A Microservice-based Infrastructure for the Academic Research on Semantic Knowledge Graphs and Derived Applications

Florian Rommel[1*], *Wolfgang* Ziegler[1**]

[1]Karlsruhe University of Applied Sciences, Faculty of Information Management and Media, 76133 Karlsruhe, Germany

Abstract. For some time past, conceptual and technological approaches in the field of semantic knowledge graphs (ontologies) have been finding their way into the domain of technical writing, technical information management and knowledge management, with the aim of enhancing the classical classification-based methods and thereby, solving problems that either cannot or can only be solved with difficulty in form of previous approaches. As these concepts are quite new in this field, they are still subject to current research in the academic field. This paper aims at providing a microservice-based software platform to support further studies in this area. In the sense of exploratory prototyping, the platform should make it possible to research various technological and conceptual questions and to demonstrate applications derived from them, regardless of the development status of existing, proprietary productive software applications.

1 Introduction

In contrast to the well-known hierarchical taxonomy-based classification models that are typically being used in the technical communication domain, which describe class-to-sub-class relationships, ontologies enable for the declaration of manifold types of logical relationships between any nodes of the ontology. With this characteristic – their ability to form an interweaved interwoven web, instead of a simple tree-hierarchy, they provide an opportunity for many different use cases in the field of technical communication.

Nowadays, ontologies are developed, explored, and used in the field of technical communication in various areas of application, in different forms of complexity for various purposes [1].

In parallelized and decoupled (agile) development processes, ontologies are being used to describe cross-organizational and cross-platform models (digital twins). Therefore, they allow the planning and creation of component-based information units alongside product development [2][3].

Another example for the use of ontologies in technical communication is the standardized exchange format iiRDS by tekom. One objective of this approach is to provide a common vocabulary for information units described by metadata that takes today's requirements into account (industry 4.0) to locate information units from different suppliers in a "cyber-physical" space [4].

Another recent field of application for ontologies in the domain of technical communication are semantic correlation rules (SCR). SCR allow to define relationships between information units based on metadata and, therefore, allow for the dynamic aggregation of so called microDocs. The concept and underlying technology are currently being implemented in CMS and CDP and by that, become more and more relevant to system providers and content engineers [1][5].

The applications of semantic networks in technical communication shown here, represent only a part of the conceivable areas of application. Although many applications have already been tested and used in practice, the use of semantic knowledge networks in technical communication is still the subject of current scientific research.

To be able to research and implement the applications presented here, a suitable software infrastructure is required. This paper is intended to deal with precisely these technological fundamentals and aims at providing a microservice-based software platform to support further studies in this area.

* Florian Rommel: hska@florianrommel.de
** Wolfgang Ziegler: wolfgang.ziegler@h-ka.de

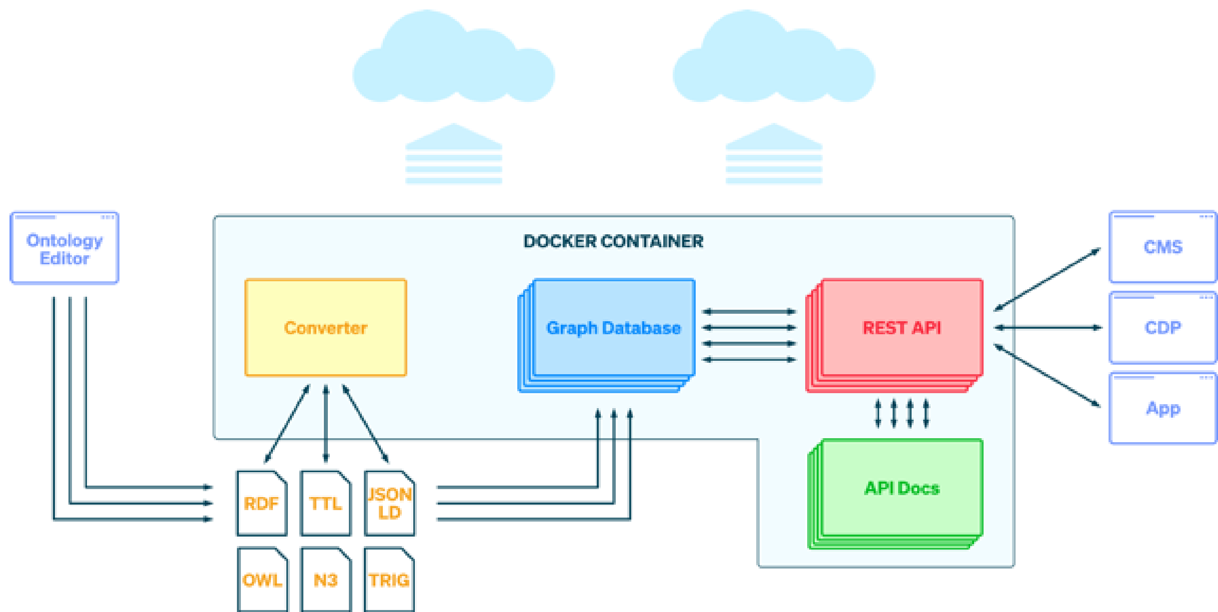

Fig. 1. Platform overview

2 Objectives

The goal of this paper is to provide a microservice-based software platform for the research and explorative prototyping of ontology-aided services in the field of technical communication. The software platform will allow to test different graph databases, to explore ontology modelling and to develop simple webservices and applications based on semantic models.

The software platform consists of components that are delivered preconfigured and ready to use (such as databases) and components that are highly use-case oriented and therefore, students will have to implement them themselves (such as REST APIs based on semantic models). For the latter, the platform has demonstration ontologies and components for learning purposes.

The need for such a platform arises from the current teaching situation: Although, there are many commercial software solutions to create ontologies and incorporate them into an existing complex software landscape in technical communication, however, these commercial software solutions often lack the flexibility that is required in a University environment. Also, with their own terminology, their specific functions, and their own graphical user interfaces, they tend to build an abstraction layer between the underlying technologies and the user.

In addition, the software platform demonstrates the interaction of systems and applications in modern software environments. Students can learn hands-on how different services are built and exchange data, how REST APIs work, and applications consume the resources behind the endpoints of those APIs.

3 Design requirements

This section is intended to provide an insight into the general and technological requirements, the desirable properties of the software platform and why these are important. It also reveals how these requirements are met.

3.1 Platform independence

One major design requirement was platform independence. The platform should be easy deployable and transferable in different environments, whether it is on a client personal computer, on an on-premises server or in a cloud environment. This implies, that the software platform also needs to be operating system agnostic.

3.2 Flexibility and expandability

To meet future demands, the software platform needs to be flexible and expandable. Therefore, instead of following a monolithic approach, the software platform consists of multiple microservices, each one serving a single purpose and exchanging data over standard protocols. By that, components can be exchanged, edited, and added, without affecting the entire software platform.

3.3 Virtualizability

Another design goal was virtualization. Having the software platform virtual allows for easier administration, the ability to backup current states (snapshots), backup recovery and it makes the platform easily transferable. The software platform can be virtualized in two ways: The use of container virtualization virtualizes the platform components

themselves. Additionally, the sever hosting the docker container virtualization can be virtualized as well.

3.4 Containerized infrastructure

All of the afore mentioned design requirements can be solved by utilizing container virtualization. Linux-based application containers can run on all major operating systems (with some additional tooling) and can, thereby, be seen as an independent platform. They can also easily be deployed in corresponding cloud environments. As application containers are ideal for the deployment of microservices, this approach can also solve the requirements flexibility, expandability and virtualizability. In addition, application containers can very easily be provisioned, duplicated, and transferred.

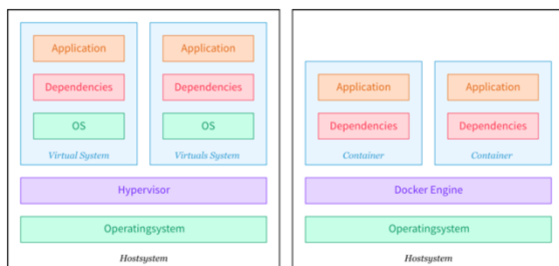

Fig. 2. Classical virtualization vs. container virtualization

The most widely used software for container virtualization is Docker. Docker container can be seen as light-weight equivalent to traditional virtual machines with the great advantage, that containers share a common operating system kernel, making them less resource-intensiv and easily deployable. A simple application container includes an application, all its dependencies and a runtime environment. Such a docker container is derived from a Docker image, which acts as blueprint for the creation of homogeneous application containers. Images are created by the Docker engine by executing the instructions of a Dockerfile.

Due to all the above listed properties and benefits, the software platform is built on container virtualization. The technology not only solves the design requirements, but also enables an easy way of deploying applications in different environments, which can be used with sufficient ease even by students with little previous experience.

4 Platform components

This section introduces the implemented platform components. Unlike the accompanying software tools, platform components are referred to as self-contained application containers and, thereby, integral parts of the platform.

4.1 Graph databases

As the software platform serves the purpose of exploring and utilizing ontologies, graph databases are key components. Their main purpose is to store the ontologies and provide access mechanisms to the information stored in these semantical models.

Different graph databases have varying characteristics. They vary in graph models, query languages, interfaces and more. Many of them also provide a graphical user interface for easier administration, data operations and most important: graphical graph analysis capabilities.

The software platform allows students to decide, which technology and standards they want to use to accomplish the task. To provide a wide range of graph models, interfaces and query languages, the software platform implements multiple graph databases to integrate their specific characteristics.

4.1.1 Graph formats

The accepted graph format relies on the import and export format capabilities of the implemented graph databases. To increase compatibility and interoperability between different systems and application, the software platform must support a wide range of graph formats. To further increase the compatibility and interoperability, the platform incorporates additional components that provide format conversion services. Therefore, the goal is to support most existing graph formats either directly or by an additional format conversion step.

4.1.2 Graph models

The software platform supports the two most commonly used graph models RDF (Resource Description Framework) and LPG (Labeled Property Graph). By supporting both graph models, the software platform does allow for the students to model and query their data in different ways and to explore different approaches and techniques [7].

4.1.3 Query languages

The software platform enables the students to utilize different graph query languages for different graph models.

SPARQL query support was a mandatory design requirement as it is a widely used W3C standard. SPARQL is a graph-based query language and protocol for querying RDF models. SPARQL, as query language for graph databases, can be seen as the equivalent of SQL for relational databases. It allows students to retrieve and modify information stored in triple stores. SPARQL is able to query data across multiple data sources, whether the data is stored natively in RDF or on any database that can be represented as RDF via a middleware [6].

However, SPARQL is not the only graph query language for expressing queries against RDF-based graph models. Additionally, as already stated, RDF is not the only graph model. The software platform initially supports queries expressed in SPARQL, SPARQL-star, openCypher and GraphQL, depending on the graph database being used. The number of supported query languages and protocols will increase, as more graph databases will be added in the future.

The support of different query languages is important, as it enables the students to research different retrieval technics, protocols and the corresponding graph models.

4.1.4 User Interfaces

The role of graphical user interfaces not only covers administrational operations, but also visual graph analytic functions. To understand the often complex structure of semantical knowledge graphs, it is important to get graphical representations, filter functions and more.

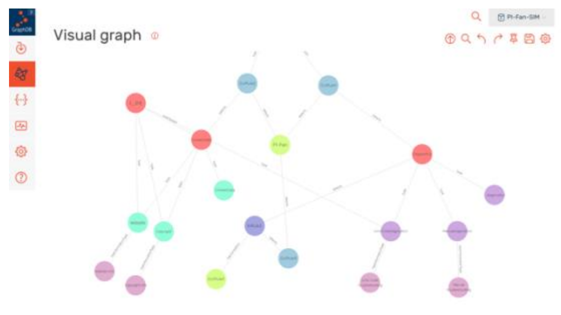

Fig. 3. Visual graph analytic functions

The software platform includes various graph visualizations and analytic functions provided by the different graph databases implemented and, thereby, helping the students to gain an insight of the graph structure.

4.2 Graph format conversion service

The software platform also includes an RDF format conversion service component, allowing the conversion of different graph formats. The service is implemented as a web application, providing an easy-to-use graphical user interface for copying and pasting different graph formats.

4.3 Interactive API documentation viewer

The platform includes an OpenAPI specification (OAS) viewer implemented as web application that accepts an OAS file and generates an interactive API documentation on the fly.

4.4 Demo components

As already stated, the platform consists of preconfigured component, as well as components that need to be developed and integrated by the students. REST APIs are highly use-case and ontology model dependent and, therefore, need to be implemented by the students. The same is true for consuming software applications.

For these cases, the software platform includes demo components. Their purpose is to demonstrate, how such applications can be developed and integrated in the software platform or consuming applications. The demo components include REST APIs written in JavaScript and Python, as well as a simple web application demonstrating the consumption of these REST APIs.

4.5 Workflow and platform documentation

The software platform includes a documentation platform based on an open-source web content management system. The documentation platform describes the platform properties and outlines a workflow for the use of the software platform.

The reason for choosing a web CMS as a foundation for the platform documentation lies in the nature of the software platform itself. As the platform is designed flexible and expandable, its capabilities and properties will change over time. The documentation is intended to grow with the software platform. Utilizing a web CMS enables the students to adapt the documentation as the software platform develops over time.

The documentation platform is implemented in the software platform in form of a dockized web application.

5 Additional software and scripts

The software platform accompanying the software includes an ontology modelling editor, an OpenAPI specification editor, another graph format converter and source code editors. Any additional software applications chosen are either open-source and/or free of charge, making it easy for students utilizing the tools.

The software platform also includes an example ontology in different graph formats and OpenAPI specification files for all included REST APIs.

Furthermore, Dockerfiles and Docker compose files for transferring web applications into docker images are included as well.

6 REST API development

The software platform is intended not only to enable researching semantical knowledge graphs and surrounding technologies, but also to enable incorporating graph knowledge into new or existing applications and systems. Therefore, the support of

the development and implementation of custom REST APIs is another objective of this software platform.

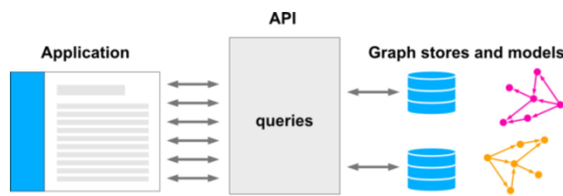

Fig. 4. REST API as middleware between the semantic models stored in graph databases and consuming applications.

The APIs will act as middleware between the graph databases on the one side and applications like CDPs on the other side, by allowing the applications to request resources from API endpoints in a standardized manner, without the need to directly querying the graph databases [8].

6.1.1 Development principles and tool support

REST APIs offer developers a reliable and standardized access to different types of resources, such as files, source code (code on demand) or, as in this case, the results of database queries. The API server exposes (parameterized) endpoints that respond to requests by transferring the requested resources. Because REST APIs can become large and complex, it is important to support the developers using the API by providing comprehensive API documentation, also known as an API contract.

There are two approaches to developing REST APIs: code-first and design-first (contract-first). While the code-first approach might seem like the obvious choice – the developer develops the API and then documents it – there are many advantages to the design-first approach. Using the design-first approach, all properties of the API are formally described before the actual programming. Since the API is formally (and machine-readably) described in advance, a variety of software tools can be used in the subsequent development process.

A widely used and supported standard for describing REST APIs is the OpenAPI specification. The standard is now implemented in many different development environments, programming libraries and software tools and enables diverse support in the development of REST API servers, client application code, interactive software documentation and much more.

By providing software tools and platform components that support the OpenAPI specification, the software platform encourages and supports a design-first approach in REST API development.

6.1.2 Programming languages and frameworks

As many students have a different academic background and varying programming skills, the software platform supports the development of REST-APIs in many different programming languages. Once an OpenAPI specification was developed, API server code can be auto generated in a wide range of programming languages and corresponding application frameworks. While the generated code is not ready to use – the students will have to implement additional business logic – code generation, nevertheless, tremendously reduces the overall development expenses.

6.1.3 Interactive API documentation

An interactive API documentation is a web application that displays all aspects of the API. It describes all API endpoints, all possible request parameters, and the response objects an API call will return. Furthermore, it enables developers to interactively test each API endpoint without writing any client code. This helps developers to quickly understand the structure and behavior of the API.

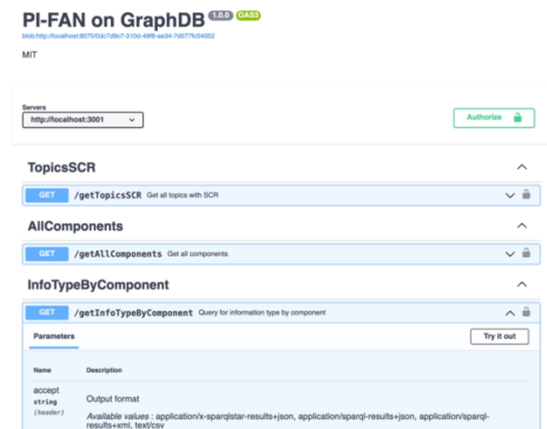

Fig. 5. Interactive API documentation

As the API contract is already developed, an interactive API documentation can be auto generated without any additional effort by the students. The software platform includes components to generate such an interactive documentation upon request.

6.1.4 Implementation and deployment

The fully developed REST APIs, including their corresponding interactive documentation, will have to become a platform component on their own. As the entire software platform runs microservice-oriented on docker, new components will be integrated and deployed as docker application containers as well.

For this purpose, the software platform includes preconfigured Dockerfiles that turn web application based on Javascript/Node.js and Python/Flask (more

technology stacks can be added in the future) into Docker images

7 Summary

In this article, we introduced a software platform for University research on semantical knowledge graphs. The platform meets the requirements that are necessary to support academic studies and explorative prototyping in these areas in the upcoming years.

The software platform provides a broad variety of technologies, like different graph databases, graph models, query languages and more. It supports the development of REST APIs and client applications in a wide range of programming languages. The platform allows and enables the students to choose, which technology they want to explore and utilize in their research projects.

Furthermore, the software platform does not only allow for the research on semantical knowledge graphs and derived applications, but also demonstrate how microservice architectures are designed and can be deployed in modern information technology environments.

Instead of pursuing a monolithic approach, the software platform is designed as a microservice-based infrastructure consisting of independent application container. This enables the software platform to be deployed in various expansion stages in different environments, whether it being a local server, a client personal computer or cloud environments. This also ensures, that the software platform will be able to grow with future demands and to adapt to upcoming use cases.

The software platform accompanying software tools are either open-source and/or free of charge, making it easy for students utilizing the tools.

In addition, the software platform also contains comprehensive workflow and platform documentation, which enables quick familiarization with the subject matter.

8 References

1. W. Ziegler: Extending intelligent content delivery in technical communication by semantics: microdocuments and content services. Proceedings of the ETLTC ACM Chapter International Conference. Volume 77. Aizuwakamatsu, Japan. (2020)
2. C. Deschner, W. Ziegler: Informationsmanagement und Ontologien – entwicklungsnahe Produktdokumentation in der Medizintechnik. Tagungsband zur tekom Jahrestagung. (2018)
3. C. Deschner: Enhanced model-based engineering for centrally managed configuration management in product lifecycle management. Proceedings of the ETLTC ACM Chapter International Conference. Volume 77. Aizuwakamatsu, Japan. (2020)
4. U. Parson, J. Saparo, W. Zieger: iiRDS for Technical Writers – Introduction to the Metadata. Tagungsband zur tekom-Jahrestagung. (2017)
5. W. Ziegler: Semantic Correlation Rules as a Logic Layer between Content Management and Content Delivery. Proceedings of the ETLTC ACM Chapter International Conference. Volume 101. Aizuwakamatsu, Japan. (2021)
6. E. Prud'hommeaux, A. Seaborne: SPARQL Query Language for RDF. https://www.w3.org/TR/rdf-sparql-query/ (2008)
7. J. Barrasa: RDF Triple Stores vs. Labeled Property Graphs: What’s the Difference?. https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/ (2018)
8. R. Fielding: Architectural Styles and the Design of Network-based Software Architectures. Irvine, CA: University of California (2000)