OpenRiskNet

RISK ASSESSMENT E-INFRASTRUCTURE

Deliverable Report D2.4

Final API available for internal and external service providers

This project is funded by
the European Union

OpenRiskNet: Open e-Infrastructure to Support Data Sharing, Knowledge Integration and *in silico* Analysis and Modelling in Risk Assessment

Project Number 731075

www.openrisknet.org

Project identification

Grant Agreement	731075
Project Name	OpenRiskNet: Open e-Infrastructure to Support Data Sharing, Knowledge Integration and *in silico* Analysis and Modelling in Risk Assessment
Project Acronym	OpenRiskNet
Project Coordinator	Douglas Connect GmbH
Star date	1 December 2016
End date	30 November 2019
Duration	36 Months
Project Partners	P1 Douglas Connect GmbH Switzerland (DC) P2 Johannes Gutenberg-Universität Mainz, Germany (JGU) P3 Fundacio Centre De Regulacio Genomica, Spain (CRG) P4 Universiteit Maastricht, Netherlands (UM) P5 The University Of Birmingham, United Kingdom (UoB) P6 National Technical University Of Athens, Greece (NTUA) P7 Fraunhofer Gesellschaft Zur Foerderung Der Angewandten Forschung E.V., Germany (Fraunhofer) P8 Uppsala Universitet, Sweden (UU) P9 Medizinische Universität Innsbruck, Austria (MUI) P10 Informatics Matters Limited, United Kingdom (IM) P11 Institut National De L'environnement Et Des Risques, France (INERIS) P12 Vrije Universiteit Amsterdam, Netherlands (VU)

Deliverable Report identification

Document ID and title	Deliverable 2.4 Final API available for internal and external service providers
Deliverable Type	Demonstrator
Dissemination Level	Public (PU)
Work Package	WP2
Task(s)	Task 2.2
Deliverable lead partner	DC
Author(s)	Daniel Bachler (DC), Joh Dokler (DC), Tim Dudgeon (IM), Egon Willighagen (UM), Pantelis Karatzas (NTUA), Iseult Lynch (UoB), Thomas Exner (DC) Reviewed by Barry Hardy (DC)
Status	Final
Version	V1.0
Document history	2018-11-20 First draft 2018-11-29 Consolidated draft 2019-01-07 Final version

Table of Contents

SUMMARY

This document reports the work on the final API specification for semantic interoperability that was developed as part of the OpenRiskNet e-infrastructure. It briefly outlines the challenges encountered and the solution that has been implemented and is now in use in OpenRiskNet.

This deliverable is related to Task 2.2 (API specification and semantic interoperability) and in a continuation of the work performed within Deliverable 2.2 (Initial API version provided to providers of services) [1], that are now in the process of being incorporated into OpenRiskNet as part of the service catalogue. Deliverable 2.2 gives in-depth information on the evaluation of the various technologies for describing APIs we considered, whereas this report focuses on the solution that was finally chosen and put in place, and the justification of this choice to support other databases / e-infrastructures in their deliberations on API solutions.

The main requirement of describing operations of an API, on both the semantic and technical (structural) levels, was met by creating a novel fusion of the well established OpenAPI standard and the tools of linked data using JSON-LD. This combination allowed us to use and reuse the already existing OpenAPI definitions (and already available OpenAPI tooling), and combine it, with relatively little effort for a service provider, with semantic annotations to facilitate interoperability. The OpenRiskNet Service Registry was developed to scan for such “OpenRiskNet annotated OpenAPI definitions” of active services running in an OpenShift cluster and indexes the data to provide semantic queries using the well established SPARQL linked data query language.

INTRODUCTION

The OpenRiskNet Consortium develops the OpenRiskNet e-infrastructure for harmonisation and improved interoperability of data and software tools in the area of predictive toxicology and risk assessment. It aims to combine interoperable web services providing data or analysis, processing and modelling tools communicating over well-defined and harmonised application programming interfaces (APIs), supplemented by a semantic interoperability layer added to every service to describe the functionality whilst guaranteeing the technical and semantic interoperability.

In computer programming, an API is a set of clearly defined methods of communication between various software components [2]. More precisely, it is a set of subroutine definitions, protocols, and tools for building application software. A good API makes it easier to develop computer programs by providing specifications for all the software building blocks. An API specification can take many forms, depending on whether it applies to a web-based tool, an operating system, database, computer hardware or software library. For examples of APIs, see the C++ Standard Template Library [3], POSIX [4], or Microsoft Windows API [5]. Previous projects like OpenTox, Open PHACTS, and eNanoMapper created a set of APIs, which were designed for the specific problems at hand like creating, validating and applying a QSAR model or bringing together pharmacological data resources. This were reviewed as a starting point for the OpenRiskNet developments, with that analysis reported in Deliverable D2.2 [1]. Of course, most life science databases nowadays provide APIs in addition to download sites too, such as PubChem, EPA CompTox Dashboard, or the ECHA Nanomaterial Observatory. This is supported by efforts such as the European ELIXIR where APIs are the cornerstone of Interoperability.

However, OpenRiskNet goals impose additional requirements as already envisioned and foreseen in the use cases developed (see below) or under development for testing specific parts of the infrastructure. The larger case studies (described in more detail in the Deliverable D1.3 report [6]) for validating the complete infrastructure, especially the much broader scope covering all areas of predictive toxicology and risk assessment as well as the semantic annotation of the APIs needed for the interoperability layer, also separates OpenRiskNet from these earlier projects and makes the development of new concepts inevitable. Taking all these points into consideration led us to the fusion of two as-yet largely unrelated technologies: OpenAPI [7] and JSON-LD [8], which was already outlined as a concept in Deliverable D2.2 report on the initial API version. The full implementation with illustrating examples are presented here.

THE OPENRISKNET API ANNOTATION

An overview of the process

At the beginning of the OpenRiskNet project, the partners started by collecting and inspecting existing web APIs, both of their own tools but also of third party offerings relevant to the field of risk assessment. We then assessed the available tools for two different aspects of documentation that we saw were required.

On one hand, we saw the need to describe the semantic meaning of the various inputs and outputs (i.e. to answer questions like “what semantic concept does this field ‘pcid’ represent?”). The solution adopted by OpenRiskNet is to use established technologies from the semantic web field, especially Resource Description Framework (RDF) and existing ontologies that are in wide use to describe concepts in scientific fields like biology and chemistry or are specifically developed for toxicology.

On the other hand, our assessment included a technical description of the many input, output and invocation mechanisms of an API that would be both human and machine readable. Here the field was much wider and we looked especially at offerings that might already bridge the technical specification with some form of semantic annotation.

We evaluated a wide array of implementations for both fields, the evaluation of which can be found in detail in the Deliverable D2.2 report [1]. In the following section the technologies we selected will be described in more detail, along with the justification for their selection.

Semantic annotation

For semantic annotation, the solution space is relatively narrow and most solutions for expressing semantic relationships revolve around the RDF data model. RDF describes relationships as triples (subject, predicate, object) and uses Uniform Resource Identifiers/Internationalized Resource Identifier (IRIs) as identifiers (where IRIs commonly point to ontology terms). Since the RDF data model offers various serialisation formats, the main question was which serialisation format to use. To enable a seamless integration with the JSON based OpenAPI format, JSON-LD was chosen. In brief, JSON-LD is a way of representing a flat list of linked triples (the RDF data model) using the logical tree data structure of a JSON document.

Technical (structural) annotation

On the technical side, the situation was more complicated. A major obstacle the OpenRiskNet Consortium faced is that Web APIs are very flexible and allow for a wide variety of specifying input and output data. For example, input parameters can be transmitted in the HTTP header, as part of the request string in the URI, or as a body

payload using a large number of serialisation formats. All of these are in active use in the community and the precise combination of encodings for inputs and outputs is chosen by the API authors according to various technical constraints and preferences. Describing a Web API in enough detail that both a human user and a machine have all the information required to construct requests to this API is a significant undertaking. This fact also meant that it would be counterproductive to design an entirely novel way of describing Web APIs for it would have meant both a huge effort of the OpenRiskNet consortium to create such a description format but also a significant learning curve and effort for Service Providers who would want to annotate their services in an OpenRiskNet-compliant way. Furthermore, there would have been no existing tooling in the form of API description syntax highlighting, verification tools etc., meaning that these would also have to be developed bespoke by the OpenRiskNet partners, again at considerable time and resource cost.

OpenAPI

After a long process of weighing pros and cons, we decided to use the well-established OpenAPI specification as our basis. OpenAPI (formerly known as Swagger) is a big effort to describe Web APIs in a way that is both human and machine readable and that is widely used already. This has the benefit that many developers will already be familiar with the majority of our description format (and often even have such OpenAPI definitions for their services), and that a substantial tooling already exists that aids developers in writing such definitions, verifying them and even generating implementation code for them in some cases. Also, OpenAPI specifications implement the best practices in API development. If these are adopted correctly, a harmonised communication pattern arises through APIs and service interoperability becomes an easier task. OpenAPI does not, however, provide any way of specifying semantic annotations.

To add semantic annotations, we wanted to find a way to annotate OpenAPI documents that would a) be able to be written inline, i.e. not create a new, separate document that has to be kept in sync with the OpenAPI description and b) keep the OpenAPI document in a form that still adheres to the OpenAPI specifications. The latter point is important so that existing tooling can still parse our newly fused “OpenRiskNet annotated OpenAPI documents” even after the semantic annotations had been applied. JSON-LD provides this capability and was thus selected.

Fortunately, OpenAPI provides a mechanism to add arbitrary annotations at almost all levels of the description - as long as they are prefixed with “x-”. We thus constructed a minimal header that aliases the standard JSON-LD terms for “@type” and “@id” to use “x-orn-@type” and “x-orn-@id” (where the orn indicated OpenRiskNet), and gives a JSON-LD “@vocab” definition so that all JSON keys that are not specifically mapped to ontology terms will be mapped to a placeholder URI so that they will not be dropped in the conversion step into RDF triples. The resulting JSON-LD can be checked for validity through online platforms such as Json-ld playground .

OpenRiskNet annotations for OpenAPI

To summarize, this means that with the addition of the following header, an existing OpenAPI description can be turned into a basic “OpenRiskNet annotated OpenAPI document” and be interpreted as linked data (given in YAML serialisation below):

```
'x-orn-@context':
  '@vocab': 'http://openrisknet.org/schema#'
  x-orn: 'http://openrisknet.org/schema#'
  x-orn-@id: '@id'
  x-orn-@type: '@type'
```

With this in place, API authors can then annotate input and output parameters (like keys in their JSON payload, query parameters etc.) by adding additional mappings to ontology terms in the 'x-orn-@context' section. For example, to map the terms "smiles" to the CHEMINF ontology term for SMILES (CHEMINF_000018) and the term "inchi" to the corresponding term for InChI (CHEMINF_00011) [9], the following two lines would be added to the section above and be included in the OpenAPI document for the API:

```
smiles: http://semanticscience.org/resource/CHEMINF_000018
inchi: http://semanticscience.org/resource/CHEMINF_00011
```

With this information added to the OpenAPI document, the service providers can supply an OpenRiskNet (semantically) annotated OpenAPI document that can then be parsed and indexed by the OpenRiskNet Service Registry. An example to clarify this will follow in the next section.

To guide service providers through the process of annotating their OpenAPI descriptions with semantic annotations and create a valid "OpenRiskNet annotated OpenAPI document", public documentation [10]has been published on the OpenRiskNet website. Additionally, an interactive tool called the OpenRiskNet Query Tester [11] was created to quickly test such annotated OpenAPI documents.

Comparison with existing approaches

As outlined in the D2.2 deliverable, there are only few existing attempts to bridge the two worlds of, on one hand, technical HTTP web API descriptions that are both computer and human readable and, on the other hand, semantic descriptions of operations and their inputs and outputs. The most visible of such attempt was the HYDRA project [12] that tried to build a technical description purely as a JSON-LD document. This allows for a very elegant semantic description of parts of an API, but creating such a description in enough detail that requests to an API can be constructed automatically is a very laborious effort and the tooling support is basically non-existent. Furthermore, while a lot of existing Web APIs are already documented using OpenAPI, HYDRA is not used by any of the consortium or associated partners and to our knowledge by hardly any Web APIs at all. Adopting this concept would therefore impose large development work on service providers and thus limit the number of services, which can be integrated by the consortium partners and supported by the implementation challenge.

Another more recent attempt is the approach taken by Schwichtenberg et al. [13], where an ad-hoc ontology is derived automatically from an existing OpenAPI document that is then aligned with global ontologies by the user. The downside of such an approach is that in this case the semantic operation is extracted into a different data model and exists as a separate document. We believe that our approach of integrating the semantic mapping

into the OpenAPI document itself but in a way that still allows ordinary OpenAPI tools to operate unhindered is a significant advantage.

Indexing and search using the OpenRiskNet Service Registry

To fully understand what happens with these annotated descriptions in the OpenRiskNet Service Registry, it is important to be aware of how JSON-LD bridges JSON and Linked data. As was illustrated above, the basic premise of JSON-LD is that the tree-like structure of JSON can be expressed as a series of triples where each “level” in the tree is a source or object node in the triple and the object key is seen as the predicate. This allows an entire JSON document to be transcribed into a flat list of triples that mimic the same structure. The OpenRiskNet Service Registry takes the OpenAPI JSON with the OpenRiskNet annotations and ingests it as a JSON-LD into a triple store. Since an OpenAPI document has a lot of keys and nodes that are of no semantic interest (they are just mandated for the full technical description that OpenAPI provides), we accept that most nodes will be anonymous and most keys will be resolved to placeholder URIs. The important part however, is that for those terms that were mapped to ontology terms with the above method, the full ontology URI will be used (if done like above, this will result in the mapping of a *predicate* but a mapping with x-orn-@id for nodes is also possible).

Based on triples that are generated for every service, a SPARQL query engine can then be used to find e.g. all services that take SMILES (identified not by some common spelling but by the precise ontology term CHEMINF_000018) as an input.

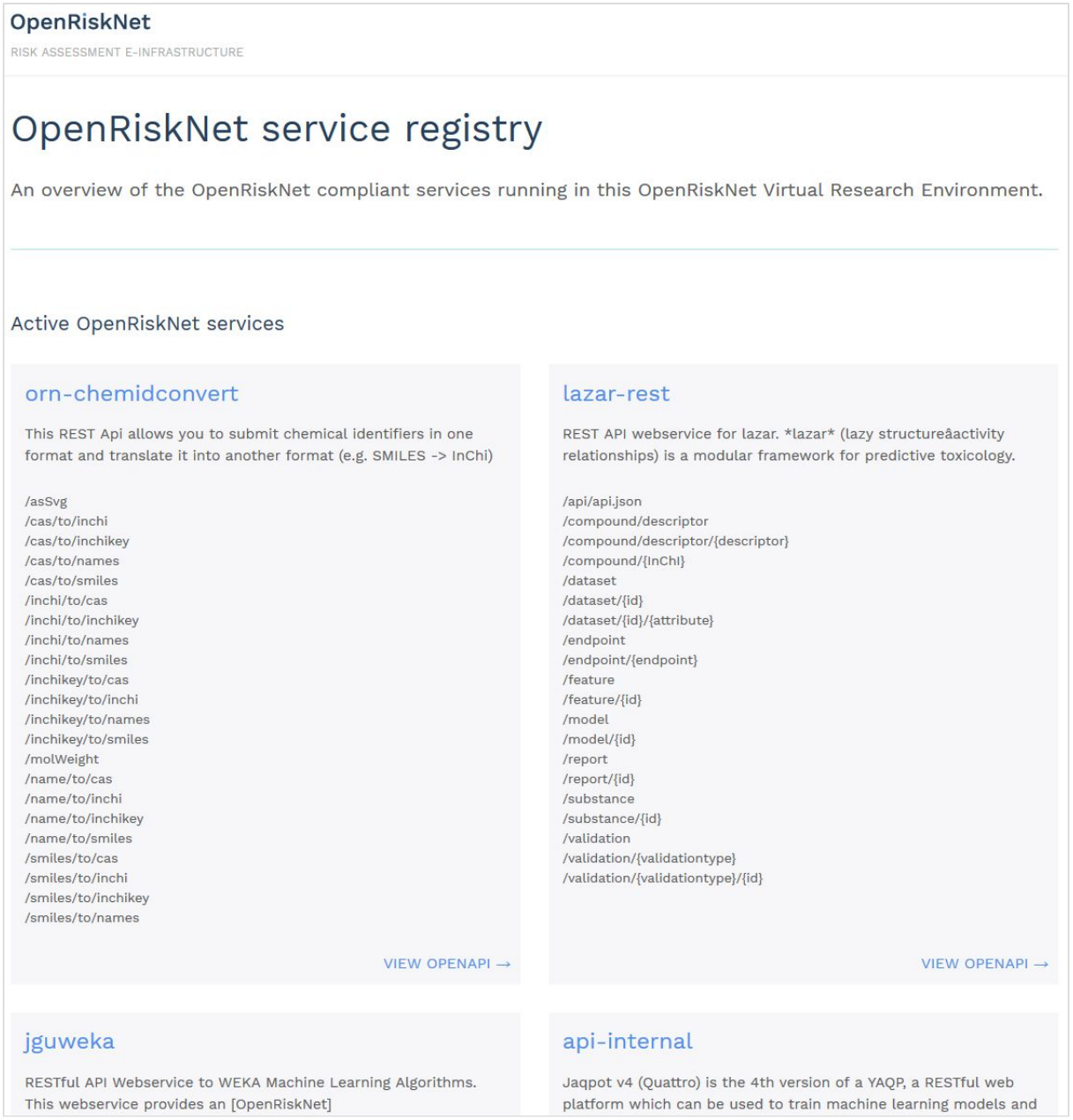

Figure 1. Screenshot of the OpenRiskNet service registry application

Examples

To clarify how these annotations are used, we show a brief excerpt of the triples that are created from the OpenAPI document that has been annotated in an OpenRiskNet compliant way. All of these examples stem from the default example of the Lazar API and can be tested by anyone at the public OpenRiskNet Query Tester [11] that was built to evaluate the feasibility of the approach described in this document.

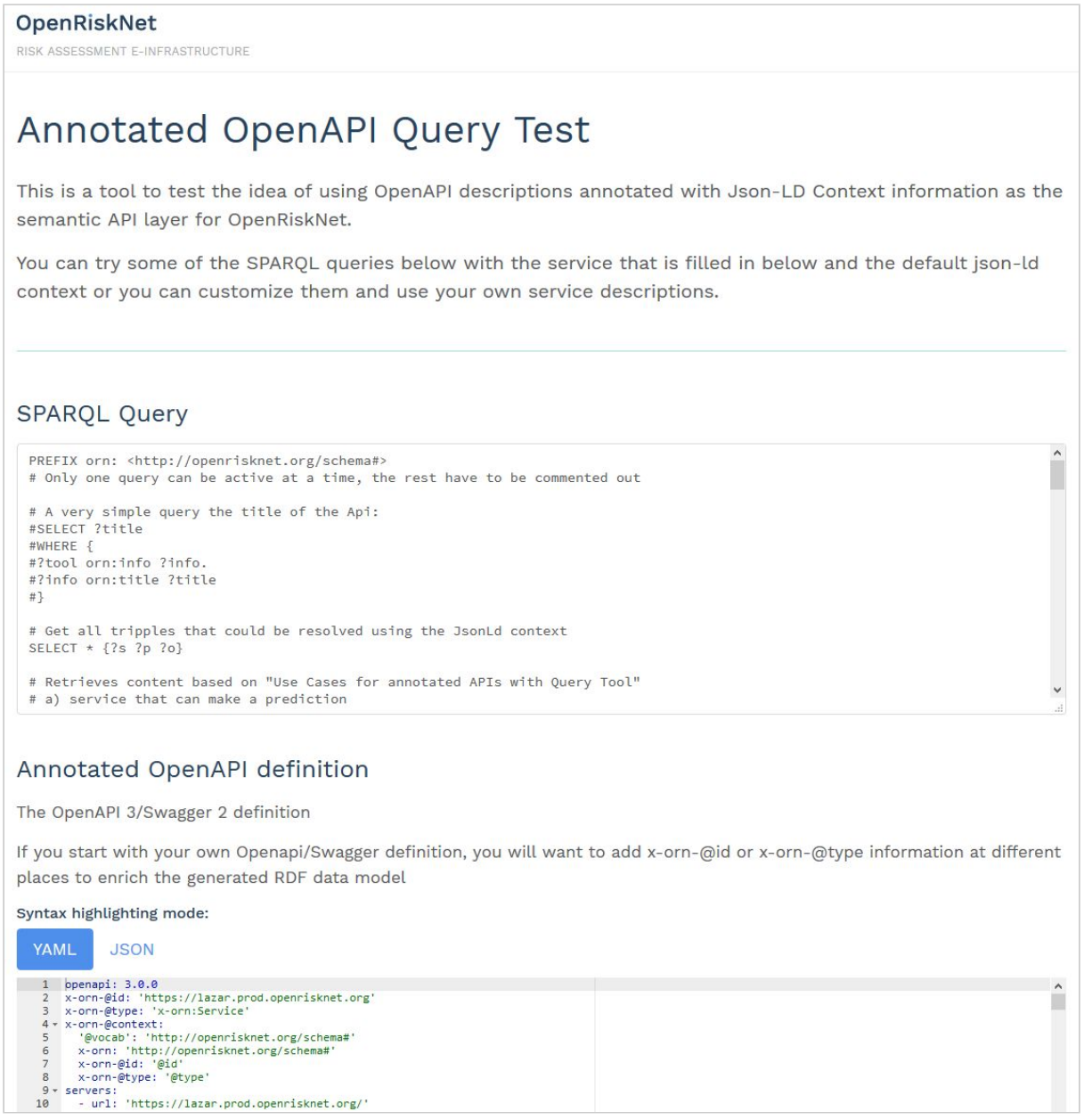

Figure 2. The OpenRiskNet Query Tester tool

Transcription of OpenAPI keys without semantic annotation

A typical OpenApi JSON document describes the entire surface of a web API and is thus often of significant size. Much of the information encoded herein is usually not of semantic interest (e.g. the document starts with an Info section that contains fields like the terms of service). By configuring the `@vocab` instruction in the JSON-LD context above, we ensure that even JSON keys that are not given a semantic meaning are kept when transcribing to triples. Thus, a fragment of the info section of the example Lazar API looking like this:

```
info:
  title: Lazar REST Service
  version: 1.4.0
```

is transcribed to anonymous RDF triples as given in Table 1.

Table 1. Resulting triples transcription of the example info section

Subject	**Predicate**	**Object**
_:b38	http://openrisknet.org/schema#title	“Lazar REST Service”^^xsd:string
_:b38	http://openrisknet.org/schema#version	“1.4.0”^^xsd:string

This may not look particularly useful and indeed, if this is all that could be done, the usefulness would be rather limited even if it does already allow us to query such triples with the SPARQL query language.

The real power of this approach comes when the API authors add semantic mappings of JSON keys to ontology terms like above with SMILES and InChI as is shown in the next example.

Transcription of OpenAPI keys with semantic annotation

Whereas in the above example none of the JSON keys like “title” or “version” were mapped to ontology terms, an interesting thing happens when such a mapping is performed. Here we look at a parameter of the Lazar API with the key of “inchi” defined:

```
parameters:
  inchi:
    name: InChI
    in: path
    description: InChI String
    required: true
    schema:
      type: string
```

This part of the OpenAPI is transcribed with the mapping of InChI given above to the triples in Table 2.

Table 2. Resulting triples of the InChI mapping

Subject	**Predicate**	**Object**
_:b1	http://semanticscience.org/resource/CHEMINF_000113	_:b10
_:b10	http://openrisknet.org/schema#name	InChI^^xsd:string
_:b10	http://openrisknet.org/schema#in	path^^xsd:string
_:b10	http://openrisknet.org/schema#description	InChI String^^xsd:#tring
_:b10	http://openrisknet.org/schema#required	true^^xsd:boolean
_:b10	http://openrisknet.org/schema#schema	_:b11
_:b11	http://openrisknet.org/schema#type	string^^xsd:string

Because the predicate in the first triple is now a well-defined ontology term, SPARQL queries can be constructed that find the semantic construct that is represented by an InChI across APIs and at arbitrary locations in the OpenAPI description. Because this is using ontology terms, additional reasoning can be performed. For example, not just API endpoints that process exactly “InChI” identifiers can be found, but instead the more general term in the CHEMINF ontology for “structural descriptor” CHEMINF:000085 can be used to find all API Endpoints that process any kind of “structural descriptor” defined by the CHEMINF ontology and OpenRiskNet identifier mapping services can be used to map between different identifiers to be able to e.g. combine datasets even if the same identifiers are not available in both.

Example SPARQL Queries

To complete the examples, we show here how the information from **Table 2** above can be used in a SPARQL Query. We currently do not foresee normal users of the OpenRiskNet system writing such queries themselves. Instead OpenRiskNet components like the Service Registry would use similar queries to perform semantic queries on behalf of the user.

In this example the triple store is queried for triples that describe a chain using the orn:info and then the orn:title predicate and only the selected title value is returned.

```
    # A very simple query the title of the API:
```

```
SELECT ?title
WHERE {
?tool orn:info ?info.
?info orn:title ?title
}
```

A second query example asks for several descriptors of all endpoints returning a certain mime type of response content.

```
SELECT ?service ?description ?path ?rest ?input ?output WHERE {
# service name
?service a orn:Service.
?service orn:info ?info.
?info orn:description ?description.
# handle SD files
?s <http://openrisknet.org/schema#chemical/x-mdl-sdfile> ?format.
# show REST method, path
?content orn:content ?s.
?200 orn:200 ?content.
?responses orn:responses ?200.
?responses orn:path ?path.
?responses orn:method ?rest.
# show input, output
?format orn:returns ?output.
?format orn:schema ?schema.
?schema orn:property ?input.
}
```

CONCLUSION

The OpenRiskNet consortium successfully created a novel way of providing semantic annotations for Web APIs. The annotation mechanism has been used to annotate existing OpenAPI descriptions of consortium partner's services and the OpenRiskNet Service Registry is built around this specification to scan for, and allow queries against, these annotated service descriptions. Documentation to guide service providers in their effort to annotate existing OpenAPI documents with semantic annotations have been created[1,2], as has the OpenRiskNet Query Tester tool to test such documents interactively before deploying any services.

It is worth noting, that to the best of our knowledge, such a bridge of a technical API description format and a semantic annotation of operations using ontology terms provided by the service providers with very low effort has never been done before. A big part of the work to arrive at such an elegant solution was the creation of the OpenRiskNet Query Test Tool, a web-based tool that allows interactive entry of: a) a plain OpenAPI document, b) a linked data context as given above and c) a SPARQL query to run. This tool allowed us to test various ways of annotating OpenAPI documents with JSON-LD and to verify that the semantic questions we would like to ask against a collection of OpenRiskNet services can be posed and answered using this approach. The approach is being written up for publication in a cheminformatics Journal also, to enable widespread adoption of this approach by the community, as a major achievement of OpenRiskNet.

GLOSSARY

The list of terms or abbreviations with the definitions, used in the context of OpenRiskNet project and the e-infrastructure development is available:

https://github.com/OpenRiskNet/home/wiki/Glossary

[1] https://openrisknet.org/e-infrastructure/development/
[2] https://github.com/OpenRiskNet/home/wiki

REFERENCES

1. Rautenberg M, Karwath A, Kramer S, Dudgeon T, Spjuth O, Bachler D, et al. Initial API version provided to providers of services (Deliverable 2.2). 2018; doi:10.5281/zenodo.1479444

2. Application programming interface - Wikipedia. In: Wikimedia Foundation, Inc. [Internet]. 30 Jul 2001. Available: https://en.wikipedia.org/wiki/Application_programming_interface

3. Containers - C++ Reference [Internet]. Available: http://www.cplusplus.com/reference/stl/

4. The Open Group Base Specifications Issue 7, 2018 edition [Internet]. Available: http://pubs.opengroup.org/onlinepubs/9699919799/

5. windows-sdk-content. Windows API Index [Internet]. Available: https://docs.microsoft.com/en-us/windows/desktop/apiindex/windows-api-list

6. Jennings P, Exner T, Farcal L, Oki N, Sarimveis H, Doganis P, et al. Final definition of case studies (Deliverable 1.3). 2018; doi:10.5281/zenodo.1479127

7. OpenAPI Initiative. In: OpenAPI Initiative [Internet]. Available: https://www.openapis.org/

8. JSON-LD - JSON for Linking Data [Internet]. Available: https://json-ld.org/

9. Hastings J, Chepelev L, Willighagen E, Adams N, Steinbeck C, Dumontier M. The Chemical Information Ontology: Provenance and Disambiguation for Chemical Data on the Biological Semantic Web. PLoS One. Public Library of Science; 2011;6: e25513.

10. Semantic interoperability • OpenRiskNet [Internet]. Available: https://openrisknet.org/e-infrastructure/development/semantic-interoperability/

11. OpenRiskNet OpenAPI Query Test [Internet]. Available: https://orn-query-test.cloud.douglasconnect.com/

12. Hydra: Hypermedia-Driven Web APIs [Internet]. Available: https://www.markus-lanthaler.com/hydra/

13. Schwichtenberg S, Gerth C, Engels G. From Open API to Semantic Specifications and Code Adapters. 2017 IEEE International Conference on Web Services (ICWS). IEEE; 2017. pp. 484–491.