we assign the URI of GraphBRAIN, i.e., `http://graphbrain.it/`*name* (where *name* is the name of the class or relationship) to these resources. So, we may return mixed resources, connecting standard ontological concepts with those defined by our schemes.

For example, supposing the property is again 'http://schema.org/name' and the value is 'The Godfather', Figure 3 shows that the film is reachable, not only through the 'actedIn' relationship, but also through 'sequel', for which no mapping has been provided. After parsing the query, again, 'http://schema.org/name' is ambiguous, so the query is decomposed as the union of many smaller queries. After the mapping, the equivalent Cypher query is

```
MATCH (n:Movie {title:'The Godfather'})-[r]-(m) return type(r)
UNION ALL
MATCH (n:Person {name:'The Godfather'})-[r]-(m) return type(r)
```

It returns two relationships from the graph: 'sequel' and 'actedIn'. The latter will be mapped, but the former will not, yielding the final result 'http://graphbrain.it/sequel', 'http://schema.org/actor'.

(3) Export a piece of the KG

This is the most general scenario, representing an example of how to enrich the SW knowledge by exporting data (pieces of KG) from GraphBRAIN. It is associated with the query:

```
MATCH (n:[Subj])-[r:[rel]]->(m:[Obj]) RETURN
labels(n), properties(n), type(r), labels(m), properties(m)
```

Here, we extract all triples consisting of a node of type [Subj] that is related to a node of type [Obj] through a [rel] arc from the graph. Suppose [Subj] is 'Person', [rel] is 'actedIn', and [Obj] is 'Movie'. This involves three elements of the mapping in Listing 5:

PersonMapping, MovieMapping, and ActedInMapping. The former indicates the class in the SW that is equivalent to 'Person' and also maps two of its properties ('born' and 'name'). The second determines the concept equivalent to class 'Movie' and also maps its property 'title'. Finally, the latter specifies how to map the relationship. ActedInMapping has as a logical source '(p)-[a:actedIn]→(m)'. p and m have their own mappings as well: the former is related to the MovieMapping, the latter to PersonMapping. Hence, we know that the subject must be a 'Person' in the graph, and the object must be a 'Movie'. This completes the mapping of all extracted elements.

The OWL/RDF Knowledge Base is created with the OWL API [53]. The results of the mapping and export are given in Listing 6.

Listing 6. Example of a mapped KB from a triple.

```
Prefix(:=<#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(gb:=<http://www.gb.it#>)
Prefix(schema:=<http://schema.org/>)
Ontology(<>

Declaration(Class(<#Movie>))
Declaration(Class(<#schema:Person>))
Declaration(ObjectProperty(<#schema:actor>))
Declaration(DataProperty(<#schema:birthDate>))
Declaration(DataProperty(<#schema:datePublished>))
Declaration(DataProperty(<#schema:name>))
Declaration(NamedIndividual(<gb:AlPacino>))
Declaration(NamedIndividual(<gb:RobertDeNiro>))
Declaration(NamedIndividual(<gb:TheGodfather>))
Declaration(NamedIndividual(<gb:TheGodfather2>))

#   Named Individuals

ClassAssertion(<#schema:Person> <gb:AlPacino>)
ObjectPropertyAssertion(<#schema:actor> <gb:AlPacino> <gb:TheGodfather>)
ObjectPropertyAssertion(<#schema:actor> <gb:AlPacino> <gb:TheGodfather 2>)
DataPropertyAssertion(<#schema:birthDate> <gb:AlPacino> "25/04/1940")
DataPropertyAssertion(<#schema:name> <gb:AlPacino> "Al Pacino")

ClassAssertion(<#schema:Person> <gb:RobertDeNiro>)
ObjectPropertyAssertion(<#schema:actor> <gb:RobertDeNiro> <gb:TheGodfather2>)
DataPropertyAssertion(<#schema:birthDate> <gb:RobertDeNiro> "17/08/1943")
DataPropertyAssertion(<#schema:name> <gb:RobertDeNiro> "Robert De Niro")

ClassAssertion(<#schema:Movie> <gb:TheGodfather>)
DataPropertyAssertion(<#schema:datePublished> <gb:TheGodfather> "21/09/1972")
DataPropertyAssertion(<#schema:name> <gb:TheGodfather> "The Godfather")

ClassAssertion(<#schema:Movie> <gb:TheGodfather2>)
DataPropertyAssertion(<#schema:datePublished> <gb:TheGodfather2> "20/06/1975")
DataPropertyAssertion(<#schema:name> <gb:TheGodfather2> "The Godfather 2")
)
```

5. End-User Interfaces

While designed for integration into any third-party system, GraphBRAIN comes with two official Web-based interfaces. The former allows form-based knowledge manipulation, in particular, the basic CRUD operations on the KG, and knowledge analysis and mining on the overall graph using the MultiStrategy Reasoning, Graph Mining, and Network Analysis algorithms and tools. These two groups of functions are strictly interconnected, meaning that the former may provide starting points and directions to the latter, while the latter may identify specific knowledge items to be handled by the former. The other interface is more oriented toward interactive and dynamic knowledge exploration and browsing. It was detached from the former interface because it has a broader range of applications, and in particular, it can be profitably applied to standard SW knowledge as well. Both are based on the GraphBRAIN API.

5.1. Knowledge Querying, Analysis and Mining

This part of the interface was developed as a Web Application based on Java Server Faces (JSF) technology. This ensures an advanced and flexible solution for dynamic Web pages on the client side and powerful and interoperable information processing on the server side. A demonstration can be found at http://digitalmind.di.uniba.it:8088/GraphBRAIN/ (accessed on 1 September 2023). The interface is organized into several tabs (see Figure 4), each devoted to a specific function.

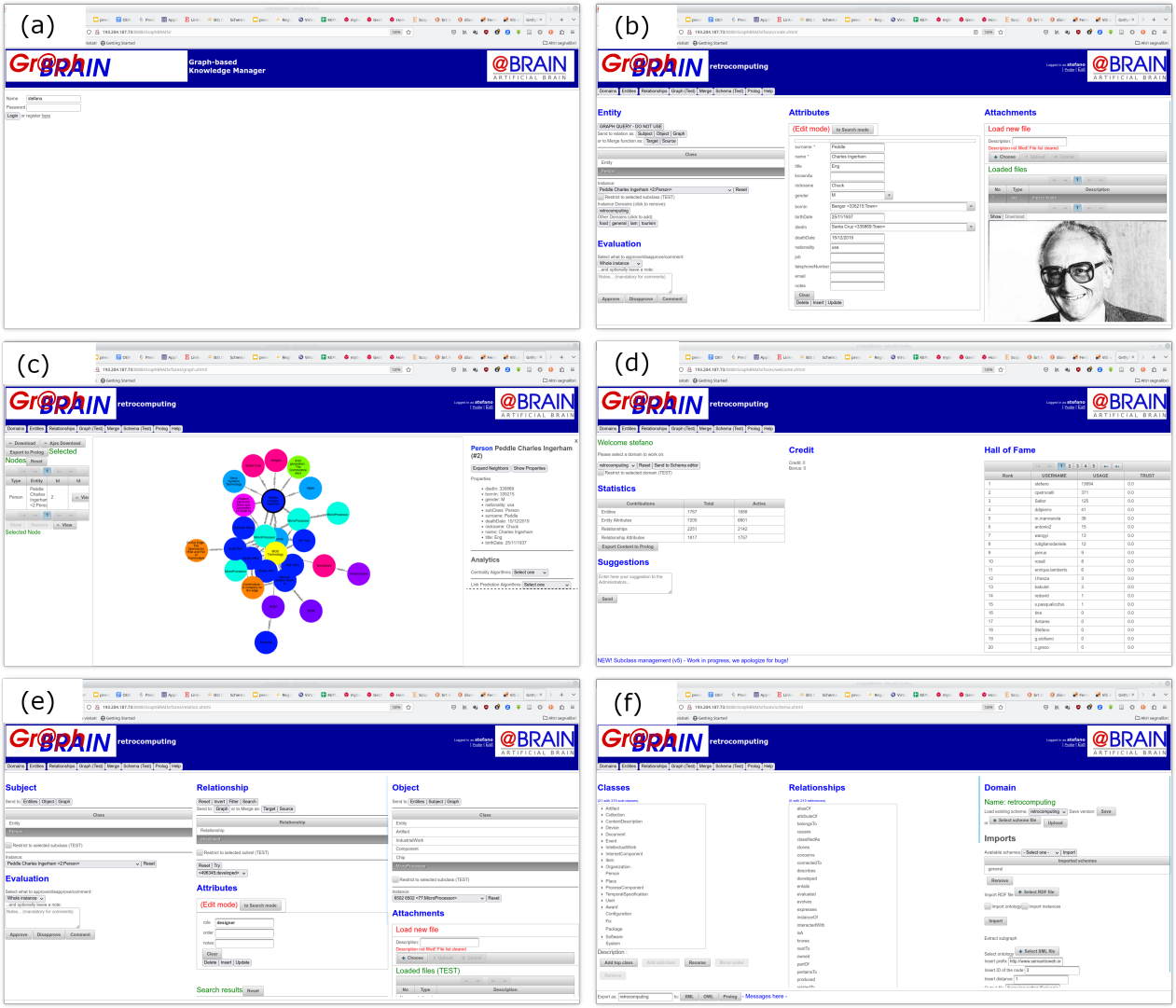

Figure 4. GraphBRAIN interface tabs— registration & login (**a**), domain selection & statistics (**d**); form-based entities (**b**) and relationships (**e**); graph browsing and network analysis/graph mining (**c**), ontology editing (**f**).

The **Domains** (Figure 4d) tab allows the user to select the domain to work on. As previously mentioned, while having a single KG underneath, GraphBRAIN can provide different partial 'views' on it based on different ontologies that act as schemas for the data in the graph. This tab also shows information about the domain organized into several tables: the amount of information available by type (entity or relationship instances and their attributes), user parameters (expressing his activities and processing budget), and contributors' data (quantity of contributions and trust). There is also a section to send comments to the administrators.

The **Entities** (Figure 4b) tab provides form-based access for searching, creating, updating, or deleting graph nodes (i.e., entity instances). On the left, the user can browse the

entity hierarchy of the currently selected domain and select any (sub)class. After a class has been selected, the corresponding attributes are determined from the ontology (using the inheritance for the subclasses) and shown in a form located in the center section of the interface. Also, a drop-down menu with all instances of the selected class (and, if requested, all of its subclasses) is shown on the left, so that the user can select any class and visualize its detailed information as attribute values that fill the form in the center. In the form, the user can enter or modify attribute values and use those values to update or add entity instances to the KG. They can also search entity instances compatible with those values, which will be shown in the instance drop-down menu on the left. On the left, the user can also see the domains associated with a selected instance as a set of buttons and act on these buttons to add or remove domains. The selected instance can also be sent to the other tabs for further processing (see next). On the right, facilities are provided to handle (upload or display) the attachments. On the bottom left, the user can approve, disapprove, or just comment on an instance as a whole or on a single attribute thereof: these contributions will be used by the system to determine the trust of the users who entered that information.

The **Relationships** (Figure 4e) tab is most similar to the Entities one, except that it refers to arcs (i.e., relationship instances) in the graph. So, it involves a subject class or instance, a relationship, and an object class or instance, which are handled, respectively, on the left, center, and right sections of the page. The subject and object sections act like the Entities tab: they allow the user to browse the class hierarchy and select a (sub)class for that role, after which the drop-down menu of instances is displayed and they can be selected. In the central section, the hierarchy of relationships provided by the currently selected domain can be browsed, and after selecting one, the corresponding attributes are determined (using inheritance for subrelationships) and shown in a form. Note that when selecting any partial combination of subject–relationship–object, the remaining components are automatically filtered to show only those provided by the ontology, e.g., if for the relationship 'wasIn', the ontology only provides the options Person.wasIn.Place and Organization.wasIn.Event, when selecting the relationship wasIn, the Subject section only displays Person and Organization, and the Object section only displays Place and Event; if further selecting Place for the Object, then the Subject section only displays Person as an option. Thanks to the LPG model adopted by GraphBRAIN, relationships may have attributes, and several instances of the same relationship can be established between the same Subject and Object instances, each with its own attributes. Controls are available to send the selected subject or object to the other tabs for further processing; work on the inverse relationship; filter only the items actually involved in that relationship in the graph in the Subject and Object instance menus; search for relationship instances based on attribute values entered by the user; and create, update, or delete relationship instances. Again, attachments can be handled in the right section, and feedback can be provided in the left section.

The **Graph** (Figure 4c) tab allows the user to display (a portion of) the KG, starting from selected nodes shown on the left (those that were sent to this tab from the Entities or Relationships tabs). The displayed portion is computed using Network Analysis algorithms (e.g., Spreading Activation or PageRank) that have been modified to ensure a more readable and pleasant result. It can also be computed based on the user profile, so that the retrieved information can be more interesting to the specific user; past user interactions with GraphBRAIN are exploited to build the user profile. If no starting nodes are selected, the starting nodes are automatically determined based on their relevance in the KG and, in turn, assessed by using various Network Analysis algorithms (e.g., centrality ones). The user can expand nodes to see all their neighbors, move the nodes to change the displayed graph topology, obtain detailed information on the nodes and arcs (i.e., their attribute values), and change the set of starting nodes by adding or deleting items. Analysis and mining functions are also provided, e.g., finding all (minimal) paths in the graph among a given set of nodes, determining the relevance of a node, clustering the nodes,

etc. Again, nodes and arcs can be sent to the Entities or Relationships tabs, respectively, for further processing.

The **Schema** (Figure 4f) tab allows the users to browse the ontology that determines the schema of the data in the KG and possibly to modify it and save a local copy. Modifications do not affect the schema on the server, because it would obviously break basic interoperability norms. Still, the users can use the modified ontologies for their GraphBRAIN-based KGs or propose the modifications to the GraphBRAIN administrators. In this section, the ontology currently selected in the Domains tab can be displayed or another one among those available on the server can be selected or the user can upload a local file in GBS format. On the left, the hierarchy of Entities/Classes can be browsed, also adding, removing, or renaming classes. After clicking on a class, its own attributes are shown below (i.e., inherited attributes are not shown when selecting a subclass) and can be modified by adding, removing, or changing the properties or the attribute (name, datatype, whether it is mandatory, etc.). Clicking on an attribute of type Select or Tree, its values are shown below. Similar controls are available for Relationships in the central section of the page. In this case, selecting a relationship also displays all possible Subject–Object pairs available for that relationship, and controls are available to add or remove pairs. The displayed relationships can be filtered based on specific Subject or Object classes. Additional controls allow the import of ontologies from the standard OWL SW format or the exportation of the currently displayed ontology and/or its instances in various formats:

- **GBS,** to be used for GraphBRAIN applications;
- **OWL,** to be used in standard SW settings;
- **Prolog,** to be used in standard Prolog programs (in this case, a domain-independent representation of the ontology and instances is obtained, expressed in terms of nodes, arcs, their attributes and values, and their labels);
- **GEAR,** to be fed to the MultiStrategy Reasoning engine, possibly along with axioms and background theories, to derive new knowledge (in this case, a domain-specific representation of the ontology and instances is obtained, in which the labels become predicate names and the attributes become atom arguments).

5.2. Knowledge Browsing and Exploration

As previously mentioned, the knowledge browsing and exploration tool for end users was developed separately from the knowledge querying, analysis, and mining tool. The first motivation is that it is aimed at supporting more dynamic interactions, while the other is more oriented toward supporting GraphBRAIN-based applications. Another reason is that, for interoperability purposes, this tool should be applicable also to standard SW resources, and the difference should be as transparent as possible to the final user. So, it was designed to work with resources in SW formats and has the case of GraphBRAIN sources as a special case, in which the GraphBRAIN KG is exposed as if it were a SPARQL resource.

SKATEBOARD, the Semantic Knowledge Advanced Tool for Extraction Browsing Organization Annotation Retrieval and Discovery, extends the capabilities of the ARCA system [55] for semantic searching over digital libraries, introducing several key enhancements. Its main features are

- **Collaborative Knowledge Creation and Updating**: This focuses on making the process of creating, modifying, and updating knowledge more collaborative. It enables users to actively participate in the development and refinement of the information within the digital library. This collaborative aspect is an important addition compared to ARCA.
- **Rapid Information Visualization**: This is designed for the rapid visualization of information directly related to the selected resource (see Figure 5). It excels at providing quick and intuitive access to relevant data, making it easier for users to extract insights from the digital catalog.
- **Geospatial and Semantic Visualization**: This offers features such as mapping locations, enriching character profiles with semantic information, and presenting books

with closely connected relationships. These visualizations enhance the understanding of the data and enable users to explore the content from different angles. The key functionalities of SKATEBOARD are listed below.

- **Combined Interaction Paradigms [56]**: This combines various interaction paradigms, including the node-link, tabular, and map-based interfaces. This versatility allows users to choose the visualization style that best suits their needs, providing a more immersive exploration experience.
- **Information Reduction Strategies**: This employs information reduction strategies based on the proximity ranking of relationships. This helps users to focus on the most relevant information, reducing information overload and improving the clarity of presented data.
- **Integration with Multiple SPARQL Endpoints**: This seamlessly integrates multiple SPARQL endpoints, enabling access to dynamic Knowledge Graphs from various sources. This capability enhances the scope and depth of the available information.
- **Collaborative Annotation and Validation**: This supports the collaborative annotation and validation of information present in the SPARQL endpoint. This feature promotes data quality and accuracy through user contributions and validations.

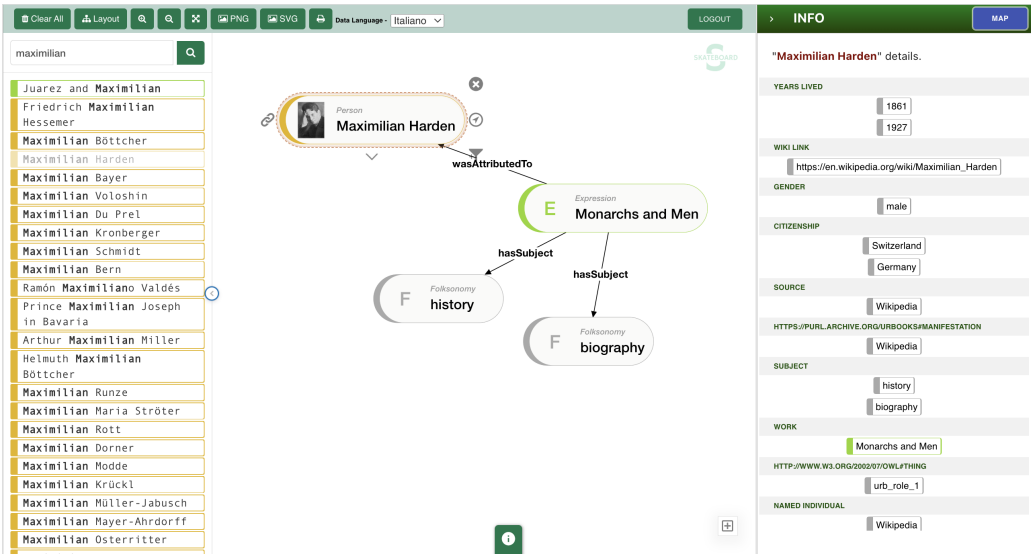

Figure 5. SKATEBOARD Interface.

The main purposes of SKATEBOARD are

- **Adaptation to Diverse Domains**: It aims to be adaptable to a wide range of domains, making it a versatile tool for exploring and understanding different types of digital content.
- **User-Friendly Interface**: It strives to provide an interface that is user-friendly, even for individuals who may not be experts in the specific domain of the digital library.

One of SKATEBOARD's main visualization features is to offer node-link-based visualization while also providing different views based on the selected entity type. Tools like Tabulator, Tabulator Redux, and S-Paths also offer various views depending on the entity type (https://linkeddata-89b9d.web.app/, accessed on 12 October 2023) [56], but SKATEBOARD distinguishes itself by allowing users to access custom views for different entity types without excluding the central node-link view.

Node-link diagrams have been employed to represent Linked Data for many years. They exhibit certain notable limitations, including poor scalability. The integration with GraphBrain enables SKATEBOARD to visualize specific segments of the LPG. These segments encompass the information pertaining to the selected node and its immediate connections, as defined by a specific parameter (the node and its context). This integration serves

as one solution to address the scalability issue associated with conventional node-link diagrams. Furthermore, this integration offers several advantages, including

1. Incremental Visualization: SKATEBOARD's capability to visualize only the pertinent portion of the LPG incrementally. It focuses on the selected node and its immediate context, reducing visual clutter and enhancing the user's ability to navigate data effectively.
2. Efficiency: The integration with labeled property graphs and the utilization of native LPG endpoints for querying, in conjunction with semantic mapping, provide the efficiency characteristic of LPG systems while retaining the inherent semantic connections in RDF data.

SKATEBOARD can be seen as a linker of SPARQL endpoints, which means that it can deal with resources expressed as RDF triples using a JSON-like language. The GraphBRAIN API function that can map some SPARQL queries into Cypher ones, keeping the same semantics, allows SKATEBOARD to be connected to GraphBRAIN KGs: GraphBRAIN is seen a SPARQL endpoint server which, when queried, is updated with a new piece of information that can be accessed via a public link by SKATEBOARD. A first consequence of this approach is that we can fully exploit both graph models to query data in a much more efficient way while visualizing them. Moreover, this makes the data available on the Web as RDF triples, in this way extending the set of interconnected data.

6. Evaluation

The evaluation of the GraphBRAIN framework and its accompanying graph exploration interface, SKATEBOARD, encompasses three main aspects: evaluation design, results, and practical implications. This comprehensive assessment aims to provide a holistic view of the capabilities, performance, and real-world applications of these systems.

6.1. Evaluation Design

The evaluation design was thoughtfully structured to ensure a rigorous and systematic assessment of GraphBRAIN and SKATEBOARD. It consisted of the following key components:

- Scope of evaluation;
- Data collection;
- Methodology;
- Participants;
- Metrics and criteria.

6.1.1. Scope of Evaluation

- **GraphBRAIN:** The evaluation focused on understanding the versatility and adaptability of GraphBRAIN as a collaborative knowledge graph framework. It assessed the depth and breadth of the knowledge representation, the domain adaptability, and the interconnectedness of ontologies. Additionally, it examined the diversity of applications supported by GraphBRAIN.
- **SKATEBOARD:** The evaluation of SKATEBOARD was centered around its performance and usability as a graph exploration interface. Specifically, it evaluated how SKATEBOARD performed within the context of the World Literature Knowledge Graph, considering factors such as query response times, system resource utilization, user satisfaction, ease of use, and result relevance.

6.1.2. Data Collection

- **GraphBRAIN:** Quantitative data for GraphBRAIN were collected by extracting statistics from the knowledge graph. These statistics included the numbers of entity instances, entity attributes, relationship instances, and relationship attributes. This quantitative information provided insights into the depth and breadth of knowledge representation.

- **SKATEBOARD:** The evaluation of SKATEBOARD involved a combination of quantitative and qualitative data collection methods. Quantitative data included metrics such as the query response times and system resource utilization, which were critical for assessing the performance. Qualitative data, obtained through user feedback, covered aspects such as the user satisfaction, ease of use, relevance of search results, and valuable suggestions for improvements.

6.1.3. Methodology

- **Analytic Hierarchy Process (AHP):** To ensure a structured and comparative evaluation, the Analytic Hierarchy Process (AHP) was employed. This systematic approach, as described by Vaidya and Kumar in their 2006 work [57], allowed for a methodical assessment of GraphBRAIN and SKATEBOARD against multiple criteria. The AHP facilitated a well-organized and data-driven evaluation process.

6.1.4. Participants

- **GraphBRAIN:** Domain experts, researchers, and developers with expertise in knowledge graph technologies actively participated in the evaluation of GraphBRAIN. Their deep understanding of the field contributed to a comprehensive analysis.
- **SKATEBOARD:** SKATEBOARD's evaluation involved a diverse group of 35 users, including researchers, students, and digital library practitioners. This user base provided valuable insights into the user experience and usability aspects of SKATEBOARD.

6.1.5. Metrics and Criteria

- **GraphBRAIN:** The evaluation metrics for GraphBRAIN encompassed knowledge graph statistics, domain adaptability, and ontology interconnections. These criteria enabled an assessment of the system's knowledge representation and flexibility.
- **SKATEBOARD:** The quantitative metrics for SKATEBOARD included the query response times, system efficiency, and overall performance. The qualitative criteria covered user satisfaction, ease of use, relevance of the search results, and suggestions for system enhancements.

6.2. Results

The evaluation results provide valuable insights into the performance and capabilities of GraphBRAIN and its associated graph exploration interface, SKATEBOARD. This section presents the findings of the evaluation, enriched with data and visual representations where applicable.

6.2.1. GraphBRAIN Evaluation Results

Knowledge Graph Statistics

The GraphBRAIN Knowledge Graph (KG) currently encompasses a wealth of entities and relationships, serving as a comprehensive repository of knowledge.

We evaluated it based on the number of knowledge items available in the current population of the KG underlying the demo prototype, available at http://digitalmind.di.uniba.it:8088/GraphBRAIN/ (accessed on 1 October 2023), as reported in Table 2, and on the variety of applications it supported, as described next. Note that Table 2 does not report figures for all domains described in this section, because some of them are still under investigation and are not yet uploaded in the prototype. On the other hand, most nodes in the KG are not labeled with any domain, representing precious background knowledge that is not part of any specific domain, but allows items that would otherwise be disconnected to be indirectly linked and inter-related across domains or even within single domains.

Table 2. Statistics on the content of the current GraphBRAIN prototype's KG.

Domain	Entity Inst.	Entity Attr.	Relationship Inst.	Relationship Attr.
Overall	337,287	2,089,580	496,839	41,594
Overall (domain)	2038	8069	2512	1958
General	102	573	222	132
LAM	63	294	93	69
Retrocomputing	1688	6801	2142	1757
Food	169	338	47	0
Tourism	14	56	8	0

Domain Adaptability

GraphBRAIN's adaptability across diverse domains is evident through its successful integration with domains such as Libraries/Archives/Museums (LAM), Retrocomputing, Food, and Tourism. These integrations enable a holistic representation of knowledge that extends beyond traditional domain-based boundaries.

The LAM (Libraries/Archives/Museums) domain was investigated in terms of using GraphBRAIN technology to overcome the limitations of traditional record-based approaches to Cultural Heritage descriptions. We proposed a 'holistic' approach aimed at representing all possible aspects of LAM: not only the formal metadata that are traditionally used to describe Cultural Heritage items, but also the *content* of cultural objects, their *physicity*, their *context*, and their *lifecycle*. Some of the data were populated using automated document processing techniques developed in previous projects.

Connected to the LAM domain is the Linguistics one, as it is based on the LAM sources. Here, the integration of semantic information into language resources showed the potential of the GraphBRAIN framework for research on semantic change in Latin.

Also, the Open Science domain relied on the LAM one and extended it to describe the context and environment in which scientific development takes place: processes and projects, datasets and corpora, scientific groups and communities, hardware and tools, software and storage facilities, etc.

The Retrocomputing domain (concerning the history of computing), in turn, relied on the LAM and Open Science domains. It is an extremely complex domain to represent. It involves documentation, hardware, software, and even, immaterial heritage. These components are inextricably interconnected, giving reason and explanation to each other. Also, the traditional description fields defined for other types of CH are unable to capture the complexity of the internal structure and the configurations of computing hardware.

The Food and Tourism domains, together with the CH proper sections of the KG, make up an ecosystem that is aimed at the enhancement and exploitation of the CH items by the final users. This again falls into our holistic perspective and provides a clear example of how it can open up new possibilities with respect to traditional, strictly domain-based representations.

These ontologies can be connected to each other via shared entities that act as bridges between the different domains and allow the reuse of knowledge across them. In addition to common entities among the above ontologies, most interconnections are due to the entities defined in a *general* top-level ontology, defined in GraphBRAIN independently of the various specific domains and including ubiquitous and highly reusable concepts such as Person, Organization, Event, Place, Collection, IntellectualWork, and Item.

Ontology Interconnections

GraphBRAIN facilitates seamless interconnections between ontologies. It achieves this by employing a top-level ontology that defines ubiquitous and reusable concepts, including Person, Organization, Event, Place, Collection, IntellectualWork, and Item. These shared entities act as bridges between different domains, promoting knowledge reuse.

6.2.2. SKATEBOARD Evaluation Results

Quantitative Evaluation

SKATEBOARD's quantitative evaluation revealed the following results:

- **Query Response Times:** On average, SKATEBOARD delivered query results in under five seconds, ensuring a responsive user experience.
- **Efficiency:** SKATEBOARD demonstrated efficient resource utilization, with low memory and CPU consumption, even during peak usage.

Qualitative Evaluation

Qualitative feedback from 35 users who engaged with SKATEBOARD within the World Literature Knowledge Graph highlighted the following aspects:

- **User Satisfaction:** Approximately 80% of the users expressed high levels of satisfaction with SKATEBOARD's interface and performance, rating their experience as "very satisfactory".
- **Ease of Use:** The users found SKATEBOARD intuitive and easy to navigate, even when exploring complex literary relationships and connections.
- **Relevance of the Results:** Feedback indicated the relevance and accuracy of the search results provided by SKATEBOARD, with 90% of users reporting that the system retrieved information that closely aligned with their queries.
- **Suggestions for Improvement:** Users offered valuable suggestions for enhancement, including expanded visualization options and additional contextual information.

6.3. Practical Implications

The evaluation results have significant practical implications for both GraphBRAIN and SKATEBOARD:

- **GraphBRAIN:** The rich knowledge representation and adaptability across domains make GraphBRAIN a valuable asset in interdisciplinary research. It can effectively support applications in fields such as Cultural Heritage, Linguistics, Open Science, Retrocomputing, Food, and Tourism. The seamless interconnections between ontologies enable knowledge reuse and cross-domain collaborations.
- **SKATEBOARD:** SKATEBOARD's efficient performance and high user satisfaction in the World Literature Knowledge Graph domain demonstrate its potential for diverse applications. Collaborations with digital libraries across various domains, including ancient manuscripts, comics, and archaeological works, can benefit from SKATEBOARD's usability and effectiveness.

In conclusion, the evaluation results affirm the capabilities and practical utility of GraphBRAIN and SKATEBOARD in supporting knowledge representation, exploration, and applications in interdisciplinary contexts.

7. Conclusions and Future Work

For complex or critical applications, knowledge-based AI systems must be used. Knowledge Graphs are currently enjoying widespread use in Semantic Web Applications, thanks to their expressive power and flexibility. This paper described GraphBRAIN, a new framework and platform for collaborative KG definition, population, and exploitation. Different from mainstream SW approaches, it is based on the LPG graph model, and adopts state-of-the-art graph DB solutions to ensure its efficiency and wide data manipulation facilities. While it can still be mapped onto standard SW representations, and thus enjoy their data repositories and reasoning tools, it was developed with the DB perspective in mind, and is open to additional functions coming from the Multistrategy Reasoning and Graph Mining branches of AI.

GraphBRAIN comes in the form of an API, which allows any Internet-based application to have access to its functionality. Given a DB and an ontology, the API wraps

the former and ensures that any access is compliant with the latter. Two Web Applications come with GraphBRAIN: one allows form-based CRUD and query operations on the knowledge (both ontology and instances), plus all of the analysis, mining, and reasoning tools embedded in the framework; the other is specifically aimed at interactive browsing and exploration by end users and ensures seamless compliance with the SW standards and repositories.

GraphBRAIN is currently adopted by several projects, many of which are in the Cultural Heritage field. This paper also reported an assessment of the framework and its interfaces in these domains, proposing it as a viable solution for semantic-based knowledge processing on the future Internet.

Author Contributions: Conceptualization, S.F., E.B., D.D.P. and D.R.; methodology, S.F., E.B., D.D.P. and D.R.; software, S.F., E.B., D.D.P. and D.R.; validation, S.F., E.B., D.D.P. and D.R.; formal analysis, S.F., E.B., D.D.P. and D.R.; investigation, S.F., E.B., D.D.P. and D.R.; resources, S.F., E.B., D.D.P. and D.R.; data curation, S.F., E.B., D.D.P. and D.R.; writing—original draft preparation, S.F., E.B., D.D.P. and D.R.; writing—review and editing, S.F., E.B., D.D.P. and D.R.; visualization, E.B.; supervision, S.F.; project administration, S.F.; funding acquisition, S.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by the projects FAIR—Future AI Research (PE00000013), spoke 6—Symbiotic AI, and CHANGES—Cultural Heritage Active innovation for Next-GEn Sustainable society (PE00000020), and Spoke 3—Digital Libraries, Archives and Philology, under the NRRP MUR program funded by the NextGenerationEU.

Data Availability Statement: The data in the prototype Web Application of GraphBRAIN can be explored at http://digitalmind.di.uniba.it:8088/GraphBRAIN/ (accessed on 1 September 2023).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
AHP	Analytic Hierarchy Process
DB	Data Base
KB	Knowledge Base
KG	Knowledge Graph
LPG	Labeled Property Graph
OWL	Ontology Web Language
RDF	Resource Description Framework
SW	Semantic Web
KRR	Knowledge Representation and Reasoning
KR	Knowledge Representation
URI	Uniform Resource Identifiers
DBMS	Data Base Management Systems
SHACL	Shapes Constraint Language
ShEx	Shape Expressions
RML	RDF Mapping Language
LOD	Linked Open Data
GBIF	Global Biodiversity Information Facility
NLP	Natural Language Processing
GBS	GraphBRAIN Schema

References

1. Robinson, I.; Webber, J.; Eifrem, E. *Graph Databases: New Opportunities for Connected Data*, 2nd ed.; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2015.
2. Ferilli, S. GEAR: A General Inference Engine for Automated MultiStrategy Reasoning. *Electronics* **2023**, *12*, 256. [CrossRef]
3. Ferilli, S. Integration Strategy and Tool between Formal Ontology and Graph Database Technology. *Electronics* **2021**, *10*, 2616. [CrossRef]

4. Di Pierro, D.; Ferilli, S. An API for Ontology-driven LPG Graph DB Management. In Proceedings of the 31st Symposium of Advanced Database Systems, Padua, Italy, 2–5 July 2023; pp. 303–316.
5. Krötzsch, M.; Thost, V. Ontologies for knowledge graphs: Breaking the rules. In *Proceedings of the Semantic Web—ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, 17–21 October 2016, Proceedings, Part I 15*; Springer: Cham, Switzerland, 2016; pp. 376–392.
6. Chiba, H.; Yamanaka, R.; Matsumoto, S. G2GML: Graph to Graph Mapping Language for Bridging RDF and Property Graphs. In *Proceedings of the Semantic Web—ISWC 2020—19th International Semantic Web Conference, Athens, Greece, 2–6 November 2020, Proceedings, Part II*; Pan, J.Z., Tamma, V.A.M., d'Amato, C., Janowicz, K., Fu, B., Polleres, A., Seneviratne, O., Kagal, L., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12507, pp. 160–175. [CrossRef]
7. Available online: https://protegeproject.github.io/owl2lpg (accessed on 9 September 2023).
8. Available online: https://github.com/SciGraph/SciGraph/wiki/Neo4jMapping (accessed on 9 September 2023).
9. Available online: https://github.com/VirtualFlyBrain/neo4j2owl (accessed on 9 September 2023).
10. Available online: https://github.com/cmungall/owlstar (accessed on 9 September 2023).
11. Hartig, O. Foundations to Query Labeled Property Graphs using SPARQL. In Proceedings of the 1st International Workshop on Semantics for Transport and the 1st International Workshop on Approaches for Making Data Interoperable Co-Located with 15th Semantics Conference (SEMANTiCS 2019), CEUR-WS.org, Karlsruhe, Germany, 9 September 2019; Volume 2447.
12. Available online: https://neo4j.com/blog/ontologies-in-neo4j-semantics-and-knowledge-graphs/ (accessed on 9 September 2023).
13. Angles, R.; Thakkar, H.; Tomaszuk, D. RDF and Property Graphs Interoperability: Status and Issues. In Proceedings of the 13th Alberto Mendelzon International Workshop on Foundations of Data Management, Asunción, Paraguay, 3–7 June 2019.
14. Saleem, M.; Khan, Y.; Hasnain, A.; Ermilov, I.; Ngonga Ngomo, A.C. A fine-grained evaluation of SPARQL endpoint federation systems. *Semant. Web* **2016**, *7*, 493–518. [CrossRef]
15. Bock, J.; Haase, P.; Ji, Q.; Volz, R. Benchmarking OWL reasoners. In Proceedings of the ARea2008—Workshop on Advancing Reasoning on the Web: Scalability and Commonsense, Tenerife, Spain, 2 June 2008.
16. Hoffart, J.; Yosef, M.; Bordino, I.; Fürstenau, H.; Pinkal, M.; Spaniol, M.; Taneva, B.; Thater, S.; Weikum, G. Robust disambiguation of named entities in text. In Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, Edinburgh, UK, 27–31 July 2011; pp. 782–792.
17. Sinaci, A.A.; Gonul, S. Semantic content management with apache stanbol. In *Proceedings of the Semantic Web: ESWC 2012 Satellite Events: ESWC 2012 Satellite Events, Heraklion, Crete, Greece, 27–31 May 2012. Revised Selected Papers 9*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 371–375.
18. Mendes, P.N.; Jakob, M.; García-Silva, A.; Bizer, C. DBpedia spotlight: Shedding light on the web of documents. In Proceedings of the 7th International Conference on Semantic Systems, Graz, Austria, 7–9 September 2011; pp. 1–8.
19. Butuc, M. Semantically Enriching Content Using OpenCalais. *Editia* **2009**, *9*, 77–88.
20. Gangemi, A. A comparison of knowledge extraction tools for the semantic web. In *ESWC 2013: The Semantic Web: Semantics and Big Data*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 351–366.
21. Nisheva-Pavlova, M.; Alexandrov, A. GLOBDEF: A framework for dynamic pipelines of semantic data enrichment tools. In *Proceedings of the Metadata and Semantic Research: 12th International Conference, MTSR 2018, Limassol, Cyprus, 23–26 October 2018, Revised Selected Papers 12*; Springer: Cham, Switzerland, 2019; pp. 159–168.
22. Available online: https://www.lerma.it/ (accessed on 9 September 2023).
23. Available online: https://www.torrossa.com/ (accessed on 9 September 2023).
24. Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; Ives, Z. Dbpedia: A nucleus for a web of open data. In Proceedings of the International Semantic Web Conference, Busan, Republic of Korea, 11–15 November 2007; Springer: Berlin/Heidelberg, Germany, 2007; pp. 722–735.
25. Virgilio, R.D. Smart RDF data storage in graph databases. In Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Madrid, Spain, 14–17 May 2017; pp. 872–881.
26. Tomaszuk, D. RDF data in property graph model. In Proceedings of the Research Conference on Metadata and Semantics Research, Gottingen, Germany, 22–25 November 2016; Springer: Cham, Switzerland, 2016; pp. 104–115.
27. Angles, R.; Thakkar, H.; Tomaszuk, D. Mapping rdf databases to property graph databases. *IEEE Access* **2020**, *8*, 86091–86110. [CrossRef]
28. Sahoo, S.S.; Halb, W.; Hellmann, S.; Idehen, K.; Thibodeau, T., Jr.; Auer, S.; Sequeda, J.; Ezzat, A. A survey of current approaches for mapping of relational databases to RDF. *W3C RDB2RDF Incubator Group Rep.* **2009**, *1*, 113–130.
29. Zhou, S. Exposing relational database as RDF. In Proceedings of the 2010 2nd International Conference on Industrial and Information Systems, Dalian, China, 10–11 July 2010; Volume 2, pp. 237–240.
30. Spanos, D.E.; Stavrou, P.; Mitrou, N. Bringing relational databases into the semantic web: A survey. *Semant. Web* **2012**, *3*, 169–209. [CrossRef]
31. Vavliakis, K.; Grollios, T.; Mitkas, P. RDOTE–Publishing Relational Databases into the Semantic Web. *J. Syst. Softw.* **2013**, *86*, 89–99. [CrossRef]
32. Bizer, C. D2R MAP—A Database to RDF Mapping Language. In Proceedings of the Web Conference, Budapest, Hungary, 20–24 May 2003.

33. Arenas, M.; Bertails, A.; Prud'hommeaux, E.; Sequeda, J. A Direct Mapping of Relational Data to RDF. Available online: http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/ (accessed on 19 October 2023).
34. Das, S.; Sundara, S.; Cyganiak, R. R2RML: RDB to RDF Mapping Language. Available online: http://www.w3.org/TR/2012/REC-r2rml-20120927/ (accessed on 19 October 2023).
35. Junior, A.C.; Debattista, J.; O'Sullivan, D. Assessing the Quality of R2RML Mappings. In Proceedings of the SEM4TRA-AMAR@ SEMANTiCS, Karlsruhe, Germany, 9 September 2019.
36. Zhao, Z.; Han, S.; Kim, J. R2LD: Schema-based Graph Mapping of relational databases to Linked Open Data for multimedia resources data. *Multimed. Tools Appl.* **2019**, *78*, 28835–28851. [CrossRef]
37. Matsumoto, S.; Yamanaka, R.; Chiba, H. Mapping RDF graphs to property graphs. *arXiv* **2018**, arXiv:1812.01801.
38. Dimou, A.; Vander Sande, M.; Colpaert, P.; Verborgh, R.; Mannens, E.; Van de Walle, R. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In Proceedings of the LDOW 2014, CEUR-WS.org, London, UK, 4–7 July 2014; Volume 1184.
39. Debruyne, C.; O'Sullivan, D. R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings. In Proceedings of the LDOW@ WWW, Montreal, QC, Canada, 12 April 2016; Volume 1593.
40. Rodriguez-Muro, M.; Rezk, M. Efficient SPARQL-to-SQL with R2RML mappings. *J. Web Semant.* **2015**, *33*, 141–169. [CrossRef]
41. Priyatna, F.; Corcho, O.; Sequeda, J. Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In Proceedings of the 23rd International Conference on World Wide Web, Seoul, Republic of Korea, 7–11 April 2014; pp. 479–490.
42. Heyvaert, P.; Chaves-Fraga, D.; Priyatna, F.; Corcho, O.; Mannens, E.; Verborgh, R.; Dimou, A. Conformance test cases for the RDF mapping language (RML). In *Proceedings of the Knowledge Graphs and Semantic Web: First Iberoamerican Conference, KGSWC 2019, Villa Clara, Cuba, 23–30 June 2019, Proceedings*; Springer: Cham, Switzerland, 2019; pp. 162–173.
43. Dimou, A.; Vander Sande, M.; Colpaert, P.; De Vocht, L.; Verborgh, R.; Mannens, E.; Van de Walle, R. Extraction and semantic annotation of workshop proceedings in HTML using RML. In *Proceedings of the Semantic Web Evaluation Challenge: SemWebEval 2014 at ESWC 2014, Anissaras, Crete, Greece, 25–29 May 2014, Revised Selected Papers*; Springer: Cham, Switzreland, 2014; pp. 114–119.
44. Penev, L.; Dimitrova, M.; Senderov, V.; Zhelezov, G.; Georgiev, T.; Stoev, P.; Simov, K. OpenBiodiv: A Knowledge Graph for Literature-Extracted Linked Open Data in Biodiversity Science. *Publications* **2019**, *7*, 38. [CrossRef]
45. Bauer, F.; Kaltenböck, M. *Linked Open Data: The Essentials*; Edition Mono/Monochrom: Vienna, Austria, 2011.
46. Purohit, S.; Van, N.; Chin, G. Semantic Property Graph for Scalable Knowledge Graph Analytics. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; pp. 2672–2677.
47. Vidal, M.E.; Castillo, S.; Acosta, M.; Montoya, G.; Palma, G. On the selection of SPARQL endpoints to efficiently execute federated SPARQL queries. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXV*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 109–149.
48. Saleem, M.; Potocki, A.; Soru, T.; Hartig, O.; Ngomo, A.C.N. CostFed: Cost-based query optimization for SPARQL endpoint federation. *Procedia Comput. Sci.* **2018**, *137*, 163–174. [CrossRef]
49. Zviedris, M.; Barzdins, G. ViziQuer: A tool to explore and query SPARQL endpoints. In *Proceedings of the Semanic Web: Research and Applications: 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, 29 May–2 June 2011, Proceedings, Part II 8*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 441–445.
50. Heibi, I.; Peroni, S.; Shotton, D. Enabling text search on SPARQL endpoints through OSCAR. *Data Sci.* **2019**, *2*, 205–227. [CrossRef]
51. Ferilli, S.; Redavid, D. The GraphBRAIN System for Knowledge Graph Management and Advanced Fruition. In *Proceedings of the Foundations of Intelligent Systems, Graz, Austria, 23–25 September 2020*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12117, pp. 308–317.
52. Pat Hayes, C.W. *Defining N-ary Relations on the Semantic Web*; W3c Working Group Note; World Wide Web Consortium: Wakefield, MA, USA, 2006.
53. Available online: https://github.com/owlcs/owlapi/wiki/Documentation (accessed on 9 September 2023).
54. Available online: https://protege.stanford.edu/ (accessed on 9 September 2023).
55. Bernasconi, E.; Ceriani, M.; Mecella, M.; Catarci, T. Design, realization, and user evaluation of the ARCA system for exploring a digital library. *Int. J. Digit. Libr.* **2023**, *24*, 1–22. [CrossRef] [PubMed]
56. Bernasconi, E.; Ceriani, M.; Pierro, D.D.; Ferilli, S.; Redavid, D. Linked Data Interfaces: A Survey. *Information* **2023**, *14*, 483. [CrossRef]
57. Vaidya, O.S.; Kumar, S. Analytic hierarchy process: An overview of applications. *Eur. J. Oper. Res.* **2006**, *169*, 1–29. [CrossRef]