Semantic Web 0 (0) 1
IOS Press

metaphactory: A Platform for Knowledge Graph Management

Peter Haase [a], Daniel M. Herzig [a], Artem Kozlov [a], Andriy Nikolov [a,*] and Johannes Trame [a]
[a] *metaphacts GmbH, Walldorf, Germany*
E-mails: ph@metaphacts.com, dh@metaphacts.com, ak@metaphacts.com, an@metaphacts.com, jt@metaphacts.com

Abstract. In this system paper we describe *metaphactory*, a platform for building knowledge graph management applications. The *metaphactory* platform aims at supporting different categories of knowledge graph users within the organization by realizing relevant services for knowledge graph data management tasks, providing a rich and customizable user interface, and enabling rapid building of use case-specific applications. The paper discusses how the platform architecture design built on open standards enables its reusability in various application domains and use cases as well as facilitates integration of the knowledge graph with other parts of the organizational data and software infrastructure. We highlight the capabilities of the platform by describing its usage in four different knowledge graph application domains and share the lessons learnt from the practical experience of building knowledge graph applications in the enterprise context.

Keywords: Knowledge graph, Data management, End-user platform, Industrial application

1. Introduction

In the recent years, knowledge graph technologies established a solid position in the enterprise world, serving as a central element in the organizational data management infrastructure. Knowledge graphs are becoming both the repository for organization-wide master data (ontological schema and static reference knowledge) as well as the integration hub for various legacy data sources: e.g., relational databases or data streams. However, the tool support for managing knowledge graphs and building custom applications on top of them is still limited. To support organizations in managing and making use of knowledge graphs, we created the *metaphactory* platform which covers the whole lifecycle of knowledge graph applications: from data extraction & integration, storage, and querying to visualization and data authoring.

In order to exploit the capabilities of knowledge graph technologies to the maximal extent, an organization requires many supporting functionalities beyond merely being able to store data as a graph and query it. Within a large organization, these functionalities have to provide support for different user groups: from lay users who merely want to explore the data in a convenient way to expert users who manage and modify the knowledge graphs and internal developers who create targeted applications customized for specific end user groups. Based on these diverse needs, such functionalities include:

- *Data management*: Common management tasks can be time-consuming and expensive without supporting tools. Even assistance in such basic functionalities like SPARQL querying with graphical UI, auto-suggestions, and query catalog can already go a long way in uncovering the added value of knowledge graphs. Moreover, support for knowledge graph authoring, including both schema ontologies and the instance data, as well as integration of data from other sources of different types are the features needed in almost any enterprise use case.

*Corresponding author. E-mail: an@metaphacts.com. The authors are listed in the alphabetical order.

– *End-user oriented interaction*: The user has to be able to interact with the knowledge graph in an intuitive and user-friendly way. This includes, for example, the need for navigation, exploration and visualization of knowledge graphs, rich semantic search with visual query construction and faceting, and a simple yet powerful interface for data authoring and editing.
– *Rapid application development*: Generic supporting toolkit must enable creating use case-specific user applications with minimal effort.

We developed the *metaphactory* platform to realize these functionalities while primarily aiming at large enterprises and organizations. The challenges of the use case scenarios in such organizations necessitate certain design principles which a generic knowledge graph management platform has to follow:

– *Reusability*: The platform can be used in a great variety of contexts, domains, and use cases to serve multiple categories of users. All aspects and functionalities of a generic platform must be customizable to enable it to be reused in any context. Whenever possible, the design must avoid any inherent assumptions about the data and user needs. The use of generic open standards such as RDF[1], SPARQL[2], SHACL[3], and others where available helps to achieve this.
– *Compatibility*: In most cases, the knowledge graph constitutes only a part of the organizational data infrastructure and must be used in combination with other elements: non-RDF data sources, data processing APIs, external data analytics tools. A knowledge graph management platform must be compatible and envisage integration with other elements of the infrastructure both at the input level (combining different kinds of data for transparent access) as well as at the output level (enabling knowledge graph data to be consumed by external tools). Open interfaces should be used whenever possible to simplify such integration.
– *Extensibility and customization*: A platform which is intended to be reused in different domains and use cases must provide an expert user with capabilities to build custom targeted applications for her needs with only a minimal support from the platform vendor.

In this paper we present the key concepts of the *metaphactory* platform showing how these design principles help to address the knowledge graph management challenges and share our experiences and lessons learnt in applying the platform in diverse real-world use cases. The rest of the article is structured as follows. Section 2 provides a high-level overview of the architecture of the platform and its main building blocks. Further sections explain the functionalities of each architecture layer in detail. Section 3 describes how the platform manages heterogeneous data sources and provides a uniform querying mechanism. Section 4 outlines the set of services realized by the platform and exposed for consuming. Section 5 outlines the design decisions behind the customizable UI of *metaphactory* and discusses different supported interaction paradigms. In section 6 we describe four application scenarios of the platform and discuss the experiences and lessons learnt. Finally, section 7 concludes the paper and provides the directions for future work.

2. Architecture

Figure 1 shows a high-level overview of the architecture of the *metaphactory* platform developed to provide a variety of functionalities aiming at different target user groups. One such group includes the *end users* within the organization. These users are not interested in the internals of the knowledge graph data structure or technical complexities of data access and integration. The *expert users* and data architects that have to author and modify the knowledge graphs and incorporate legacy data are primarily interested in convenient and efficient tools to make data management operations easier. Finally, *application developers* within the organization require that it is easy to incorporate the knowledge graph infrastructure into the overall software infrastructure of the organization and build new targeted end-user applications exploiting the knowledge graph with a minimal effort. The architecture was designed to reconcile the diverse requirements of these groups while maintaining the design constraints outlined above: enabling reusability in different domains, compatibility with other data sources and applications, and supporting extensibility and customization.

The platform operates on top of a *graph database* storing the knowledge graph. The communication is

[1] https://www.w3.org/RDF/
[2] https://www.w3.org/TR/sparql11-query/
[3] https://www.w3.org/TR/shacl/

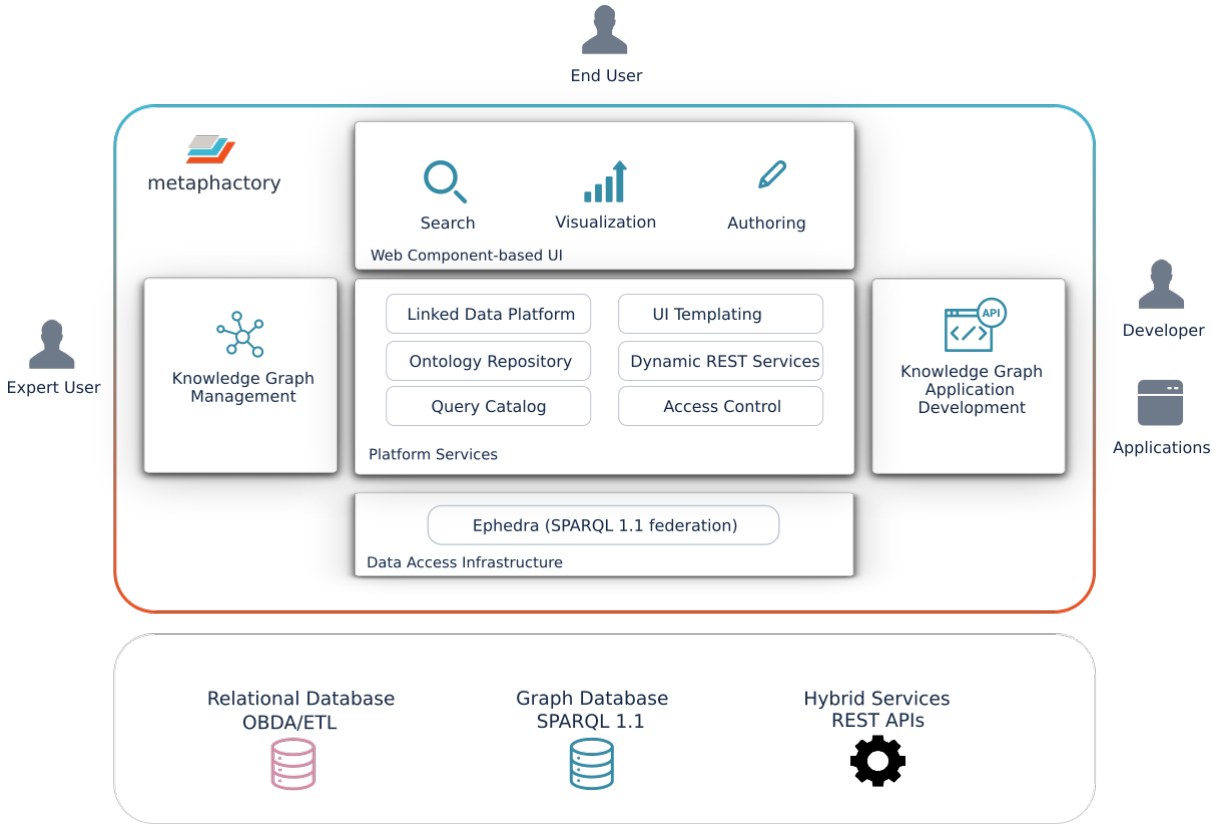

Fig. 1. *metaphactory* system architecture.

performed by the *data access infrastructure* of the platform using standard SPARQL 1.1 queries, which makes the system independent from a specific database vendor. Depending on the customer needs, this allows the platform to connect to various different triple stores or even non-RDF sources virtually integrated and exposed via a SPARQL interface: e.g., a relational database integrated using R2RML[4] mappings or a custom keyword search index. To be able to interact with multiple data sources using virtual data integration, the platform contains a SPARQL federation engine Ephedra [1], which realizes SPARQL 1.1 query federation over both SPARQL endpoints and custom compute services.

On top of the data access infrastructure, the *platform services* layer implemented at the platform backend side realizes a range of generic functionalities for interaction with knowledge graphs. The services extend the capabilities of the standard SPARQL access normally provided by triple stores and offer the specific capabilities to serve the needs of each target user groups: simplify the communication between web-based end-user UI and the knowledge graph, perform knowledge graph management operations over the ontological schema and data required by expert users and knowledge graph maintainers, and enable easy interaction with other tools, which is needed by in-house development teams. These services are exposed as REST APIs to be consumed by the client-side user interface as well as by third-party tools: e.g., Tableau[5] or KNIME[6] for data analytics. One critical functionality implemented at this level is the user access control: the platform allows configuring fine-grained access at the level of specific APIs available to each user role.

The *web-component based user interface* is built using a customizable templating mechanism. Each URI resource in the knowledge graph is visualized using an HTML page. While one can design a separate HTML page for each resource, in the majority of cases this is

[4]https://www.w3.org/TR/r2rml/

[5]http://www.tableau.com
[6]http://www.knime.com

not required: it is possible to specify *template pages* that can be applied to any resource of the same type. When visualizing a data resource, the platform tries to select an appropriate template among the available ones. This template then gets rendered using the actual URI of the resource. The *metaphactory* client-side UI provides a plethora of customizable UI components for interacting with semantic data: from table-based and graph-based visualizations to structured search environments and data authoring controls based on knowledge patterns. Each one represents an HTML5 component that can be directly inserted and configured within HTML code.

In this way the user interface can be easily customized for the specific application use case at hand. The use of the standard HTML5 format for storing client-side UI views enables an expert user to create and edit her own interface. Use case-specific configuration parameters and UI templates can be packaged as a separate app and added to the default platform installation to realize a custom domain-specific knowledge graph management application.

At the core of the architecture is the use of open standards at all levels to enable smooth integration of the platform with existing tools and data providers. This also allowed using open-source libraries developed in the Semantic Web community to implement the platform functionalities. Table 1 summarizes the open standards and tools used for implementation of the *metaphactory* platform at different architecture layers.

3. Data Access Infrastructure

The *metaphactory* data access infrastructure is built on top of the RDF repository that serves as the storage for the managed knowledge graph data. The SPARQL 1.1 query language supported by most available RDF databases serves as a common communication protocol. The platform reuses a popular RDF4J framework[7] to realize access to data repositories. In this way, (a) the platform can be installed in any environment that already includes a triple store chosen by the customer organization as well as (b) enables selecting any data storage solution based only on the use case requirements without introducing additional constraints on its own. In particular, this helps addressing the scalability issue: an appropriate backend triple store and a suitable hardware infrastructure for it can be selected depending on the size of the dataset. The platform can be installed separately from the triple store and interact via HTTP.

The *metaphactory* platform officially supports most of the well-known triple store solutions available on the market, e.g. Blazegraph[8], Stardog[9], Amazon Neptune[10], GraphDB[11], Virtuoso[12], and others.

While the platform requires the existence of one main default RDF repository containing domain data, it allows working with multiple repositories as well: the platform's *repository manager* enables configuring and managing connections to many data repositories in a declarative way. In particular, this enables managing domain data separately from the system data such as saved SPARQL queries or SHACL data quality reports. Different data sources maintained by the repository manager are described in RDF using the RDF4J repository configuration ontology. These repositories can include not only native RDF triple stores but other data sources accessible via SPARQL: most importantly, relational databases virtually integrated using R2RML mappings and exposed via SPARQL with an ontology-based data access engine, either a separate one like Ontop [2] or one integrated with a triple store like Stardog.

In many use case scenarios there arises a need to handle *hybrid information needs* that require combining information from multiple data sources. These needs are characterized by such dimensions as:

- *Variety of data sources*: The data to be integrated is often stored in several physical repositories. Such repositories can include both RDF triple stores and datasets that are only virtually presented as RDF: e.g., relational databases exposed using R2RML mappings.
- *Variety of data modalities*: RDF graph data often needs to be combined with other data modalities: e.g., textual, temporal, or geospatial data. To be able to integrate those, SPARQL queries need to support special extensions for full-text, spatial, and other corresponding types of search.

[7] http://rdf4j.org/

[8] https://www.blazegraph.com/

[9] https://www.stardog.com/

[10] https://aws.amazon.com/neptune/

[11] https://ontotext.com/products/graphdb/

[12] https://virtuoso.openlinksw.com/

Table 1
Open standards and open-source semantic web tools utilized in the *metaphactory* platform.

Architecture layer	Standards used	Open-source semantic web tools
Data access infrastructure	RDF, SPARQL 1.1, GeoSPARQL, R2RML	RDF4J, Ontop
Platform services	LDP, SHACL, RDFS/OWL, REST	RDFUnit
User interface	HTML5, Web Components	SPARQL.js

– *Variety of data processing techniques*: Relevant data is often not stored directly in some repository, but has to be computed by some dedicated domain-specific services: e.g., graph analytics (finding the shortest path or interconnected graph cliques), statistical analysis and machine learning (applying a machine learning classifier, finding similar entities using a vector space model), etc.

An application scenario can require dealing with several of these aspects simultaneously. While federated query processing appears a natural way for on-the-fly integration of diverse data sources, the research effort, however, mainly concentrated on achieving the transparent query federation over native RDF datasets as opposed to the hybrid query challenges. Existing approaches either utilize meta-level information about federation members in order to build an optimal query plan (e.g., DARQ [3], SPLENDID [4], ANAPSID [5], or HiBISCuS [6]) or use special runtime execution techniques targeting remote service queries (e.g., FedX [7]). Among the approaches targeting hybrid query processing, SCRY [8] and Quetzal-RDF [9] deal with calling data processing services using SPARQL queries. Quetzal-RDF defines custom functions and table functions (generalized aggregation operations) and invokes them from a SPARQL query, but does not follow the SPARQL 1.1 syntax. SCRY conforms to SPARQL 1.1 using special GRAPH targets to wrap service invocations, although it cannot distinguish between multiple input/output parameters. Thus, both these solutions are not generic enough to handle the whole range of available hybrid data sources that include arbitrary structure of input and arbitrary size of the solution sets.

Given the limitations of existing solutions, to address these challenges we implemented Ephedra: a SPARQL federation engine for hybrid queries [1]. We adopt the SPARQL 1.1 federation mechanism using the SERVICE keyword, but broaden its usage to enable custom services to be integrated as data sources and optimize such hybrid SPARQL queries to be executed efficiently.

Ephedra defines a common implementation interface, in which interactions with external services are encapsulated in an RDF4J SAIL module[13]. In this way, a custom compute service can be registered in the repository manager as yet another SPARQL repository and referenced inside SERVICE clauses in SPARQL queries. SPARQL graph patterns specified inside such SERVICE clauses are parsed to extract input parameters for a service call as well as the variables to bind the results returned by the service. The Ephedra SPARQL query execution strategy sends the sub-clauses of a query to corresponding data sources, gathers partial results, combines them using union and join operations, and produces result sets. In this way, processing hybrid queries is transparent and performed in the same way as ordinary SPARQL queries.

In order to express custom service requests as part of SPARQL queries, hybrid services integrated through Ephedra are declaratively described using the Ephedra *service descriptor ontology*. This ontology extends the well-known SPIN[14] ontology to define the accepted graph patterns, input arguments, and output results. For example, a wrapper for a custom service that retrieves similar entities based on the proximity in the word2vec vector embedding space looks like the following:

```
:Word2VecSimilarityService a eph:Service ;
  rdfs:label "A wrapper for the word2vec
  similarity service." ;
  eph:hasSPARQLPattern (
    [
      sp:subject :_entity ;
      sp:predicate mph:hasSimilar ;
      sp:object :_similar
    ]
  ) ;
  spin:constraint
  [
    a spl:Argument ;
    rdfs:comment "URI of the search entity" ;
    spl:predicate :_entity ;
    spl:valueType xsd:anyURI
  ] ;
  spin:column
  [
    a spin:Column ;
    rdfs:comment "URI of a similar entity" ;
    spl:predicate :_similar ;
    spl:valueType xsd:anyURI
  ] .
```

[13] http://docs.rdf4j.org/sail/
[14] http://spinrdf.org/

Based on this descriptor, Ephedra will be able to interpret and process the following query answering the question "Which painters are similar to Rembrandt?" and returning service outputs among the query result:

```
SELECT ?artist ?label WHERE {
  SERVICE mpfed:wikidataWord2Vec {
     wd:Q5598 mph:hasSimilar ?artist .
  }
  ?artist wdt:P106 wd:Q1028181 . # painter
  ?artist rdfs:label ?label .
}
```

With this approach, services with standardized interfaces (e.g., REST APIs) can be included into the Ephedra federation in a fully declarative way, without the need to implement specific service wrappers.

Processing hybrid queries including custom service calls requires modifying the query processing procedures, because a hybrid query does not follow the standard SPARQL semantics. Processing a SERVICE clause realizing a custom service call requires all input parameters to be bound, which means that the join operands cannot be arbitrarily re-ordered. For this reason, Ephedra implements dedicated static query optimization strategies, which produce an optimal and executable query plan. The SPARQL query algebra is extended to include *service call patterns* Σ^S as first-class elements. Building a query plan containing hybrid service call patterns involves join order optimization to ensure that such patterns can be only joined after their input dependencies are bound. Moreover, presence of a service call pattern in the query plan imposes additional restrictions on the selection of suitable join operators: for example, it necessitates the use of nested bound join as opposed to hash join.

After constructing an executable hybrid query plan, Ephedra uses dynamic query optimization techniques to reduce the processing time: in particular, *synchronizing loop join requests* and *adaptive processing of n-ary joins*. The evaluation experiments reported in [1] show that these techniques result in runtime improvements for all test queries, sometimes by an order of magnitude.

4. Platform Services

The platform backend realizes a range of services that implement additional functionalities on top of the standard interfaces of triple stores. These services streamline the specific types of interactions with the knowledge graph that are required by different target user groups. They include (a) convenience services that realize commonly required tasks that are usually not provided by triple stores directly and (b) connector services that make data from the knowledge graph consumable by applications.

Convenience services implement commonly required routines that are either too cumbersome to be realized by sending SPARQL queries directly or require additional tools (e.g., a separate keyword search index). These services are implemented in a generic way avoiding dependencies on a particular triple store and/or ontology. A simple example of a convenience service is a generic label service retrieving a human-readable label for a given resource. The service can be configured to deal with various modelling patterns (e.g., taking into account not *rdfs:label*, but *skos:prefLabel* or even property paths) and language preferences. Such services provide the functionalities utilized by the end-user interface components.

Another set of generic convenience services realizes the knowledge graph management tasks required by expert users and their tools:

- Linked Data Platform (LDP) service for resource-based access, creation, update, and deletion of linked data according to the W3C LDP specification[15]
- Data quality service for performing data validation using SHACL constraints.
- Query catalog service for saving and managing reusable SPARQL query templates (expressed using the SPIN ontology).
- Keyword search service based on the GraphScope[16] data search engine integrated into the platform.

Manual modifications of a knowledge graph as well as bringing in transformed legacy data sources cause the need to verify and maintain the knowledge graph data quality: its adherence to the schema ontologies and domain constraints. To this end, the *metaphactory* platform utilizes the SHACL standard for defining and verifying the data constraints. SHACL provides an ontology for expressing data constraints in the form of *shapes*: RDF structures describing sets of restrictions that the data in the main knowledge graph must conform to. SHACL allows defining complex combinations of constraints more expressive than those supported by OWL. The *metaphactory* platform supports both SHACL constraints defined manually by the user as well as automatically generated from the OWL on-

[15] https://www.w3.org/TR/ldp/
[16] http://metaphacts.com/graphscope

tology. Auto-generator bootstrapping rules process the RDFS and OWL restrictions and generate corresponding SHACL shapes. The platform includes a SHACL execution engine based on the open-source RDFUnit[17] implementation that transforms SHACL shapes into SPARQL queries, executes them against the knowledge graph and generates a data quality report.

Connector services aim at pre-processing and exposing knowledge graph data in a way that makes it easily consumable by external applications. These services are primarily available for the needs of application developers utilizing the knowledge graph information in combination with other elements of the infrastructure within an organization. Examples of such services are the *Tableau connector service* that exposes the data for analysis by a popular Tableau application and the *Alexa skill service* that generates an Alexa[18] skill description to enable voice interaction with knowledge graph. To ease the integration of the platform into an organizational infrastructure, the Query-as-a-Service (QaaS) functionality is implemented. This allows exposing pre-saved parameterized SPIN query templates as custom REST APIs that are easy to call by internally developed tools within a customer organization. Ability to work with a REST API is usually preferred by internal developer teams to the need to compose and send SPARQL queries directly.

To reconcile the needs to support diverse user groups and to keep the platform adaptable to various different use cases and ease integration with other tools and applications in the organization, the service layer was implemented based on open standards whenever possible. While SPARQL 1.1 is used as a common query language, other standards are used for realizing more specific functionalities: for example, SHACL to specify and process the constraints over the data, LDP to enable management operations over RDF using HTTP requests, REST as a communication interface for external applications.

5. User Interface

The user interface design of the *metaphactory* platform aims at satisfying the requirements outlined in section 1: enabling reusability, compatibility, and customization. For this reason, the core platform interface is implemented in a generic way, providing tools for custom-building of all specific views for the use case at hand. In addition to that, the platform envisages external applications connecting to it, which gives possibility to use alternative user interfaces according to the user needs. One example of such an alternative UI is Amazon Alexa.

5.1. Customizable UI: templates and custom components

The web-based UI of the *metaphactory* platform is resource-centric: the platform renders an HTML view for a provided URI of a resource from the knowledge graph (Fig. 2). Resource views can be specified at different levels of abstraction: it is possible to define a dedicated HTML view for one specific resource as well as provide *templates* that will be applied to all resources of a specific kind. The criterion for choosing a template for a resource is configurable: by default, the template is selected according to the *rdf:type* property value, but this can be changed by providing a special template selection SPARQL query. This allows adapting the platform setup depending on the high-level structure of the knowledge graph.

Resource views and templates are defined as HTML pages. These can be created and edited directly in the platform using the provided HTML editor. Besides standard HTML tags, a view can contain special UI components: configurable custom client-side UI elements which can be referenced in the template source code as special HTML5 tags. A custom UI component receives its configuration parameters as tag attributes. A generic component that needs to interact with the knowledge graph (e.g., to visualize data) receives the data selection SPARQL query as part of its configuration and interacts with the platform backend service layer APIs to obtain the required data. In particular, the built-in components reuse the open-source SPARQL.js[19] library to operate with SPARQL queries.

Moreover, the web components provided by *metaphactory* are not only customizable individually, but are composable. Some component can define an environment that can include other nested components able to communicate via pre-defined interfaces. Such environments can define common configuration parameters that are applied to all included components as well as define a layout for these components. An example of such environment is a structured search com-

[17] https://github.com/AKSW/RDFUnit
[18] https://developer.amazon.com/alexa
[19] https://github.com/RubenVerborgh/SPARQL.js

Fig. 2. Example resource view for a protein in the Wikidata knowledge graph.

ponent that is able to combine different input controls for generating a search request (e.g., keyword search or form-based search) with various options for visualizing and exploring search results. With this approach, complex user interaction functionalities can be specified in a fully declarative way by composing pre-existing atomic elements without the need to build new components for each new use case application.

In this way, the user interface addresses the design requirements supporting reusability and custom application building. View templates and configurable custom components provided by the platform enable parts of the UI to be reused for different resources, datasets, and applications. Building a custom end-user application for a specific use case involves defining and packaging a set of required UI templates and configurations and does not require modifications of the platform itself. An application can be delivered as a simple plugin that can be installed automatically on top of the standard platform release.

The platform provides a wide range of custom components for various user needs primarily focusing on the functionalities most often required by end users: search, visualization, and authoring. The main protocol of interaction with RDF stores is the SPARQL query language: an expressive formalism, which however is not convenient for non-expert users. For this reason, search and visual exploration capabilities that would capture information needs in a user-friendly way and generate information retrieval queries based on them are crucial for an end-user tool working with knowledge graphs.

Semantic search and visual exploration capabilities complement each other in enabling the user interaction with the knowledge graph. Semantic search allows defining exact information needs in a user-friendly way and retrieving relevant data from the knowledge graph. On the other hand, “exploration is about efficiently extracting knowledge from data even if we do not know exactly what we are looking for” [10]. Structure of semantic datasets was exploited to generate facets for search refinement [11] and linked data browsing [12], where facets are selected based on data distribution within the dataset. The need to express information needs using complex SPARQL queries led to development of query builder tools providing visual assistance to the user. For example, the ExConQuer [13] framework provides an interface for constructing SPARQL queries where the user expresses her information need by selecting the concepts and properties from the ontology and building an abstract query structure which is afterwards translated into

a SPARQL query. OptiqueVQS [14] provides visual query formulation allowing the user to select relevant concepts and properties, express value restrictions, and visualize the resulting query pattern as a graph structure. QueryVOWL [15] combines elements of search and exploration as it provides a visual construction of queries over the linked data and converts the user-generated diagrams into SPARQL. However, its use of exploration is rather limited as its only goal is construction of the SPARQL query, rather than enabling the user working with its results. Others, like Lodlive [16] or Aemoo [17] primarily focus on browsing and graph-based exploration of data. In *metaphactory* we aimed at combining components for structured semantic search to capture the user needs and translate them into SPARQL queries with powerful exploration tools that would pick up the semantic search results and allow the user to further explore them (e.g, by means of the Ontodia tool [18]).

In its structured search functionality, the *metaphactory* platform puts emphasis on the capability to configure search and adapt it to different use cases with a minimal effort. In particular, search over the knowledge graph is realized as an environment, which allows specifying different ways for generating the search query, refining it, and visualizing the search results. A search request can be generated in several ways: e.g., by using a simple keyword search text field, by entering parameters in a pre-defined form, or by constructing the search criteria iteratively, selecting relevant properties and constraints. All these multiple approaches for expressing user information needs generate the search request in the same form: as a SPARQL query. The query can be further refined using facets that can be configured declaratively as part of the search environment. Finally, the search results returned by the produced query can be visualized by any appropriate UI component depending on the form of data and customer preferences: e.g., as a table or as a chart.

Built-in visualization components in the *metaphactory* platform provide the capabilities to show subsets of a knowledge graph in an informative way, facilitate exploration of the graph, and summarize the information. Depending on the structure of the data and the target view, the developer can select alternative visualization strategies, e.g., a table that condenses graph data into a more traditional relational format or a graph that emphasizes inter-relations between entities. For the latter, the *metaphactory* platform is integrated with a powerful Ontodia [18] tool for building custom RDF graph diagrams (Fig. 3). At the moment, Ontodia represents one of the most powerful ontology visualization tool taking into account the range of supported interaction techniques [19]. Ontodia allows not only visualizing parts of an RDF graph, including concepts, entities, relations, and properties, but also enables the user to further explore the dataset and extend the view in an interactive way. User experiments performed to validate Ontodia in combination with semantic search within the diagrammatic question answering workflow on the Wikidata dataset have shown that the system allows the users to interact with the knowledge graph effectively without having upfront knowledge of the dataset structure [20].

To summarize aggregated data and show the user the analysis results, various charts are available. Common special types of data, such as geospatial and temporal, can be visualized using appropriate views: e.g., map or timeline.

To support creation and editing of knowledge graphs, the platform provides end-user friendly, customizable, and extensible authoring UI based on the composite component environment and *knowledge graph patterns*. In essence, a knowledge graph pattern is a structure including a SPARQL query pattern with some additional metadata that is used for creation and validation of the user input. This concept helps to hide the complexity of the underlying data model from the end user, but at the same time allows expert users to precisely capture information needs and adjust authoring UI for these needs by using various components for data input, ranging from simple text inputs to hierarchical suggestion components and nested forms (Fig. 4). The application of knowledge graph patterns is not limited only to data authoring. The same pattern can be re-used in structured search for query generation and in visualization components for data retrieval. This re-usability helps to make sure consistency of the UI across the whole application.

5.2. *Expressive keyword search querying with GraphScope*

GraphScope is a data search engine for knowledge graphs that allows the user to access RDF data in a simple way by entering keyword queries. These keyword queries are interpreted by GraphScope and the results matching the information need are shown to the user. GraphScope tries to answer the user's query by finding the most suitable interpretation of each keyword with respect to concepts, properties, and instances of the knowledge graph. To this end, GraphScope relies on

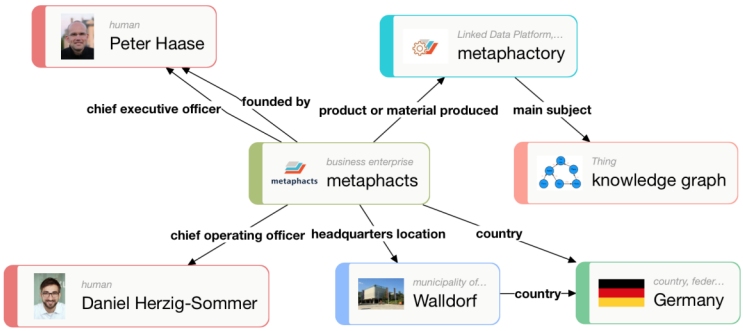

Fig. 3. Graph visualization with Ontodia.

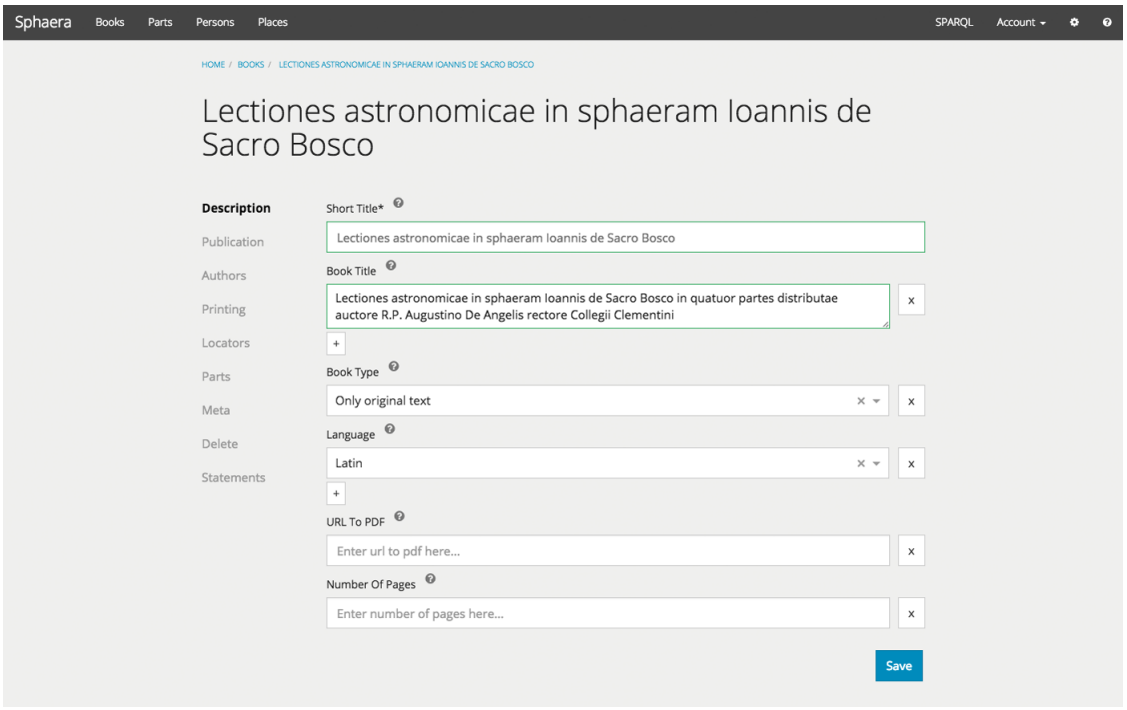

Fig. 4. Authoring forms using knowledge graph patterns.

a set of specialized indices that contain various meta-information about the distribution of entities and associated keywords in the data repository. Based on these indices, GraphScope identifies the most appropriate matches for the keywords in the user query and generates a corresponding SPARQL query to retrieve the results. After the system returned an initial result set for the user's keyword query, GraphScope provides possibilities to further refine the query and explore the initial result set. The user can select an appropriate interpretation for keywords to refine the meaning of the query as well as further expand the results by exploring the relations of the entities in the result set. These refinements, which the user performs from the UI, are automatically transformed into modifications of the SPARQL query.

GraphScope, originally developed as a standalone tool, was integrated with the *metaphactory* platform to serve as an additional query generation approach in the generic architecture. The integration is realized both at the backend and frontend levels. GraphScope and *metaphactory* reuse the same connection to the backend triple store to access data via SPARQL. At the frontend level, the GraphScope user interface for defining a keyword search query and refining the search re-

sults is wrapped as a metaphactory UI HTML5 component that can be added on a template page.

5.3. Voice interface using Amazon Alexa

Separating the service layer backend and the client-side UI makes it easy to integrate the platform with external tools at the UI level: an alternative UI can be deployed on top of the platform, while it can interact with the knowledge graph by invoking relevant platform services. An example of such alternative user interface is voice interaction using Alexa that can be used alongside the more traditional paradigms described above [21]. The Amazon Alexa framework allows the developer to define a specific service (called *Skill*) to process user requests and generate verbalized answers. The Alexa framework includes two parts: an *Alexa skill* which provides an abstraction over complex voice processing and generation services and an Amazon *Lambda service* implementing the application logic. An Alexa skill processes the voice messages from the user and transforms them into service requests with provided parameters. A skill can define a number of request types called *intents*, which can in turn be mapped to several *utterances* (natural language phrases). An utterance can be parameterized using *slots* (request parameters): for example, when Alexa receives a user's question "Alexa, ask Wikidata what is the capital of Poland", it extracts the skill name (Wikidata), identifies the intent corresponding to the question "what is the capital of...?" and the corresponding Lambda service. Then, the Lambda service is invoked with the extracted parameters: the ID of the intent (e.g., "capitalcity") and the request parameter ("Poland"). The Lambda service is then responsible for finding the right answer and returning it in a verbalized form. The Alexa service is then merely pronounces the returned verbalized answer.

Our Lambda function finds an answer to the query in the knowledge graph by mapping an intent to a SPARQL query pattern. For example, to answer our example question over the Wikidata knowledge graph, we need to find the value of the property *P36* "capital" for the entity *Q36* "Poland".

To handle such a direct factual question, our Lambda function uses a SPARQL query pattern of the following form (here using the Blazegraph full-text search):

```
SELECT ?answer WHERE {
    ?uri rdfs:label ?label.
    ?label bssearch:search "${entity}".
    ?label bssearch:minRelevance "0.5".
    ?label bssearch:matchAllTerms "true".
    ?uri ${property} ?answer .
} LIMIT 1
```

We use an automated procedure to bootstrap the Alexa skill definition and generate descriptions of intents as well as example entities and verbalization templates. Our Alexa skill is available in the Amazon Skill store under the name "metaphacts" in German and English[20].

6. Experiences and Lessons Learnt

The *metaphactory* platform is used in production in a variety of use cases involving knowledge graph management in different application domains. In the following we consider four diverse practical use cases from the open knowledge graphs, cultural heritage, energy industry, and life sciences domains highlighting different aspects of knowledge graph management.

6.1. Open Knowledge Graphs (Wikidata)

Wikidata[21] is a free and open knowledge graph containing general-purpose data across domains. It is used as a reference data storage for other Wikimedia projects (in particular, Wikipedia). Due to its comprehensive nature and popularity, there exists a large volume of mappings between Wikidata entities and instances of other repositories, including domain-specific ones (e.g., UniProt[22] for proteins, GeoNames[23] for geographical locations, etc.).

To exploit this data, we have set up a public showcase system for the community[24], easing access to the information provided by the Wikidata query service. The system provides different search interfaces as entry points into Wikidata's knowledge base and visualizes search results based on a comprehensive HTML5 based templating approach. Internally, the public *metaphactory* Wikidata system is used both as an integration hub to facilitate integration of data from multiple sources for the needs of industrial use cases as well as a demo system to highlight various platform features. Among others, the public Wikidata *metaphactory* system includes such functionalities as

- Structured search over Wikidata (configured for general-purpose person-organisation data as well as for the life sciences domain)

[20] https://www.amazon.com/metaphacts/dp/B0745KLCFX/ for US English.
[21] https://www.wikidata.org
[22] https://www.uniprot.org/
[23] http://www.geonames.org
[24] https://wikidata.metaphacts.com/

- Integration with the Wikidata free-text search service
- Integration with life science-specific repositories to show additional views
- Semantic similarity search based on a word2vec vector space embedding model [22]
- Geospatial search using a combination of GeoSPARQL[25] queries, external services like OpenStreetMap[26], and map-based visualizations.

Using the links to external repositories, the Wikidata *metaphactory* system is used to integrate external data in other use cases: e.g., to bring in geospatial data in the cultural heritage use case or additional protein tissue data from neXtProt[27] in the life science scenario.

6.2. Cultural Heritage

Knowledge graph technologies have become prominent in the context of the cultural heritage domain, where the CIDOC-CRM ontology[28] became a popular standard for exposing cultural heritage information as linked data. The *metaphactory* platform is utilized in the context of the ResearchSpace project[29] to manage the British Museum knowledge graph and help the researchers (a) explore meta-data about museum artifacts: historical context, associations with geographical locations, creators, discoverers and past owners, etc, and (b) use this meta-data in collaborative work by creating annotations, narratives involving semantic references, and argumentations exploiting knowledge graph data as evidence [23].

A crucial piece of functionality is the *structured search*, where the user can define complex multi-criteria information needs iteratively, by selecting appropriate clauses in the user interface (Fig. 5). A query request, which can look like "give me all bronze artifacts created in Egypt between 2500BC and 2000BC", then gets translated into SPARQL and answered using the knowledge graph data. Given the complexity and very high expressivity of the CIDOC-CRM ontology, using it directly at the level of user interface would make the system complicated for the end-user. For this reason, we defined a set of special *fundamental concepts* (FCs) and *relations* (FRs) [24], which abstract over physical classes and properties of the ontology, while being intuitively understandable to the user (e.g., Thing, Actor, Event and relations between them). With the structured search interface, the user can specify the criteria for data selection, interactively refine the initial selection results, explore the returned result set with faceted search and different results visualization views, and finally save the defined search for future reuse.

Knowledge graph data is used to support research collaborations by enabling the argumentation process and making assertions about graph entities following the direction outlined in [25]. User assertions have explicitly stored provenance information and can compose complex exchanges of arguments allowing to trace the whole reasoning process back to its origins and base evidence. Finally, the data selected from the knowledge graph can be used as references in *semantic narratives* that combine user-authored text, references to knowledge graph entities and different data visualizations supported by the platform: from images associated to entities to charts summarizing selected data subsets.

A demo installation of the ResearchSpace platform over the British Museum knowledge graph collection is available on the Web[30].

While the British Museum data collection was the first target use case in the cultural heritage domain, the resulting ResearchSpace platform extension was re-applied for other use cases in this area. Sphaera CorpusTracer [26], a practical application developed in collaboration with the Max-Planck Institute manages a knowledge graph addressing science history information: e.g., tracing the survived printed publications of medieval astronomy textbooks in the early modern period[31].

6.3. Engineering & Manufacturing Industry

Large-scale organizations involved in the engineering and manufacturing industry use knowledge graph representation to model master data: e.g., information about products and their typology, separate assembly parts, projects and their topics, etc. An inherent trait of these use cases is the condition that the knowledge graph constitutes only a part of a complex infrastructure involving heterogeneous data sources and large-scale networks of atomic devices able to communicate data. This raises a number of challenges for data man-

[25] http://www.opengeospatial.org/standards/geosparql
[26] https://www.openstreetmap.org
[27] https://www.nextprot.org/
[28] http://www.cidoc-crm.org/
[29] http://www.researchspace.org/
[30] https://demo.researchspace.org
[31] http://db.sphaera.mpiwg-berlin.mpg.de

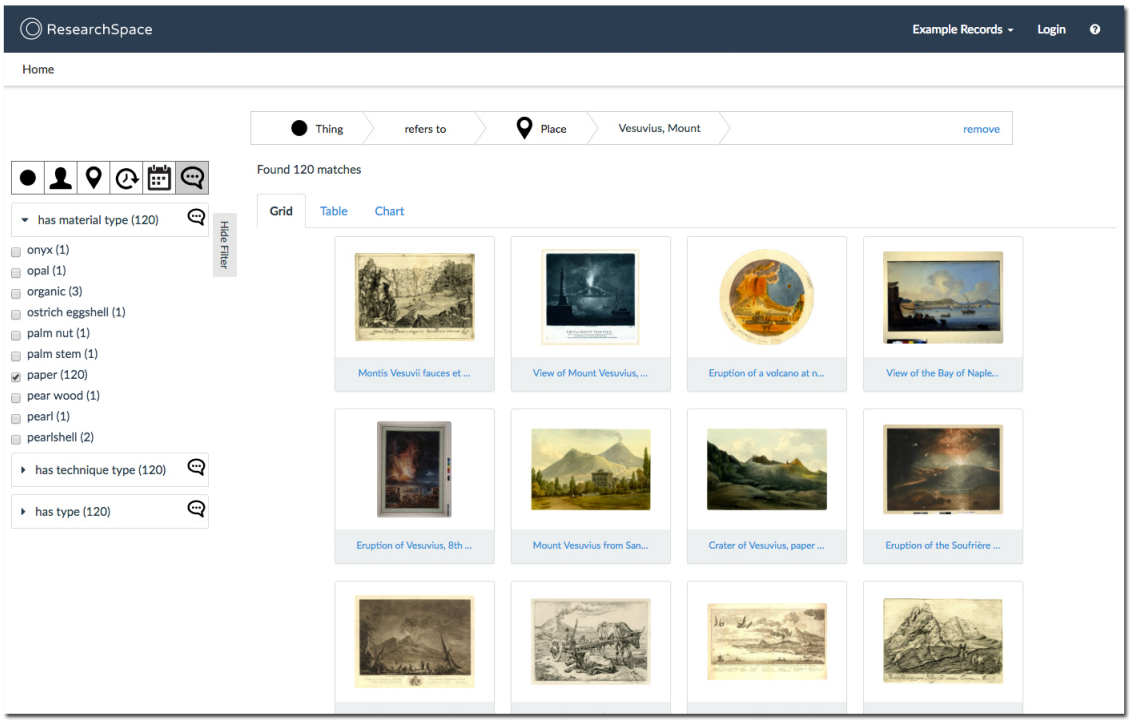

Fig. 5. Structured semantic search over British Museum artifacts.

agement and utilization: in particular, the need to integrate static knowledge with runtime data and apply potentially complex analysis procedures.

These challenges are present in full in the use case scenarios in which *metaphactory* is applied at Siemens [27]. For example, a typical Siemens gas turbine has about 2,000 sensors and a diagnostic task to detect whether a single particular purging operation is over requires applying multiple hundreds of signal processing rules. To be able to process relevant information in a timely way and achieve intelligent diagnostics, knowledge graph data must be integrated with runtime information provided by relevant APIs on demand. In this context, knowledge graph information acts as the integration backbone for the heterogeneous data infrastructure. To provide the integrated view of this hybrid infrastructure, the integration capabilities of the *metaphactory* platform play a crucial role.

The Ephedra federation architecture enables combining the "native" knowledge graph data with the data produced by the API services available in the Siemens infrastructure, such as MindSphere[32], as well as with analysis results produced by machine learning algorithms in a transparent way, at the level of SPARQL queries.

For example, one of the use cases involves using the knowledge graph to integrate and represent knowledge about a building in order to create a full virtual copy (Digital Twin) of the real building, including the static information describing the building as well as dynamic sensor data reflecting its current state. In another case, a trained machine learning model is exposed as a REST API that takes as input a list of features and returns as response the classification results of a text categorization problem. The use of Ephedra allows wrapping these diverse API as SPARQL federation members and processing federated queries that join the raw data with the predictions generated by machine learning. In this way, the hybrid data integration capabilities enabled by the platform enable several practical end-user scenarios within the Siemens infrastructure using a common approach to build several distinct knowledge graph applications targeting different user groups.

[32] https://siemens.mindsphere.io/

6.4. Life Sciences

The life sciences and pharmacy industry has been among the early adopters of the semantic web/knowledge graph technologies. Currently there exists a wide range of reusable life-science reference datasets already published in RDF and available in public linked data repositories (e.g., neXtProt [28], ChEMBL[33] [29] and others). In order to be exploited in an industrial environment, these datasets must be integrated with the internal data sources collected within the organization.

Such a scenario involved utilizing the *metaphactory* platform within the Sanofi S.A.[34] pharmaceutical company. The project focused on building a *target dashboard* to support internal decision making processes using integrated data from over 20 different data repositories including both company-internal and public data. The *metaphactory* platform served as a one-stop portal for consolidated access to target-related information such as function, expression, genetics, interactions/pathways, etc.

To achieve transparent querying of integrated data, relevant data sources were brought together using a combination of ETL processes and virtual integration. Data contained in relational databases were transformed using R2RML mappings. A high-level ontology was designed to serve as a global schema to abstract from the set of local ontologies of each data source. To support the end-user in finding and interpreting the data, the target dashboard user interface was built using the *metaphactory* UI templating functionality and the generic platform services. The dashboard includes complementary search mechanisms, which can be used depending on the user task at hand: structured search that realizes ontology-based graphical query building and keyword search. Search results are visualized using template pages according to the relevant types of entities. In addition, the data exposed via the platform API services is further exploited by third-party tools like KNIME to perform advanced data analysis.

Given the diversity of data to be integrated and the likelihood of error, ensuring the quality control becomes a matter of critical importance. To this end, the platform setup included the use of SHACL to verify the quality of the data. Using a combination of SHACL constraints automatically generated from the ontology as well as manually curated domain rules help to ensure that the added value of data integration is not compromised by reduced data quality.

6.5. Discussion

The *metaphactory* platform has been deployed in various knowledge graph management use cases targeted at different industries, organization types, and user groups. This experience provided us with valuable hints regarding the management and exploitation of knowledge graphs within enterprise organizations as well as the challenges which have to be addressed by a general-purpose knowledge graph management platform. The use cases demonstrate the importance for a knowledge graph application to adapt to the needs of existing data infrastructure aiming to complement it with missing functionalities rather than replace it. The knowledge graph platform must support rapid development and deployment of a use case application that would demonstrate the added value of knowledge graph technologies. Two examples of such functionalities are transparent data integration and semantic search.

Data integration needs of real-world customers often go beyond RDF data and relational/tabular data sources. While static data can be integrated by means of a dedicated ETL process and physical materialization of data as RDF, using dynamic data in combination with the knowledge graph requires extension of common data access mechanisms. The *metaphactory* approach to the problem includes support of hybrid information needs by means of extended SPARQL 1.1 federation capabilities. In this case, dynamic data exposed by means of a service can be accessed by request and combined with the knowledge graph information at the level of query results.

The use of open standards is crucially important in realizing interactions with third-party tools at all levels. For example, using SPARQL 1.1 as a generic interface for communicating with data repositories enables the platform to interact with any triple store selected by the customer without imposing any restrictions on the infrastructure. For the same reason, it is also beneficial if a knowledge graph management platform can provide its output using interfaces preferred by the organization's developer teams: for instance, if some data can be retrieved by sending a SPARQL query to the endpoint, it is still often preferable if the same data can be provided via a REST API call with a set of key-value parameters.

[33] https://www.ebi.ac.uk/chembl/
[34] http://www.sanofi.com

One generic observation regarding the requirements towards end-user supporting tools in the industrial setting concerns the way reusability is achieved. In academic research the focus is often on providing a maximally generic approach that is able to adapt to the specific task at hand by using intelligent heuristics: for example, by generating automatically the most suitable set of facets for semantic search results exploration or by automatic selection of relevant data sources to achieve transparent query federation. In the industrial setting, the possibility to adjust the configuration of some module manually in a very fine-grained way is usually more important, while automating heuristics can play only an auxiliary role. This is mainly due to the need to deal with "long tail" edge cases in the knowledge graph structure, for which a generic technique would fail: improving the coverage of a fully generic approach for such cases is usually more expensive than manual configuration. This often creates an obstacle for applying open source tools developed in the research community to industry use cases.

A survey in [30] introduces an assessment framework for knowledge graph tools, listing such dimensions as *human interaction*, *machine interaction*, and *strategic development*. Table 2 summarizes the features implemented in the *metaphactory* platform to support each of the assessment parameters of these dimension. To realize its goal of supporting the whole knowledge graph lifecycle, *metaphactory* realizes all dimensions and parameters listed in the assessment benchmark. No other tool evaluated in [30] includes all the features. Moreover, the variety of different use cases described in sections 6.1 to 6.4 demonstrate the platform ability to support rapid development of custom applications with minimal effort by providing custom configurations of core features rather than custom implementations.

However, based on the experience of using the platform in industrial use cases, it is apparent that support for some of these functionality dimensions still has limitations and requires further improvement. The most important of these are *data curation*, *linking*, and *analytics*. Manual knowledge graph population and editing is often a costly process where automation can provide added value in many ways, both to assist the user in manual curation workflows and to support automated knowledge graph population from unstructured sources. Given the variety of the data integration use cases, data interlinking challenges arise in different contexts and require the ability to deal with different data modalities and formats (e.g, free text, XML&JSON documents) as well as apply semi- or fully automated data interlinking algorithms. Finally, supporting data scientists in performing advanced data analytics tasks requires better integration of the platform with a variety of common machine learning and analysis tools as well as generic out-of-the box support for integration of analysis results back into the knowledge graph.

7. Conclusions and Outlook

The *metaphactory* platform has been developed with the aim to support interaction with knowledge graphs and utilize the added value of knowledge graph data within organizational environments. The trial version of the platform can be accessed for free after registration[35].

Given the feedback gathered in multiple use cases and the limitations listed in Section 6.5, we consider several directions particularly important to improve the current version of the platform.

- *Intelligent data authoring*. Whenever knowledge graph data has to be inserted manually by the user rather than imported from pre-existing sources, it often constitutes a cumbersome and time-costly process. It is important to assist the user in knowledge graph population tasks whenever possible providing intelligent auto-suggestions as well as validating and refining user input. To this end, the use of novel machine learning techniques for knowledge graph population, particularly, embedding-based methods appears particularly promising.
- *Integration with machine learning technologies*. Knowledge graphs provide added value serving as input for machine learning techniques producing new information, as well as can themselves be extended using machine learning. *metaphactory* platform already includes interfaces that can be utilized for interaction with machine learning tools. Providing configurable general-purpose extensions enabling application of machine learning tools with minimal effort would further benefit data scientists.

[35] http://www.metaphacts.com/trial, at the time of writing the version 2.3.2 is available.

Table 2
Assessing *metaphactory* platform features along benchmark dimensions [30]

Dimension	Parameter	Features
D1 Human Interaction	P1 Modeling	Taxonomies, Ontologies, Rules, Mappings
	P2 Curation	Collaborative, Form-based, GUI
	P3 Linking	Mappings, Virtual integration (Federation)
	P4 Exploration/Visualization	Charts, Maps, Graphs, Faceted browsing, Web GUI Templates
	P5 Search	Full-text, Structured, Faceted, Federated, NL question answering
D2 Machine Interaction	P6 Data Model	RDF, RDFS, OWL, RDBMS + R2RML
	P7 APIs	SPARQL, LDP, custom REST APIs
D3 Strategic Development	P8 Governance	Data workflows, Best practices
	P9 Security	Role-based AC (Data-level, API-level)
	P10 Quality & Maturity	SHACL
	P11 Provenance	Data-level (Context)
	P12 Analytics	Graph-based (Ontodia), ML, Statistical workflows (KNIME, Tableau)

– *Advanced data integration tasks*. This involves, in particular, platform-level support for managing mappings for integration of non-RDF structured (e.g., JSON, XML) and unstructured (e.g., text) sources as well as common integration procedures, such as instance matching.
– *Development of "blueprints" for common use cases*. Given commonalities between different use cases, it is desirable to develop common "blueprint" templates for applications. Such application templates will greatly reduce costs for building applications targeted at new use cases.

References

[1] A. Nikolov, P. Haase, J. Trame and A. Kozlov, Ephedra: Efficiently Combining RDF Data and Services Using SPARQL Federation, in: *8th International Conference on Knowledge Engineering and Semantic Web (KESW 2017)*, Szczecin, Poland, 2017, pp. 246–262.

[2] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro and G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web* **8**(3) (2017), 471–487.

[3] B. Quilitz and U. Leser, Querying Distributed RDF Data Sources with SPARQL, in: *7th International Semantic Web Conference (ISWC 2008)*, Vol. 5021, Springer, Karlsruhe, Germany, 2008, pp. 524–538.

[4] O. Görlitz and S. Staab, SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions, in: *COLD2011 Workshop, 10th International Semantic Web Conference (ISWC 2011)*, Bonn, Germany, 2011.

[5] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo and E. Ruckhaus, ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints, in: *10th International Semantic Web Conference (ISWC 2011)*, Bonn, Germany.

[6] M. Saleem and A.-C. Ngonga Ngomo, HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation, in: *11th Extended Semantic Web Conference (ESWC 2014)*, Heraklion, Greece, 2014.

[7] A. Schwarte, P. Haase, K. Hose, R. Schenkel and M. Schmidt, FedX: Optimization Techniques for Federated Query Processing on Linked Data, in: *10th International Semantic Web Conference (ISWC 2011)*, Bonn, Germany, pp. 601–616.

[8] B. Stringer, A. Meroño-Peñuela, S. Abeln, F. van Harmelen and J. Heringa, SCRY: Extending SPARQL with Custom Data Processing Methods for the Life Sciences, in: *9th International Conference on Semantic Web Applications and Tools for Life Sciences (SWAT4LS)*, Amsterdam, The Netherlands, 2016.

[9] J. Dolby, A. Fokoue, M. Rodriguez-Muro, K. Srinivas and W. Sun, Extending SPARQL for Data Analytic Tasks, in: *15th International Semantic Web Conference (ISWC 2016)*, Kobe, Japan, 2016, pp. 437–452.

[10] S. Idreos, O. Papaemmanouil and S. Chaudhuri, Overview of Data Exploration Techniques, in: *ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, Melbourne, Victoria, Australia, 2015, pp. 277–281.

[11] M. Tvarozek and M. Bieliková, Generating Exploratory Search Interfaces for the Semantic Web, in: *Human-Computer Interaction - Second IFIP TC 13 Symposium, (HCIS 2010)*, Brisbane, Australia, 2010, pp. 175–186.

[12] T. Franz, J. Koch, R.Q. Dividino and S. Staab, LENA-TR : Browsing Linked Open Data Along Knowledge-Aspects, in: *Linked Data Meets Artificial Intelligence, 2010 AAAI Spring Symposium*, Stanford, CA, USA, 2010.

[13] J. Attard, F. Orlandi and S. Auer, ExConQuer: Lowering barriers to RDF and Linked Data re-use, *Semantic Web* **9**(2) (2018), 241–255.

[14] A. Soylu, M. Giese, E. Jiménez-Ruiz, G. Vega-Gorgojo and I. Horrocks, Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users, *Universal Access in the Information Society* **15**(1) (2016), 129–152.

[15] F. Haag, S. Lohmann, S. Siek and T. Ertl, Visual Querying of Linked Data with QueryVOWL, in: *SumPre 2015, HSWI 2015 Workshops, 12th Extended Semantic Web Conference (ESWC 2015)*, Portoroz, Slovenia, 2015.

[16] D.V. Camarda, S. Mazzini and A. Antonuccio, LodLive, exploring the web of data, in: *8th International Conference on Semantic Systems, I-SEMANTICS '12*, Graz, Austria, 2012, pp. 197–200.

[17] A.G. Nuzzolese, V. Presutti, A. Gangemi, S. Peroni and P. Ciancarini, Aemoo: Linked Data exploration based on Knowledge Patterns, *Semantic Web* **8**(1) (2017), 87–112.

[18] D. Mouromtsev, D. Pavlov, Y. Emelyanov, A. Morozov, D. Razdyakonov and M. Galkin, The Simple Web-based Tool for Visualization and Sharing of Semantic Data and Ontologies, in: *14th International Semantic Web Conference (ISWC 2015)*, Bethlehem, PA, USA, 2015.

[19] M. Dudás, S. Lohmann, V. Svátek and D. Pavlov, Ontology visualization methods and tools: a survey of the state of the art, *Knowledge Engineering Review* **33** (2018).

[20] D. Mouromtsev, G. Wohlgenannt, P. Haase, D. Pavlov, Y. Emelyanov and A. Morozov, A Diagrammatic Approach for Visual Question Answering over Knowledge Graphs, in: *ESWC 2018 Satellite Events*, Heraklion, Crete, Greece, 2018, pp. 34–39.

[21] P. Haase, A. Nikolov, J. Trame, A. Kozlov and D.M. Herzig, Alexa, Ask Wikidata! Voice Interaction with Knowledge Graphs using Amazon Alexa, in: *16th International Semantic Web Conference (ISWC 2017)*, Vienna, Austria, 2017.

[22] A. Nikolov, P. Haase, D.M. Herzig, J. Trame and A. Kozlov, Combining RDF Graph Data and Embedding Models for an Augmented Knowledge Graph, in: *BigNet@WWW'18 Workshop, The Web Conference 2018, (TheWebConf 2018)*, Lyon, France, 2018, pp. 977–980.

[23] D. Oldman and D. Tanase, Reshaping the Knowledge Graph by connecting researchers, data and practices in ResearchSpace, in: *17th International Semantic Web Conference (ISWC 2018)*, Monterey, CA, USA, 2018.

[24] K. Tzompanaki and M. Doerr, Fundamental Categories and Relationships for querying CIDOC-CRM based repositories, Technical Report, TR-429, ICS-FORTH, 2012.

[25] M. Doerr, A. Kritsotaki and K. Boutsika, Factual argumentation - a core model for assertions making, *JOCCH* **3**(3) (2011), 8–1834.

[26] F. Kräutli and M. Valleriani, CorpusTracer: A CIDOC database for tracing knowledge networks, *DSH* **33**(2) (2018), 336–346.

[27] T. Hubauer, P. Haase and D.M. Herzig, Use Cases of the Industrial Knowledge Graph at Siemens, in: *17th International Semantic Web Conference (ISWC 2018)*, Monterey, CA, USA, 2018.

[28] L. Lane et al., neXtProt: a knowledge platform for human proteins, *Nucleic Acids Research* **40**(Database–Issue) (2012), 76–83.

[29] E.L. Willighagen et al., The ChEMBL database as linked open data, *J. Cheminformatics* **5** (2013), 23.

[30] M. Galkin, S. Auer, M. Vidal and S. Scerri, Enterprise Knowledge Graphs: A Semantic Approach for Knowledge Management in the Next Generation of Enterprise Information Systems, in: *19th International Conference on Enterprise Information Systems (ICEIS 2017)*, Porto, Portugal, 2017, pp. 88–98.