ET-12 01 21 CAE-PA Project Report: HAZOP2RDF

A robust program to handle HAZOP studies

Dmytro Kostiuk
Marcus Rothhaupt
Artan Kabashi
Vincenz Forkel

TECHNISCHE UNIVERSITÄT DRESDEN

21.07.2021

Contents

List of Tables

List of Figures

Abstract

This research paper describes a program that handles HAZOP data from an excel spreadsheet. With the described program an easy to handle back-and-fourth conversion from HAZOP data in excel Format to RDF format is possible. Furthermore a verification of the HAZOP data takes place and can also be configured.

By analysing the needs of the faculty and working together with process control engineers we developed this application.

Our solution features a command line interface to give users an easy access to our software.

Using our Importer interface the user can import and validate incoming HAZOP data in Excel format and generate RDF graphs from it. They can be locally stored or uploaded to a Fuseki server.

Using our Exporter interface the user can export RDF graphs containing HAZOP data to Excel format. The source for the Exporter interface can either be a locally stored RDF file or an RDF file stored on a Fuseki server.

The overall result of our findings is the largely increased compatibility of RDF files over excel files when storing, transforming or manipulating HAZOP studies.

- needs to be more specific
- what was done in this research and why?
- what method was used?

Introduction

Klose et al. (2019) Being able to convert excel spreadsheets to the RDF format is a huge time saver for engineers working with HAZOP studies. By analysing the structure of the HAZOP files we can draw conclusions about the risks of a plant.

In order to do so we started by writing sophisticated code to develop an engine that transforms the lines of HAZOP study excel spreadsheet into an RDF turtle formatted code.

In the following parts of this paper it is shown what our software is capable of, how we solved our problem and how the different commands of our command line interface can be used to ease the process of handling HAZOP studies.

It can be noted that our software can be especially useful when designing a modular plant with multiple HAZOP studies for each part of the plant.

Motivation

Modular plants are getting more and more important in today's industry. They enable flexible production in small quantities and over variable time periods. A modular plant can be built very quickly, because all interfaces are compatible, but a HAZOP analysis of the entire plant is still necessary, which strongly limits the advantages of modularity.

In order to meet high safety standards a hazard and operability study, short HAZOP has to be conducted on all parts of the modular plant.

The HAZOP risk analysis is an important part of every safety concept of industrial plants. This analysis is done manually and individually in Excel. Since it is done in large packed tables it can not be automated or reused. This inefficiency can be remedied by converting the large HAZOP excel tables into an easier and better to handle format.

The Resource Description Framework, short RDF can be such a format. Because of its easy to handle language, reusability and automatability we choose this widely spread format.

Through the usage of a DB Server like Fuseki, the transformed file can be shared and stored very conveniently.

One of the biggest problem we faced was to verify and define a well-formed excel spreadsheet so that our importer was able to generate a correct RDF file. Now the verification process uses a user defined configuration to analyse the excel spreadsheet and import it. The importer had to be able to distinguish between meaningful and useless Hazop-Cases inside the spreadsheet.

Problem Analysis

here, an analysis section would be nice. from your motivation and state of the art, you should derive a list of requirements that your concept later on should solve. e.g.: - HAZOPs need to be stored - HAZOPs need to be machinereadable - ... each requirement should have an argument, either from literature or logically derived

Concept

Program Component Design

We decided to structure our solution into three separate programs, all of which can stand alone, and operate independent of each other; and a Command Line User Interface for easier operability. - The Importer: Takes an Excel HAZOP-analysis file, validates it, converts the table structure into RDF-Triples, and finally stores the created RDF-Triples with all metadata either locally or in the TripleStore. - The Exporter: Takes a HAZOP-analysis in RDF-Triples, extracts information and metadata, and stores them in the industry standard formatting in an Excel file. - The TripleStore: Fuseki server that stores HAZOP data in RDF-Triples. Acting as a central database for machine readable completed HAZOP-Analysis with easy accessibility. Great care was taken to design the programs separate and allow for alternative

and independent use cases with a focus on reusability. Thus, both the Importer and Exporter are capable of reading and storing data locally or through the TripleStore database. Likewise, the TripleStore is useable for multiple purposes and not limited to just the storing of HAZOP-triples.

Figure 1: Program components

User interaction scenario

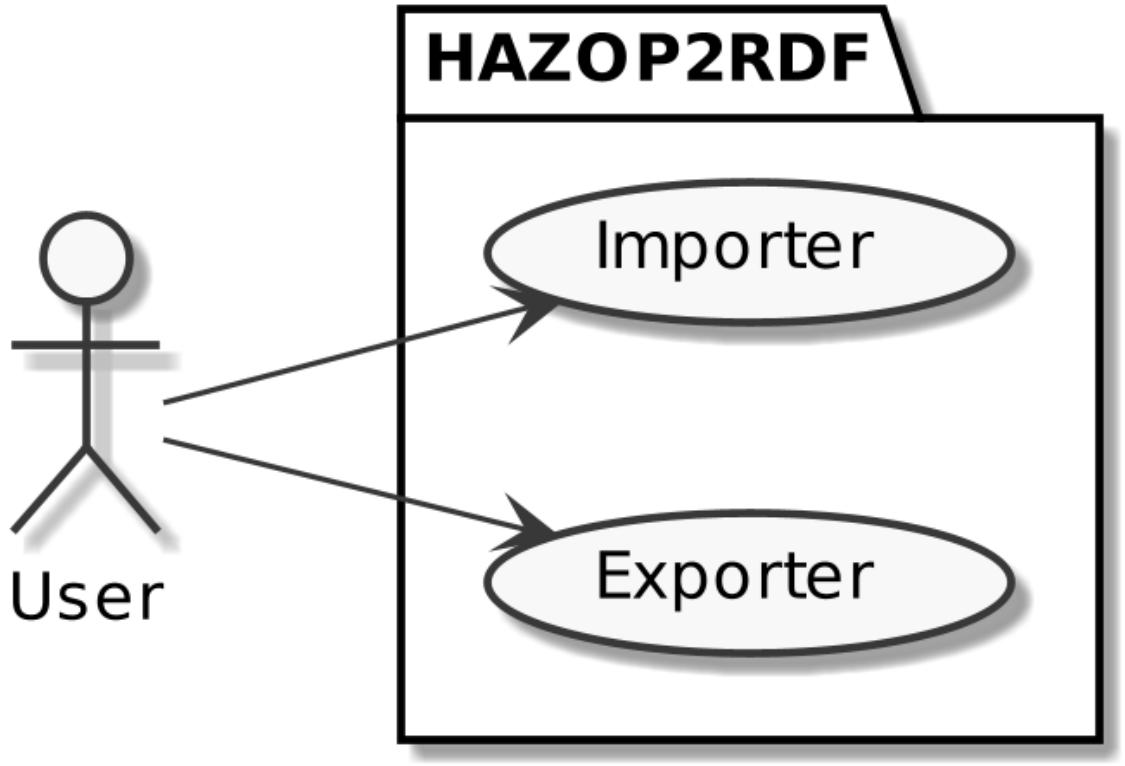

Figure 2: User interaction

Design Command Line Interface

To take full advantage of the modularity of our solution, a Command Line User Interface was chosen. This allows specific access to all functionalities of only the Importer and Exporter programs and eliminates convolution of the separate functions inside each program through precise commands. For security reasons, the user is only able to communicate with the Triplestore indirectly through Importer and Exporter functions. This furthermore ensures that data on the Triplestore will not be corrupted through operator errors.

Figure 3: Design of Command Line Interface

Implementation

We implemented the Command Line Interface using Click[1] package. It is highly configurable and can build very complex applications. The ComplexCLI utility, we used in our project, combines multiple interfaces in a single Command Line Interface.

The following diagram shows the structure of the Command Line Interface. It contains Importer and Exporter interface, which use services. The services contain utilities needed for the interfaces to perform lower level actions.

[1] Python package: Click

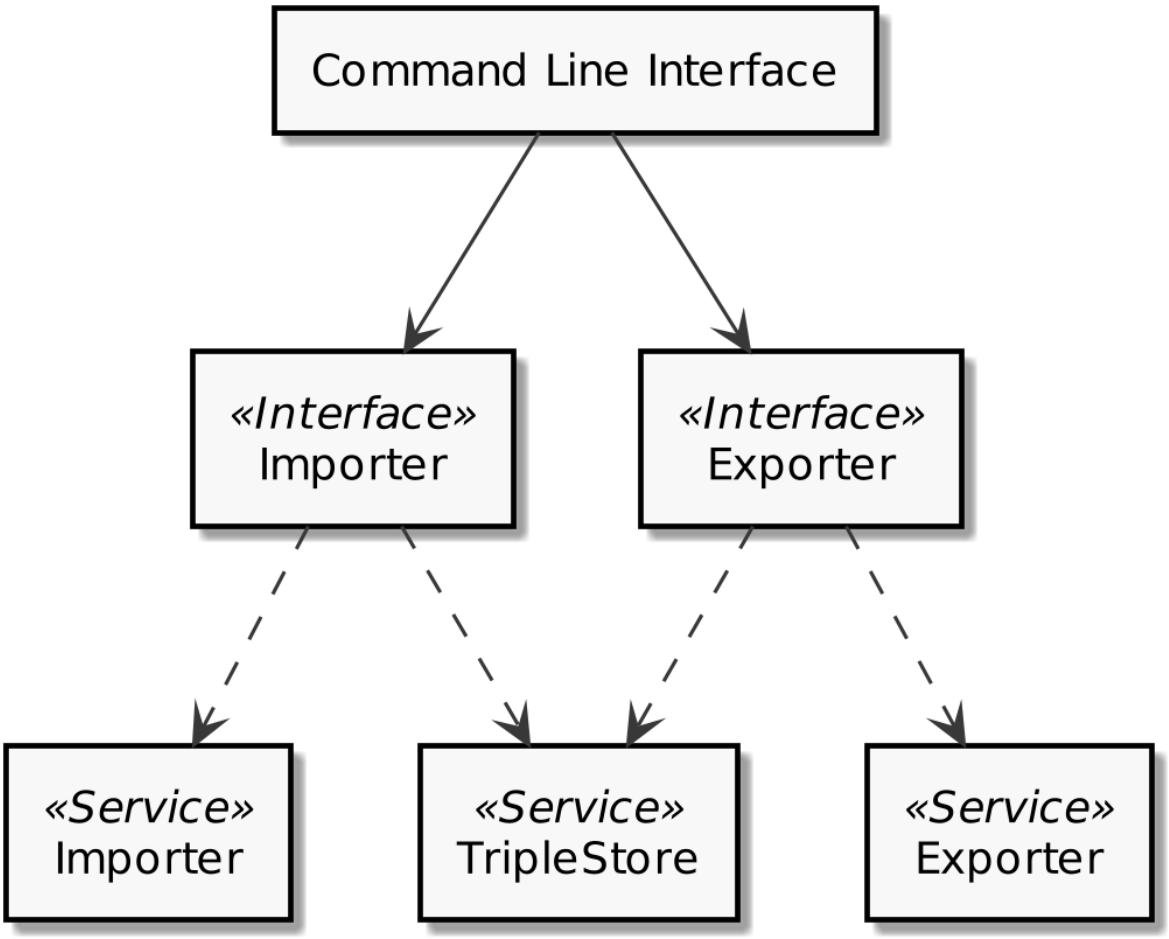

Figure 4: Structure of Command Line Interface

Using the Command Line Interface users can interact with our software.

Using our Importer interface the user can import and validate incoming HAZOP data in Excel format and generate RDF graphs from it. They can be locally stored or uploaded to a Fuseki server.

Using our Exporter interface the user can export RDF graphs containing HAZOP data to Excel format. The source for the Exporter interface can either be a locally stored RDF file or an RDF file stored on a Fuseki server.

Importer interface

The main purpose of the Importer interface is to build an RDF graph from incoming Excel data. To build an RDF graph, we carefully read the incoming HAZOP data and validate it. To validate the data correctly we implemented a config file, which stores all the metadata needed to describe the importing and validating process.

The main command of the Importer interface is cmd-build-hazop-graphs, which reads the HAZOP data stored in a local directory and transforms it to an RDF graph. The graph can be consequentially stored locally or uploaded to a Fuseki server. The two other commands cmd-read-excel-data and cmd-read-hazop-data make it possible for the user to perform steps of the main importer command individually.

The installation of a Fuseki server is optional. If the server is offline, the files cannot be uploaded to the server resulting in an error message which is displayed to the user.

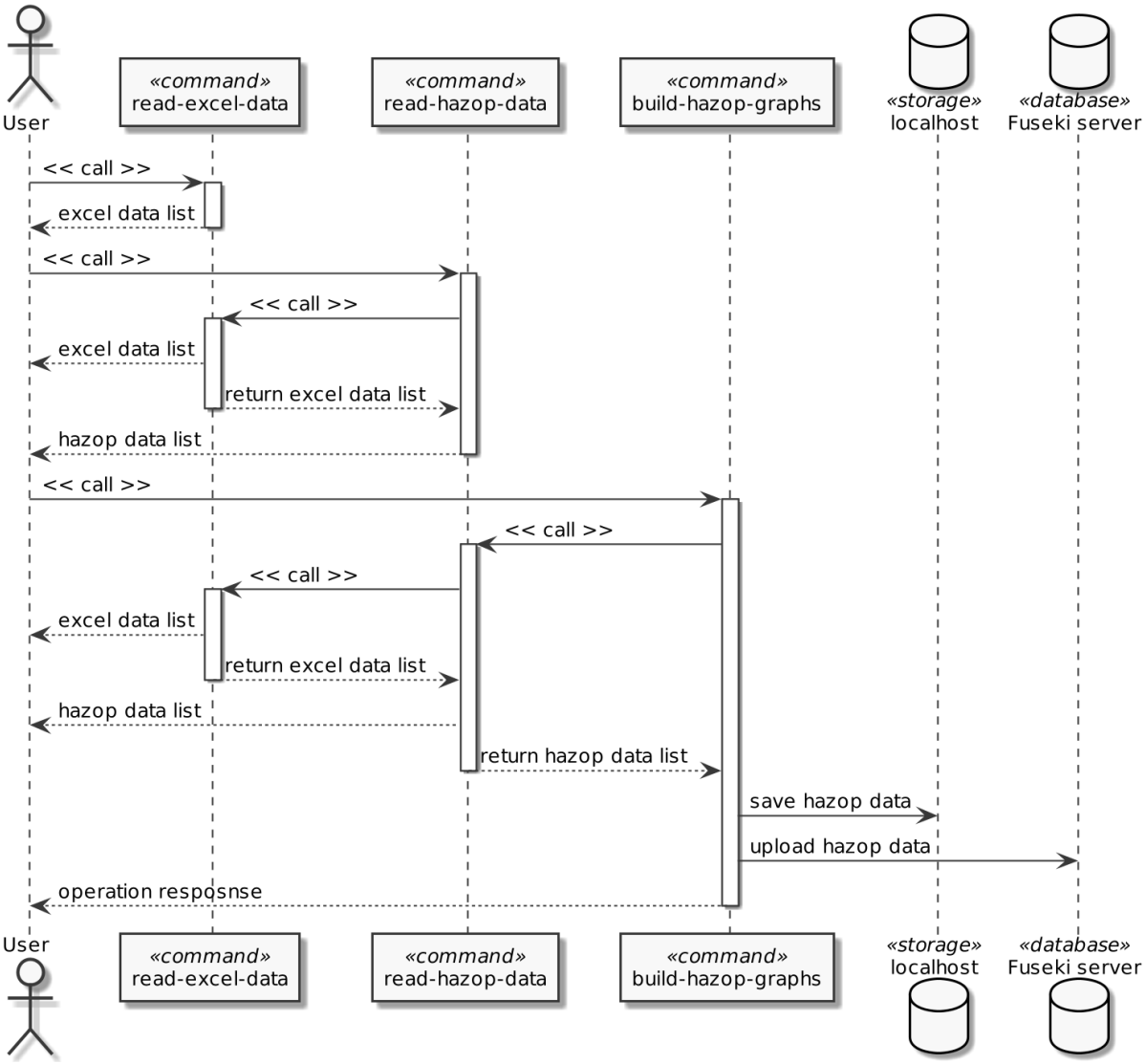

Figure 5: Sequence diagram of Importer interface

Exporter interface

After the HAZOP data was successfully imported and stored, the user can convert the RDF graph to Excel format again.

There are two main commands in the Exporter interface for the users to interact with. The user can either export data from an RDF file located in a local directory or from a file located online on a Fuseki server. For the successful export from the

Fuseki Server the server needs to be running.

As a result, the RDF graphs will be stored locally in Excel format again.

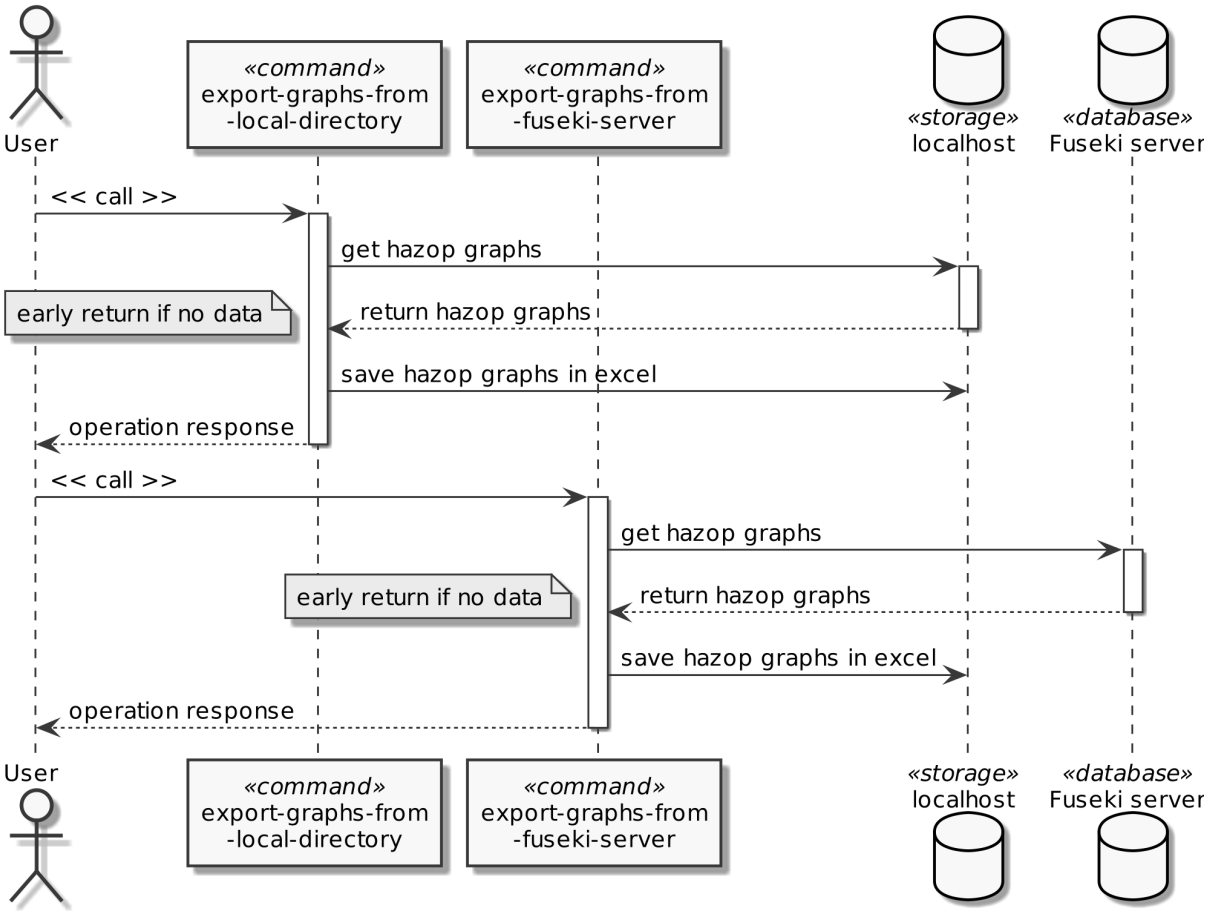

Figure 6: Sequence diagram of Exporter interface

Remarks

We developed the HAZOP2RDF project with version control on GitHub. The program is available for Windows and macOS. We also included a detailed installation guide in the documentation.

Verification

The verification process includes checking of the code with the intent of finding failures. To make the program perform well, it should not contain critical errors and bugs. We designed a test pattern to verify the results of the program execution. This pattern is simple and extendable and covers the main parts of the program.

In the tests' directory of the project you can find a list of the following files:

- test_cli .py - test command line interface initialisation
- test_cmd_importer.py - test importer interface
- test_cmd_exporter.py - test exporter interface

The prime objective of the pattern is consistency. It allows us to apply this pattern to every test we want to implement.

The pattern covers the following test cases:

- execution errors
- execution exceptions
- output verification

The code coverage value varies around 94%. The value depends on the state of the Fuseki server. It increases if the Fuseki server is running and there is pre-uploaded data on the server.

The coverage report below shows the detailed information about the tests results.

Listing 1: Coverage report

```
1  ---------- coverage: platform darwin, python 3.8.2-final-0
       -----------
2  Name                                  Stmts   Miss  Cover   Missing
3  ---------------------------------------------------------------------
4  src/__init__.py                           0      0   100%
5  src/cli.py                               20      3    85%   34-35,
       43
6  src/commands/__init__.py                  0      0   100%
7  src/commands/cmd_exporter.py             50      6    88%   37, 43,
       48-50, 65
8  src/commands/cmd_importer.py             66      2    97%   38, 80
```

```
 9  src/config/__init__.py                0      0   100%
10  src/config/config.py                  3      0   100%
11  src/services/__init__.py              0      0   100%
12  src/services/svc_exporter.py         28      0   100%
13  src/services/svc_importer.py         66      0   100%
14  src/services/svc_triplestore.py      16      3    81%   37-40
15  ------------------------------------------------------------
16  TOTAL                               249     14    94%
```

The user can although generate a coverage report in HTML format and easily discover the missing statements. See Appendix section for HTML coverage report example.

Summary

With increasing complexity of systems, the amount of generated data increases proportionally and easily exceeds the human capabilities for interpretation.

Therefore, we propose an approach for data compression and models for correlations of causes, deviations and consequences.

The result are enriched HAZOP studies that are designed to be conducive for human interpretation and also support machine-readability The HAZOP-data was stored as Resource Description Framework (RDF). An import and export was implemented to support the original documentation of the used HAZOPs

The vision would be to support humans in the decision-making in such a way that, based on HAZOPs from experts, even novices could also decide whether a plant is safe or not. Since safety scenarios are always critical, this aspect needs to be analysed in more detail and is focus of future research

Future Work

The application provides essential functionality for the HAZOP transformation. There is still a room for adoptions, tests and experiments have been left for the future work.

- The import of the HAZOP data can be improved and adopted with fixed constrains. It can be conditional statements, which parse different shapes of the incoming data and serve them for the RDF transformation.
- Fuseki server offers HTTP access. Currently, we use a set of command line scripts[2] to work with SPARQL. This API can be although extended with Requests[3] or SPARQLWrapper[4] package to perform HTTP requests and internally customize them.

Obviously, other ideas and improvements have their place. We focused on the basic functionality, needed to transform the HAZOPs. To provide a richer experience for the client, the application can be extended dependent on the current environmental requirements.

Appendix

- turtle data, output ontology in excel, documentation

Listing 2: HAZOP ontology in Turtle format

```
@prefix blanknode: <http://www.hazop2rdf.de/hazop/blanknode/> .
@prefix hazopcase: <http://www.hazop2rdf.de/hazop/hazopcase/> .
@prefix predicate: <http://www.hazop2rdf.de/hazop/predicate/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

hazopcase:1 blanknode:cause [ predicate:description "Zugeführtes
    Prozessmedium zu heiß (>200°C)" ;
            predicate:guideword "Mehr" ;
            predicate:hazopnode "In 1 - Feed-Eingang" ;
            predicate:parameter "Temperatur" ] ;
    blanknode:consequence [ predicate:description "
       Materialversagen der Dichtungen, Leckage" ;
            predicate:guideword "NaN"^^xsd:double ;
            predicate:hazopnode "Speicherbehälter" ;
            predicate:parameter "NaN"^^xsd:double ] ;
    blanknode:deviation [ predicate:description "Überschreitung
       der zulässigen Temperatur im Behälter" ;
```

[2]Set of scripts: SOH (SPARQL over HTTP)
[3]Python package: Requests
[4]Python package: SPARQLWrapper

```
15              predicate:guideword "Mehr" ;
16              predicate:hazopnode "Speicherbehälter" ;
17              predicate:parameter "Temperatur" ] ;
18      blanknode:restrisiko [ predicate:avoiding "G2 - fast unmö
           glich" ;
19              predicate:presence "A2 - häufig bis andauernd" ;
20              predicate:probability "W2 - gering" ;
21              predicate:severity "S1 - minimale " ] ;
22      blanknode:riskgraph [ predicate:avoiding "G2 - fast unmö
           glich" ;
23              predicate:presence "A2 - häufig bis andauernd" ;
24              predicate:probability "W2 - gering" ;
25              predicate:severity "S2 - geringe" ] ;
26      blanknode:safeguard [ predicate:hazopnode "Speicherbehälter"
            ;
27              predicate:otherinfo "NaN"^^xsd:double ;
28              predicate:parameter "Hochwertige Dichtungen für Temp
                   . über 200°C (bei 25bar)" ;
29              predicate:recommendation "NaN"^^xsd:double ] .
30
31  hazopcase:10 blanknode:cause [ predicate:description "reines Lö
       sungsmittel wird zugeführt" ;
32              predicate:guideword "Kein" ;
33              predicate:hazopnode "In 1 - Feed-Eingang" ;
34              predicate:parameter "Konzentration" ] ;
35      blanknode:consequence [ predicate:description "kein" ;
36              predicate:guideword "NaN"^^xsd:double ;
37              predicate:hazopnode "Speicherbehälter" ;
38              predicate:parameter "NaN"^^xsd:double ] ;
39      blanknode:deviation [ predicate:description "nur reines Lö
           sungsmittel in Behälter" ;
40              predicate:guideword "Kein" ;
41              predicate:hazopnode "Speicherbehälter" ;
42              predicate:parameter "Konzentration" ] ;
43      blanknode:restrisiko [ predicate:avoiding "NaN"^^xsd:double
           ;
44              predicate:presence "NaN"^^xsd:double ;
45              predicate:probability "NaN"^^xsd:double ;
46              predicate:severity "NaN"^^xsd:double ] ;
47      blanknode:riskgraph [ predicate:avoiding "G1 - möglich" ;
48              predicate:presence "A2 - häufig bis andauernd" ;
49              predicate:probability "W3 - relativ hoch" ;
50              predicate:severity "S1 - minimale " ] ;
51      blanknode:safeguard [ predicate:hazopnode "In 1 - Feed-
           Eingang" ;
```

```
52              predicate:otherinfo "NaN"^^xsd:double ;
53              predicate:parameter "keine Aktion erforderlich" ;
54              predicate:recommendation "Normalzustand" ] .
```

		Deviation		Cause		Consequence		Risk Graph			Safeguard		Risk Graph	
	ZOP Case	HAZOPNo	Parameter	Guideword	Description	HAZOPNo	Parameter	Guideword	Description	Severity	cy of presen	of avoidan	HAZOPNo	Description	mmendati	Other info	Severity	cy of presen	of avoidan
Index																			
0	1	Speicherbeh	Temperatur	Mehr	Überschreitu	1 - Feed-/	Temperatur	Mehr	Zugeführtes	Speicherbehälter									
1	2	Speicherbeh	Temperatur	Mehr	Überschreitu	Speicherbeh	Temperatur	Mehr	Hohe Temp	Speicherbeh	Druck	Mehr	Materialversa	S2 - geringe	A2 - häufig	G2 - fast un	W2 - gering	Speicherbeh	Hochwertige
2	3	Speicherbeh	Temperatur	Mehr	Überschreitu	Speicherbeh	Medium	Anderer/An	Brand im A	Speicherbehälter			Schadgastigk	S3 - schwer	A2 - häufig	G1 - möglic	W1 - sehr gr	Speicherbeh	Sicherheitsvo
3	4	Speicherbeh	Temperatur	Mehr	Überschreitu	Speicherbeh	Medium	Anderer/An	Verunreinig	Speicherbeh	Druck	Mehr	Leckage dur	S2 - geringe	A2 - häufig	G1 - möglic	W1 - sehr gr	Speicherbeh	Stromlos-se
4	5	Speicherbeh	Temperatur	Mehr	Überschreitu	Speicherbeh	Medium	Anderer/An	Verunreinig	Speicherbeh	Druck	Mehr	Schadgastigk	S3 - schwer	A2 - häufig	G1 - möglic	W2 - gering	Speicherbeh	Sicherheitsvo
5	6	Speicherbeh	Temperatur	Mehr	Überschreitu	2 - N2 Ei	Medium	Anderer/An	Stoffverwec	Speicherbehälter			Bilden eines	S3 - schwer	A2 - häufig	G1 - möglic	W1 - sehr gering		
6	7	Speicherbeh	Temperatur	Weniger	Unterschreit	Speicherbehälter													
7	8	Speicherbeh	Druck	Mehr	Überschreitu	1 - Feed-/	Druck	Mehr	Druck des e	Speicherbeh	Druck	Mehr	Druckerhöh	S3 - schwer	A2 - häufig	G1 - möglic	W2 - gering	Speicherbeh	Sicherheitsvo
8	9	Speicherbeh	Druck	Mehr	Überschreitu	Speicherbeh	Medium	Anderer/An	Verunreinig	Speicherbeh	Druck	Mehr	Temperatur	S3 - schwer	A2 - häufig	G1 - möglic	W2 - gering	Speicherbeh	Sicherheitsvo
9	10	Speicherbeh	Konzentrati	Kein	nur reines L	In 1 - Feed-/	Konzentrati	Kein	reines Lösu	Speicherbehälter			kein	S1 - minima	A2 - häufig	G1 - möglic	W3 - relativ	In 1 - Feed-/	keine Aktio
10	11	Speicherbeh	Konzentrati	Mehr	starke Säure	In 1 - Feed-/	Konzentrati	Mehr	starke Säure	Speicherbeh	Konzentrati	Mehr	Korrosion a	S4 - katastro	A2 - häufig	G2 - fast un	W1 - sehr gh	In 1 - Feed-/	Schulung u
11	12	Speicherbeh	Konzentrati	Mehr	organisches	In 1 - Feed-/	Konzentration		organisches	Speicherbeh	Konzentrati	Mehr	Beschädigun	S3 - schwer	A2 - häufig	G1 - möglic	W2 - gering	In 1 - Feed-/	Schulung u
12	13	Speicherbeh	Konzentrati	Mehr	hohe Viskos	In 1 - Feed-/	Konzentrati	Mehr	hoch viskos	Out 1 - Feed	Leistung	Weniger	Pumpe ist üb	S2 - geringe	A2 - häufig	G2 - fast un	W1 - sehr gh	In 1 - Feed-/	Schulung u
13	14	Speicherbeh	Konzentrati	Mehr	Feststoffe bi	Speicherbeh	Konzentrati	Mehr	Feststoffe d	Out 1 - Feed	Druck	Weniger	Höherer Dru	S2 - geringe	A2 - häufig	G1 - möglic	W2 - gering		
14	15	Speicherbeh	Konzentrati	Mehr	Feststoffe w	In 1 - Feed-/	Konzentrati	Mehr	Feststoffe g	Out 1 - Feed	Druck	Weniger	Höherer Dru	S2 - geringe	A2 - häufig	G1 - möglic	W2 - gering	In 1 - Feed-/	Filter vor Bi
15	16	Speicherbeh	Konzentrati	Mehr	Dichtungsal	Speicherbeh	Konzentrati	Mehr	Feststoffe g	Out 1 - Feed	Druck	Weniger	Höherer Dru	S2 - geringe	A2 - häufig	G1 - möglic	W2 - gering		
16	17	Speicherbeh	Konzentrati	Anderer/An	Stickstoff zu	2 - N2 Ei	Konzentrati	Anderer/An	Luft, O2, Cl	Speicherbeh	Druck	Mehr	Explosionsr	S4 - katastro	A2 - häufig	G2 - fast un	W1 - sehr gering		

Figure 7: Graph ontology in Excel

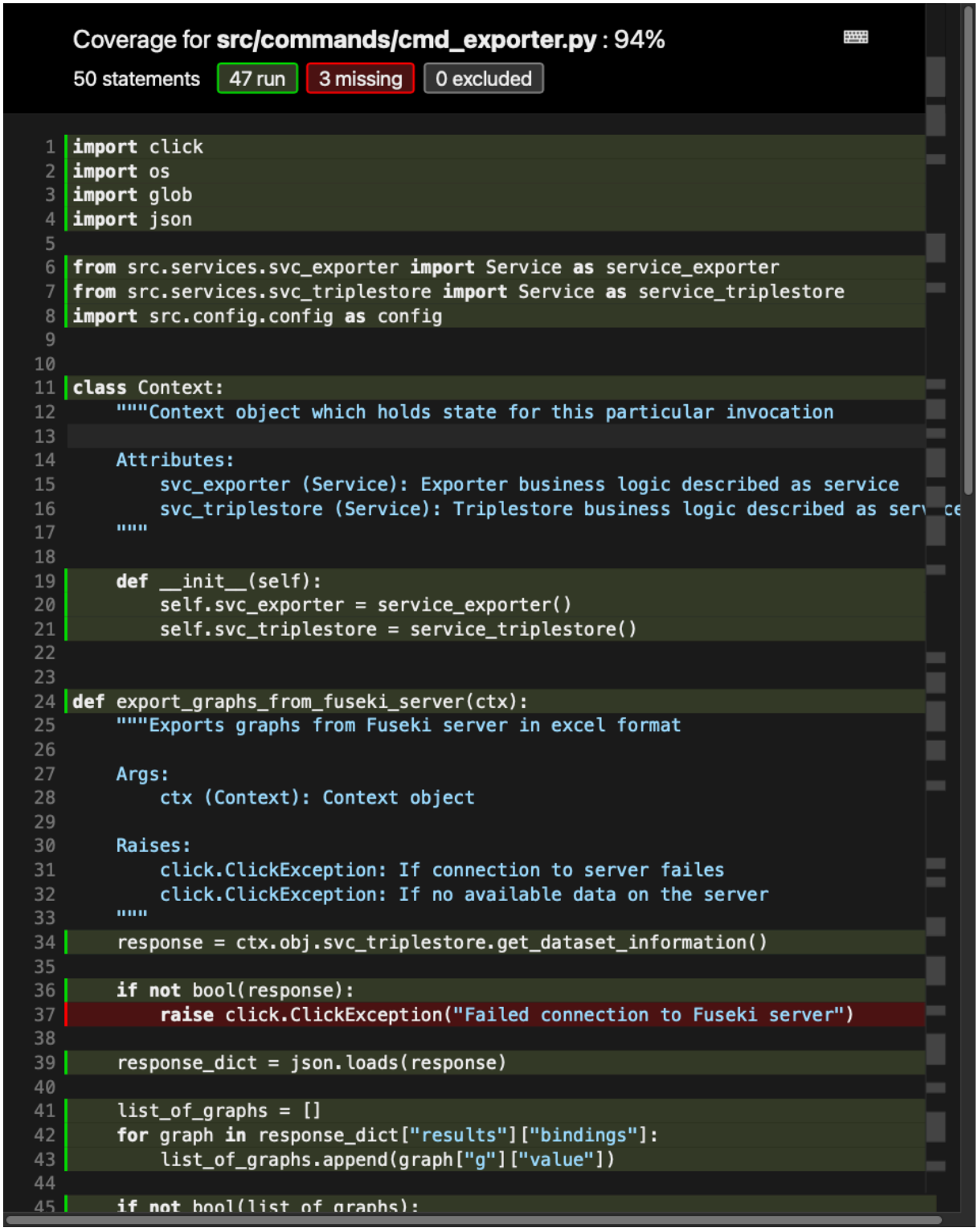

Figure 8: HTML coverage report example

References

Klose, Anselm, Christian Bramsiepe, Stefanie Szmais, Christian Schäfer, Niclas Krink, Wolfgang Welscher, and Leon Urbas. 2019. "Safety-Lifecycle of Modular Process Plants - Information Model and Workflow." In *2019 4th International Conference on System Reliability and Safety (ICSRS)*, 509–17. https://doi.org/10.1109/ICSRS48664.2019.8987685.