Triple Pattern Interfaces for Object-Oriented Software APIs

Maximilian Weigand[1]

[1]Institute of Automation Technology, Helmut-Schmidt-University / University of the Federal Armed Forces Hamburg, Holstenhofweg 85, 22043 Hamburg, Germany

Abstract

In multi-disciplinary engineering projects, engineers collaborate and need to exchange information in between different engineering software. Often, the lack of interoperability of software requires engineers to implement complex transfer processes to overcome the heterogeneous syntax and semantics of information. Although knowledge graphs are proven to be a solution to overcome this heterogeneity, engineering software often does not provide interfaces for knowledge graph-based information exchange. To mitigate this lack, this research proposal presents an approach to create virtual knowledge graphs from software-internal data. Following the motivation and importance of the approach, it is broken down into several research questions, as well as research artifacts. The artifacts include software components that virtualize software-internal data as RDF knowledge graphs and make them queryable with SPARQL. Further, other preliminary results including a proof-of-concept implementation are presented. Finally, the further work that is required to evaluate and generalize the approach is specified.

Keywords

Virtualization, Knowledge Graph, Interoperability, Software

1. Problem Statement

In the context of multi-disciplinary engineering, heterogeneous data and information needs to be exchanged within software toolchains [1]. For example, in order to program an assembly process in a robot software, an automation engineer might need the positions of bolts from an assembly drawing created in a CAD software. This kind of exchange is still often implemented with manual exports and imports, possibly with tedious transformation processes in between.

Often, an engineering software offers an Application Programming Interface (API) that can be used to extend the functionality of this software. Within the scope of this publication, the term *API* refers to such software APIs. These differ from web APIs, which are interfaces to servers in a network and are currently the more common meaning of the term *API*. Software APIs of engineering software can be used to automate information exchange processes within software toolchains. However, this requires a high implementation effort and is not a reusable solution, as APIs are usually structurally and functionally different.

Ontologies and knowledge graphs (KGs) are sustainable solutions for information exchange. W3C standards, such as RDF(S) and OWL, set a uniform syntax and enable specifying the

Doctoral Consortium at ISWC 2023 co-located with 22nd International Semantic Web Conference (ISWC 2023)

maximilian.weigand@hsu-hh.de (M. Weigand)
https://www.hsu-hh.de/aut/team/weigand (M. Weigand)

semantics of the exchanged information. With SPARQL, there is also a standard for retrieving KG-based information systematically.

Despite of this, the majority of software in engineering does not support RDF exports or SPARQL interfaces to query information directly. As a result, there is a need to be able to efficiently develop interfaces for engineering software that enable this kind of information exchange. The focus of this work will be on SPARQL interfaces, as RDF exports alone are a static solution and can be implemented quite easily once a SPARQL interface has been realized.

2. Importance

A solution for the stated problem would be important for the semantic web community, software engineering and classic engineering domains, such as mechanical engineering.

In engineering, ontologies and KGs gain in popularity as means for information exchange. Facilitating this exchange would increase the acceptance of semantic web technologies in this domain. In the long term, this would result in an efficient and sustainable information management within engineering companies.

Within engineering companies, software solutions or add-ons for existing software are usually not implemented internally, but supplied by software companies. Those software companies could also benefit from the presented approach, as it lowers the effort for developing a software with a SPARQL interface or an add-on for an existing software. While the motivation is described from an engineering point-of-view, the approach can also be implemented in the software of other domains. Additional (SPARQL-)interfaces increase the openness of software which creates an additional value.

Within the semantic web community and in particular the Ontology-Based Data Access (OBDA) research community, this approach is novel, as it enables access to a new kind of data source. Also, research on KG integration or federated query systems can benefit from this.

3. Related Work

By the definition of Poggi et al., ”Ontology-based data access is the problem of accessing a set of existing data sources by means of a conceptual representation expressed in terms of an ontology” [2]. Within the OBDA research field, different approaches and tools exist, particularly for accessing data from relational databases (RDBs).

Data virtualization in general means to present arbitrary data to users according to a global schema, without details of the actual data sources. The virtual KG approach therefore has a KG as its global schema. The data source itself is not materialized in the global schema during access, but kept virtual. [3]

OBDA systems for RDBs, such as the Ontop Virtual Knowledge Graph system [4], rewrite SPARQL queries into queries in the RDB’s native query language, e.g. SQL. In order to do so, Ontop parses the SPARQL query and generates an algebra expression [4]. The SPARQL algebra is a standardized set of computational operations to evaluate SPARQL queries [5]. Among others, those operations include filters, joins and data access operations. From the SPARQL algebra expression, Ontop generates one single SQL query. As RDB systems are often highly optimized regarding query answering time, this promises high efficiency, as the overhead for making multiple requests to the RDB system and reconciling the results is minimal. In Ontop, querying implicit information is possible due to additional mappings that are created based on the ontology of the knowledge graph before querying [4]. In further research, it has been shown that this approach is also applicable to non-relational database systems [6].

In contrast to Ontop, an early approach in OBDA, D2RQ [7], rewrites only the data access operations of the SPARQL algebra expression into SQL queries. D2RQ implements the *Graph* interface [7] of the Jena2 Semantic Web toolkit [8].

The current implementation of the Apache Jena Framework[1] features a similar interface (*GraphBase*) that can be implemented to encapsulate non-RDF data as an RDF graph [9]. The Apace Jena query engine ARQ parses SPARQL queries and generates algebra expressions consisting of different operations. The data access operations in SPARQL are *triple patterns* and consist of a subject, predicate and object node, each being either bound or unbound. Also complex operations, as sequences of nodes linked with a fixed or arbitrary number of predicates (*property paths*), are handled by the query engine by executing multiple triple pattern operations. The *GraphBase* interface in Apache Jena allows to implement this operation with a custom logic, while other operations are executed by the query engine conventionally. Using this, arbitrary data sources can be accessed, thus creating a virtual KG from the data source.

A similar kind of interface was described by Verborgh et al. in their work on *triple pattern fragments* (TPFs). A TPF is a subset of all triples in a KG matching a triple pattern. TPFs are divided into these subsets to control and limit data transfer. Within a TPF, the approximate number of all matching triples, as well as controls to access all other matching TPFs are included. TPFs lower the server-side effort for exploring RDF graphs, but require client-side intelligence in order to handle the evaluation of the SPARQL query. [10]

This research proposal focuses on the creation of SPARQL-based interfaces for APIs of engineering software. As APIs are usually structurally and functionally different, a lightweight interface is required. A query rewriting approach is not realizable, as APIs cannot be expected to be queryable in a dedicated query language. An interface based on triple patterns is therefore a reasonable solution.

The client-side of such interfaces, i.e. the entity requesting triple patterns, is well researched by Verborgh et al. [10] and is also included in the Apache Jena Framework [9]. Regarding the server-side, i.e. the entity accepting triple pattern requests and replying with matching triples, different approaches for linked data documents exist [10], but no approaches exist for arbitrary APIs of software.

The identified gap that will be addressed within this research project is therefore the creation of server-side triple pattern interfaces for APIs of engineering software.

4. Research Questions

The main research question within this research proposal is:

RQ0 *How can ontology-based data access be performed on internal data of a software at its runtime?* This research question is further decomposed into the following partial research

[1]https://jena.apache.org

questions **RQ1** to **RQ5**.

Assumption *The target software has to feature an API following the object-oriented programming paradigm.* While the existence of an API is a minimal requirement to interact with the software at its runtime, the limitation to the object-oriented programming paradigm is a scope-limiting decision.

RQ1 *How can the structure of the data available from object-oriented software APIs be mapped to the structure of a KG?*

H1 In object-oriented programming and KGs, similar concepts exist, like objects and resources or classes and types. A mapping can be formally defined for further usage.

RQ2 *What is the generic structure and functionality of an add-on for the target software enabling a KG-based information exchange?*

H2 As described Sec. 3, an interface based on triple patterns is a reasonable solution as it limits the server-side complexity, i.e. the complexity of the add-on. Functions to access and retrieve data from the softwares API according to the requested triple pattern have to be implemented. They can be implemented in a generic way based on the generic mapping mentioned in **H1**.

RQ3 *How can the implementation of software-specific add-ons be automated or facilitated?*

H3 Usually, documentation for APIs exist, e.g. the *javadoc* for APIs implemented in Java. This documentation can be analyzed automatically. The results can be used to support or automate the implementation of the software-specific add-on in combination with the generic description mentioned in **H2**.

RQ4 *How can a software add-on based on a triple pattern interface be queried using SPARQL?*

H4 As described in Sec. 3, triple pattern interfaces need an intelligent client, that parses SPARQL queries and decomposes it into triple patterns in the first place. To query the data of a software using SPARQL, a middleware handling the SPARQL queries has to be implemented. However, the middleware can be a generic solution that does not need to be customized to the software.

RQ5 *Is it possible to modify this approach to query implicit information?*

H5 The add-on mentioned in **H2** cannot be expected to do any reasoning on the software-internal data. Therefore, querying implicit information must be realized by the middleware mentioned in **H4**. The applicability of approaches for existing OBDA systems (as discussed in Sec. 3) to a triple pattern interface based system has to be checked.

5. Preliminary Results

5.1. Published Work

In a first publication [11], the problem context and the motivation for this research project has been described.

In the publication, an architecture has been proposed, which is conceptually shown in Fig. 1. It contains an engineering software, an add-on for this software implementing the triple pattern interface, and a middleware handling the SPARQL query processing. The middleware is implemented by modifying a graph database with a SPARQL query engine that can be used by human operators or other applications. The graph database contains a graph for the software's

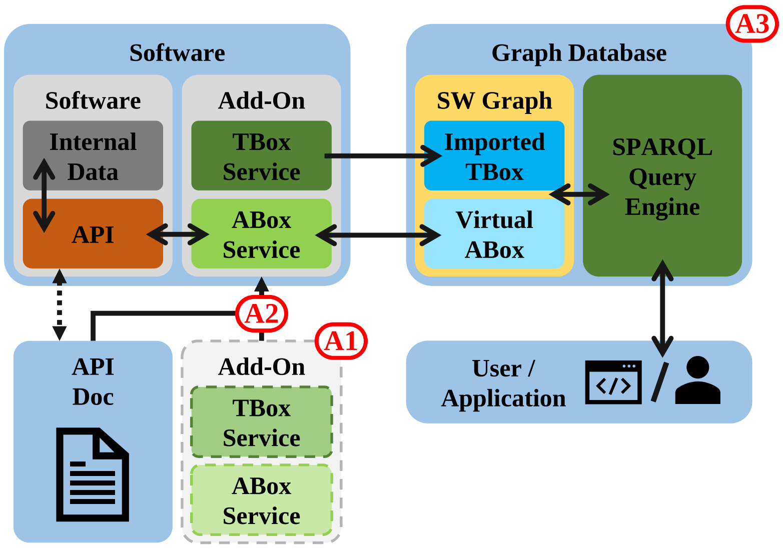

Figure 1: Proposed Architecture

data that is a union of two sub-graphs, an imported, materialized TBox and a virtualized ABox. The materialized TBox is a static KG that is imported from the TBox service of the add-on. The virtualized ABox is a virtual KG as it does not contain materialized data, but redirects triple pattern requests from the SPARQL query engine to the ABox service of the add-on. Redirected triple pattern requests are handled by the ABox service by retrieving internal data from the software via its API. This part of the architecture has also been demonstrated in a proof-of-concept implementation. The corresponding code is available on GitHub[2]. [11]

5.2. Research Artifacts

This research project is oriented on the Design Science Methodology by Wieringa [12]. It is therefore organized in the research artifacts **A1** to **A3**, that will be presented in this subsection. The artifacts are also placed representatively in Fig. 1.

A1 *A software component to handle triple pattern request and to evaluate them over data that is available from an object-oriented software API.*
As described in Sec. 3, triple pattern interfaces are a reasonable solution to implement KG-based access to arbitrary data sources, in this case to an object-oriented API. The primary objective of this artifact is to establish an intermediary layer between incoming triple pattern requests and the API's functionality for data access. This intermediary layer is based on a mapping between object-oriented programming concepts and concepts of KGs. For example, objects can be mapped to resources (in terms of RDF), while methods can be mapped to predicates. This mapping is the foundation of a functional description of the process of evaluating triple pattern requests over object-oriented data. This includes steps that can be based on the mapping, e.g. calling the method mapped to the predicate if a triple pattern includes the predicate. Other steps

[2]https://github.com/mxweigand/xapi

require auxiliary functions, e.g. finding all objects of a class mapped to a `rdf:type` or finding an object based on the URI of a resource. Both, the mapping and the functional description were briefly addressed in the first publication and are implicitly included in the proof-of-concept implementation [11]. However, complete and generalized versions, defined in a formalism accepted within the research community, still need to be defined. Using this, the functionality and structure of a generic add-on can be described, e.g. in UML, to serve as a template for software-specific implementations. Within the design of this artifact, research questions **RQ1** and **RQ2** are addressed.

A2 *A method to analyze the documentation of a software's API in order to support the implementation of a software-specific add-on.*
This artifact is based on the generic add-on that results from **A1**. The generic add-on should work like a template, with an implemented generic logic and placeholders for software-specific functions and parameters. The method described in this artifact is a structured analysis of the documentation of a software's API. Obtained results should either directly be used to define the template's placeholders, or be documented in machine-readable way, in order to fill the template automatically afterwards. The method will contain necessary steps, as identifying the functions of the API that are required for the add-on to work, as well as optional, repeatable steps. Latter are, for example, the identification of classes of the API that can be mapped to a type in the KG. Within the design of this artifact, research question **RQ3** is addressed.

A3 *A software component that evaluates SPARQL queries by requesting data from triple pattern interfaces and combining the retrieved data.*
In the architecture proposed in the previous subsection, this artifact is solved by modifying an existing graph database. In particular, the Apache Jena Framework, which offers certain extension points (cf. Sec 3), was used in the proof-of-concept implementation [11]. Another option is to base this artifact on TPF clients as described by Verborgh et al. [10]. The first publication has shown that with such approaches, only data that is explicitly available from the API can be queried by the modified graph database [11]. Possibilities to query implicit data have to be examined. This would allow multiple data sources to be integrated, which means a better use of the potentials of KGs. For RDB OBDA systems, such approaches exist and have to be examined, in order to determine if they can be adapted to this approach. An optional topic within this artifact is *query federation*, i.e. evaluating queries over multiple data sources, which has also been addressed in the context of triple pattern fragments [13]. Within the design of this artifact, research questions **RQ4** and **RQ5** are addressed.

Artifact **A1** and **A3** have been conceptually presented in a first publication [11], without showing how to generalize the approach. Minor work has been done regarding artifact **A2**.

6. Evaluation Plan

In the first step, artifact **A1** and **A2** will be evaluated, testing research questions **RQ1**, **RQ2** and **RQ3**. An exemplary software fulfilling the basic assumptions defined in Sec. 4 will be used as a test case. Using the method defined in artifact **A2**, the documentation of the software's API will be analyzed. In combination with the generic add-on defined in artifact **A1**, this results in a usable software-specific add-on. As described in artifact **A1**, the analysis includes necessary and optional, repeatable steps. Those will be documented separately. The evaluation criteria in this first step will be the effort for the user to create the software-specific add-on. While no metric to measure the effort has been determined yet, two factors have been identified and will be taken into account: the amount of documentation the user has to read and the amount of code the user has to write to extend the generic add-on. An important impact on the effort will be the share of the API's data model, that the user decides to map to KG entities. The effort will therefore be divided into necessary and optional effort.

In the second step, artifact **A3** will be evaluated, testing research questions **RQ4** and **RQ5**. This evaluation step will reuse the results of the first evaluation step, i.e. the software-specific add-on. The software component resulting from artifact **A3** does not need software-specific customization and should work with the previously created add-on. This can be evaluated by defining test data in the software and test queries to retrieve parts of this data. The query results will be compared with expected results that have to be defined in advance, in order to analyze the validity and completeness of the results, the primary evaluation criteria in the second step. A secondary criterion is the absolute performance of the approach (in terms of query evaluation time), which can be analyzed by defining several queries with different levels of complexity. The relative performance compared to other approaches can not be evaluated, as no other approaches with this kind of data source exist so far. However, it is possible to compare the performance of queries over the virtual KG of this approach with the performance of queries over an equivalent but materialized data source. The virtual graph then has to be materialized in the first place, which can be realized by entirely retrieving it via the triple pattern interface. Afterwards, the materialized graph can be stored in a graph database and queried using the same set of queries as for the virtual KG.

Within the first publication [11] described in Sec. 5, an add-on for a software for model-based systems engineering (MBSE) was implemented as a proof-of-concept. As test data, a process description of a robotic manufacturing process was queried using the newly created SPARQL interface. The information retrieved was the sequence of sub-processes as well the associated target positions for the robot's movement. This use case can be extended for evaluating this research project. In order to conduct the evaluation of artifact **A1** and **A2**, a triple pattern interface can be created for a 3D-CAD software. This interface can be reused for the evaluation of artifact **A3**. With this use case, query federation can also be examined. The existing MBSE-interface can be used to retrieve the process sequence and the new 3D-CAD-interface can be used to retrieve target positions from a 3D-CAD model.

7. Reflection and Future Work

Artifact **A1** and **A3** have been conceptually presented in a first publication [11], without showing how to generalize the approach. As artifact **A3** can be based on existing implementations, as shown in Sec. 3, it will not be focused in detail in the next publication. The essential next steps in this research project are therefore to generalize the existing results concerning artifact **A1** and to examine artifact **A2**. Consequently, research questions **RQ1**, **RQ2** and **RQ3** are the most relevant ones within this research project.

Especially the mapping between object-oriented programming concepts and concepts of

KGs and the functionality of the generic add-on are important results for the remainder of this research project. Those topics will therefore be addressed in the next publication.

Acknowledgments

I would like to express my thanks to my supervisor Prof. Dr.-Ing. Alexander Fay for supporting my research work. This work has been performed within the research project ADAPT (20D1916C) supported by the Federal Ministry for Economic Affairs and Climate Action (BMWK) based on a decision taken by the German Bundestag.

References

[1] F. J. Ekaputra, M. Sabou, E. Serral, E. Kiesling, S. Biffl, Ontology-Based Data Integration in Multi-Disciplinary Engineering Environments: A Review, Open Journal of Information Systems 4 (2017) 1–26.

[2] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking Data to Ontologies, in: Journal on Data Semantics X, 2008, pp. 133–173.

[3] G. Xiao, L. Ding, B. Cogrel, D. Calvanese, Virtual Knowledge Graphs: An Overview of Systems and Use Cases, Data Intelligence 1 (2019) 201–223.

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

[5] P. Hitzler, M. Krötzsch, S. Rudolph, Foundations of Semantic Web Technologies, 2009.

[6] E. Botoeva, D. Calvanese, B. Cogrel, M. Rezk, G. Xiao, OBDA Beyond Relational DBs: A Study for MongoDB, in: Proceedings of the 29th International Workshop on Description Logics, 2016.

[7] C. Bizer, A. Seaborne, D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs, in: Proceedings of the 3rd International Semantic Web Conference (ISWC2004), 2004.

[8] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, Jena: Implementing the Semantic Web Recommendations, in: Proceedings of the 13th International World Wide Web Conference, 2004.

[9] The Apache Software Foundation, ARQ - Extending Query Execution, 2021. URL: jena.apache.org/documentation/query/arq-query-eval.html, last accessed 2023/07/03.

[10] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, P. Colpaert, Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web, Journal of Web Semantics 37–38 (2016) 184–206.

[11] M. Weigand, A. Fay, Creating Virtual Knowledge Graphs from Software-Internal Data, in: IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society, 2022.

[12] R. J. Wieringa, Design Science Methodology for Information Systems and Software Engineering, 2014.

[13] R. Taelman, J. Van Herwegen, M. Vander Sande, R. Verborgh, Comunica: a Modular SPARQL Query Engine for the Web, in: Proceedings of the 17th International Semantic Web Conference, 2018.