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

Software-Aided HAZOP: Handling HAZOP studies written in Excel

Dmytro Kostiuk
Marcus Rothhaupt
Artan Kabashi
Vincenz Forkel

TECHNISCHE UNIVERSITÄT DRESDEN

21.07.2021

Contents

Abstract **3**

Introduction **4**
Motivation 4

Problem Analysis **6**

Concept **7**
Program component design 7
User interaction scenario 8
Design Command Line Interface 8

Implementation **9**
Importer interface 10
Exporter interface 12
Remarks 13

Verification **14**

Summary **15**
Future Work 15

Appendix **16**

References **20**

List of Figures

1 Program component integration 8
2 User interaction scenario 8
3 Design Command Line Interface 9
4 Structure Command Line Interface 10
5 Sequence diagram Importer interface 12
6 Sequence diagram Exporter interface 13
7 Graph ontology in Excel 18
8 HTML coverage report example 19

Abstract

A Hazard and Operability (HAZOP) study is a widely used safety related document in the process manufacturing industry. Creating such a document is a time and labor-intensive process. This document is hand-crafted and written, thus human error cannot be avoided. To avoid such errors computer-aided HAZOP systems can be used to support human experts.

In this research paper, we propose such a computer aided HAZOP system. The system we propose is able to handle HAZOP data in the Excel and RDF format and allows for an easy-to-handle, back-and-forth conversion. Furthermore, a verification of the HAZOP data is integrated and can be configured further.

The program features a command line interface which allows access to an Importer and Exporter. The Importer can be used to import and validate incoming HAZOP data in Excel and generate RDF graphs from it, these can then be stored locally or uploaded to a Fuseki server. The Exporter can export RDF graphs containing HAZOP data to Excel. The RDF file for the Exporter can either be locally stored or on a Fuseki server.

Our findings show that our proposed computer-aided HAZOP system can effectively create RDF ontologies for manually created HAZOP studies. These RDF ontologies, not only enable better storage for HAZOP studies, but they are also machine readable, queryable, and therefore allow for further automation.

Keywords: HAZOP to RDF transformation, computer-aided HAZOP studies, safety engineering ontologies, handling HAZOP studies, Excel to RDF transformation, converting HAZOP studies

Introduction

Hazard and Operability (HAZOP) studies are conducted to identify and assess hazards and malfunctions that arise from processes and process plants. The HAZOP methodology is a human-centered and moderated technique and it is conducted by an interdisciplinary team of experts. (Single, Schmidt, and Denecke (2020))

For over 30 years, different research groups proposed rule-based expert systems or graph-based approaches in order to automate HAZOP studies, see (Single, Schmidt, and Denecke (2019)). Like (Rodríguez and Laguia (2019)) our software makes use of a promising ontology-based technology.

Being able to convert excel spreadsheets to the RDF format is a big time saver for engineers working with HAZOP studies. By analyzing the structure of the HAZOP files, conclusions can be drawn about the risks of a plant.

In the following parts of this paper, we show the capabilities of our software, how we solved the interfacing-problem of traditional HAZOP formats 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

In the ever more connected world of the twenty-first century consumer and supplying industry trends are changing faster than ever. Digital ordering of products from all over the world with the click of a button and one-click setups of internet store fronts that offer these products are creating a new set of demands for the global supply chain and the plants that form that chain. Gone are the days of large scale, single purpose plants and factories that can only create one product, no matter the

market conditions. This realization has become undeniable in the face of the global Covid-19 pandemic that tore apart global supply chains and brutally exposed the inflexibility of modern industrial production. First world countries in Europe and the USA faced long lasting and critical shortages of trivial items like cotton-swaps and surgical masks, along with simple chemicals like Isopropyl Alcohol. While no one would state that the actual production process for these items or compounds are overly complex, the setup of traditional single purpose, stationary plants is far too big an undertaking to simply solve a temporary shortage or to create a single large batch order from an oversea internet store front company. Not only is the planning period too long to solve the temporary but pressing shortage, or supply a onetime order, but the fully assembled plant will be underused and uneconomical once the crisis or consumer trend has passed.

The concept of the modular plant attempts to overcome all these limitations, allowing flexible production in small or large quantities over variable periods of time. Furthermore ,through modularization of modern process plants, an acceleration in design and engineering is achieved like described from Klose et al. (2019). The setup process is far quicker, and therefore more reactive to market demands, than that of a traditional plant. All modules are functionally tested and verified on an individual basis and are equipped with standardized mechanical and electronic interfaces, thus allowing for much simpler planning, construction, and operationality. Safety, however, must always be a top priority, both ethically and from a cost, cleanup, and downtime perspective. In order to meet those high safety standards, a HAZOP study, must be conducted on all parts of every plant. This study is vital and costly engineering work that must be done meticulously. Errors and oversights cost lives and damages that can range into the billions of Euros.

HAZOP studies today are still created analogous to the stationary plants they are covering. They are single purpose and offer no reusability despite the great efforts that go into creating them. Creating HAZOP studies in this fashion runs counterintuitive to the idea of the modular plant. Many of the advantages of modularity are lost, when a plant can be put together quicker, by less specialized personal, and in the end the ready-to-operate plant must stand idle until a classic, from the ground up, HAZOP study is finished. To meet the modern global demands of industry the modular plant needs a modular HAZOP. A HAZOP that, just like the module it covers has standardized mechanical and electronic interfaces, also has standardized

HAZOP interfaces. An interface that allows for a module specific HAZOP to be done at the module factory and reused for other modules of the same series, thus creating reusability of manual engineering work and allowing for a safety study of a full plant that is just as easy and quick to assemble as the modular plant itself.

Problem Analysis

The roots of the classic HAZOP risk analysis lie in the early concepts of (Fussell (1973)) in the 1970s. Fussell (1973) described automated fault tree analysis by piecing together "mini fault trees," which provides a methodology for filling out the cause columns of a HAZOP table. This early start of the HAZOP study methodology eventually developed into a risk analysis method used by many companies. Since then, there have been many attempts to automate HAZOP studies. Yet there are still very few industrial applications of automated HAZOP, and companies still invest tens to hundreds of thousands of dollars in performing HAZOP studies manually in HAZOP study workshops, see (Taylor (2017)).

Today this manual work is widely done in Excel spreadsheets. While those are easy to work in and readable for humans, they present a barrier to our vision of a truly modular plant. Excel sheets do not provide structures that are machine readable. Since the results are stored in large, packed tables for each individual plant, they lack an accessible way of storing and reading them. This lack hinders the potential to reuse the work that was put into the HAZOP studies, making them essentially one time use. Furthermore, the ontology used, in human readable form, is not rigid enough to apply semantic to it, hindering any automatic processing.

These inefficiencies can be remedied by converting the large HAZOP excel tables into an easier and better to handle format. We propose the use of a Resource Description Framework (RDF) ontology, instead of the classic Excel table approach. RDF ontologies allow for easy storage, locally or on designated servers. They are machine readable and combined with its wide level of adaption and easy to handle language, RDF HAZOPs solve the problem of reusability, saving both work and time in the plant building process. The queryable nature of RDF databases creates the possibility for automatic interfacing between individual module HAZOPs, analogous to the mechanical and electrical interfacing they already possess.

We recognize the advantages of Excel spreadsheets in the process of creating HAZOP safety studies. Further, we believe in giving the safety engineers the maximum amount of freedom to conduct their primary work in a way that suits their personal workflow best. Thus, we believe the best way to implement this solution is to create an Importer/Exporter and database that converts Excel sheets, which are easy to read for humans, to RDF ontologies and vise-versa.

Concept

Program component design

We decided to structure our solution into separate components. The components should have their own communication interfaces to perform certain actions with the incoming HAZOP data.

- **The Importer** should take the HAZOPs, validate them and convert them into the RDF format. Finally, the created RDF-Graph with all metadata is stored either locally or uploaded on the Fuseki server.
- **The Exporter** should take the RDF-Graphs, parse the information and metadata, and store them in an Excel file.
- **The TripleStore** is a Fuseki server API. It stores HAZOP data in RDF-Triples. Acting as a central database for machine-readable, completed HAZOP-Analysis with easy accessibility.

We planned to integrate the components through dependency injection and open the functionality of the components to each other. So, we can perform the complete cycle of RDF transformation from and to the Excel format.

Figure 1: Program component integration

User interaction scenario

Through the communication with the program the user should have the opportunity to load, read and store the HAZOPs in either format.

The concept allows the interaction with up to two interfaces, the importer and exporter.

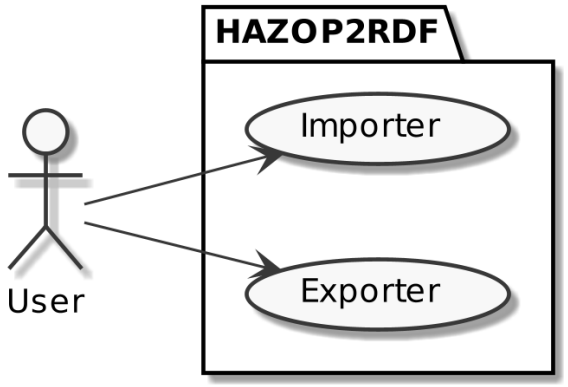

Figure 2: User interaction scenario

Design Command Line Interface

We choose a Command Line Interface (CLI) as a fundament for our project. The CLI should be able to provide the complete functionality spectrum of the importer and exporter to the user.

The diagram below shows the functionalities available to the user.

Figure 3: Design 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 interfaces, which use services. The services contain utilities needed for the interfaces to perform lower level actions.

[1] Python package: Click

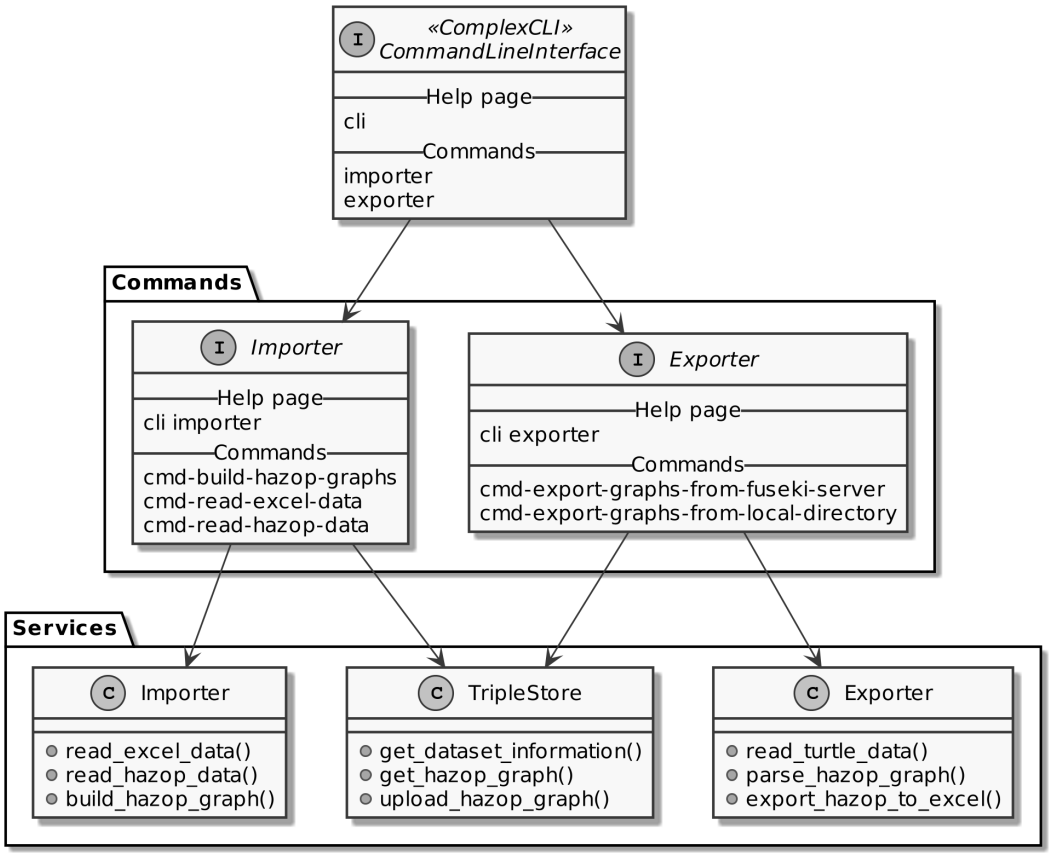

Figure 4: Structure Command Line Interface

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 the Excel format. The source for the Exporter interface can either be a locally stored RDF file or a 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 validation processes.

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 then be stored locally or uploaded to a Fuseki server. The two other commands cmd-read-excel-data and cmd-read-hazop-data can be used to check the local directory for the existing data.

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

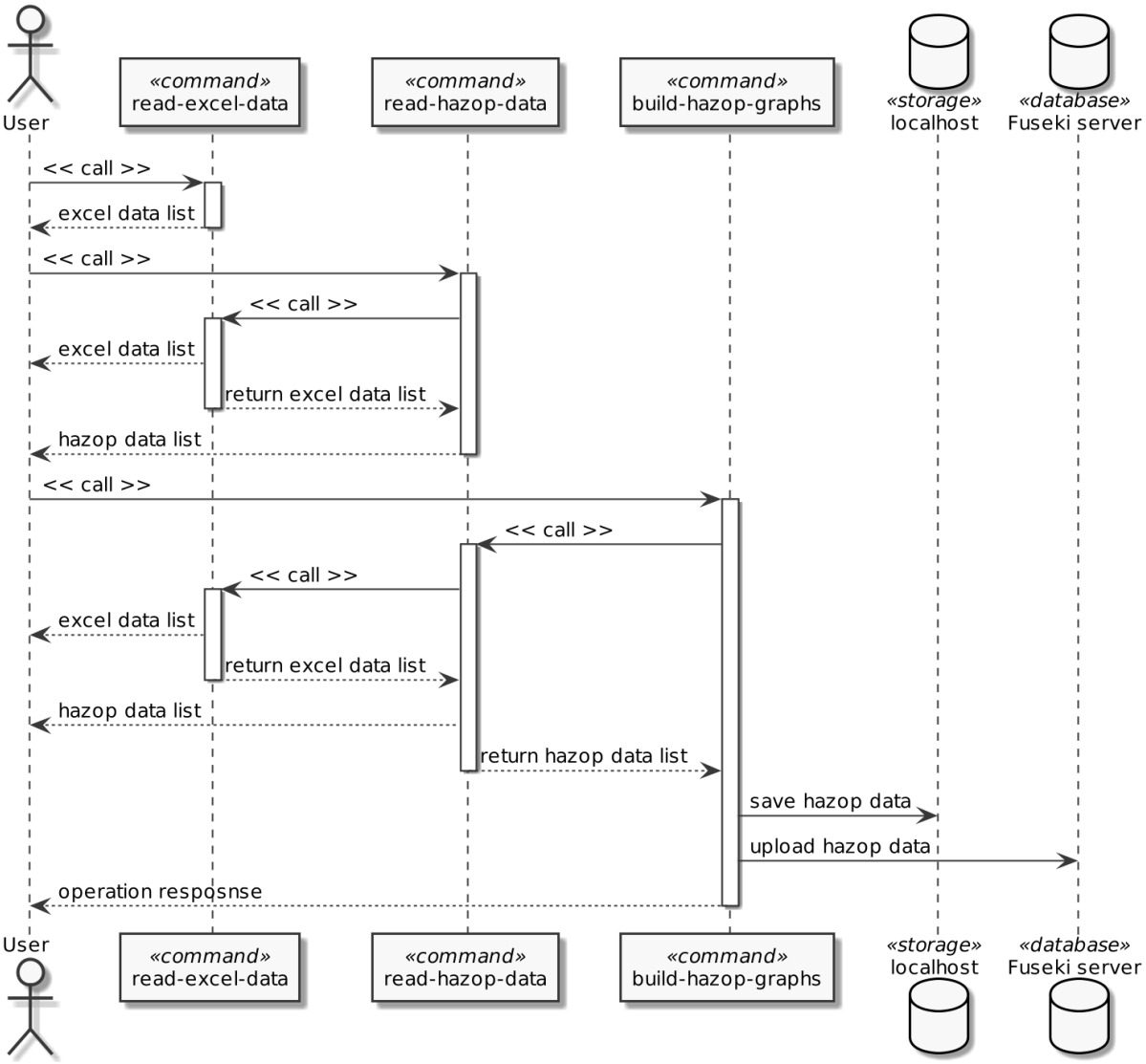

Figure 5: Sequence diagram Importer interface

Exporter interface

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

There are two main commands in the Exporter interface for the user 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 the Excel format again.

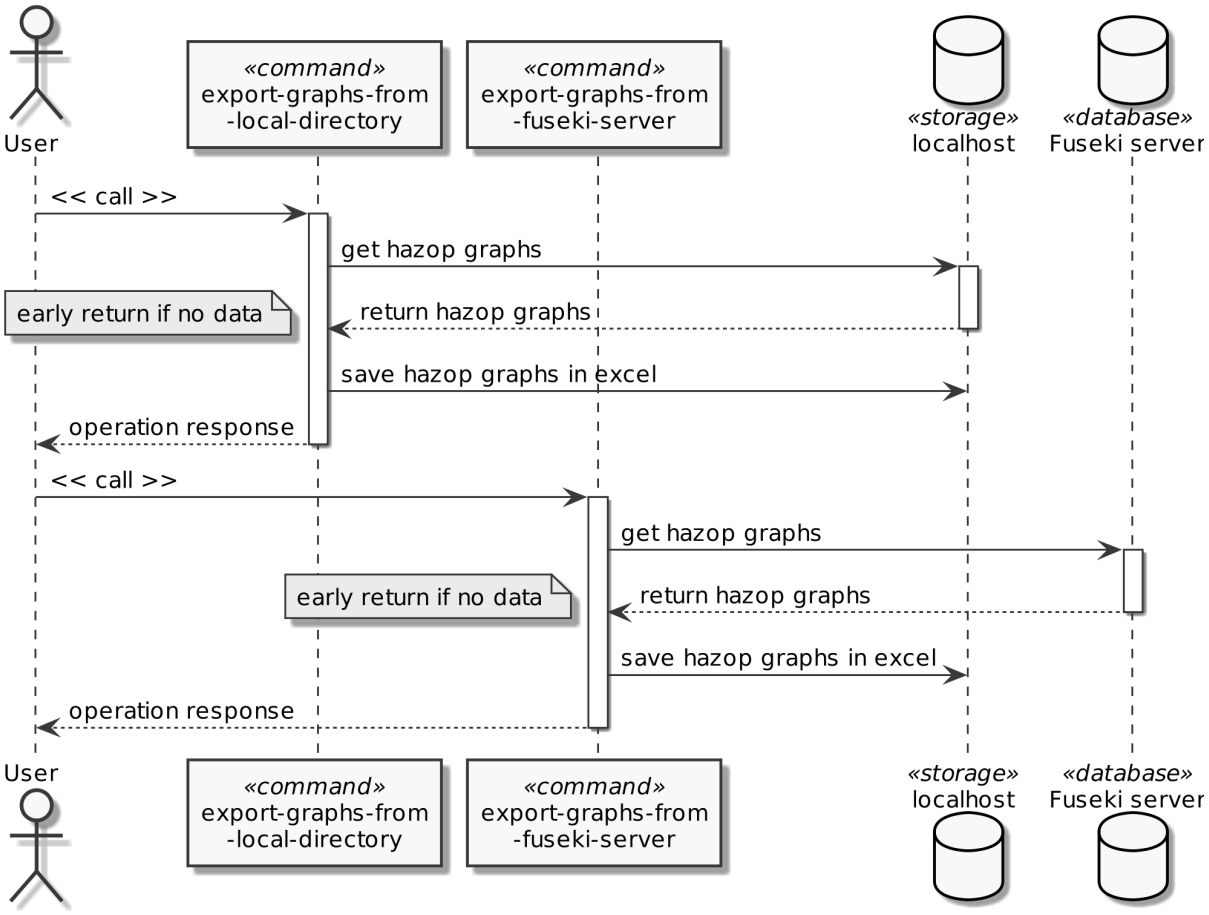

Figure 6: Sequence diagram Exporter interface

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 aims to check 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 a list of the following files can be found:

- 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

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

```
 9  src/excel_config/__init__.py          0      0   100%
10  src/excel_config/excel_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 also generate a coverage report in HTML 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 HAZOP studies

The vision would be to support humans in the decision-making in such a way that, based on HAZOP studies 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 room left for future adoptions, tests and experiments.

- 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 servers offer HTTP access. Currently, we use a set of command line scripts[2] to work with SPARQL. This API can be extended with the Requests[3] or SPARQLWrapper[4] package to perform HTTP requests internally and customise them.

Other ideas and improvements have their place. We focused on the basic functionalities, needed to transform the HAZOPs. To provide a richer experience for the client, the application can be extended to meet their individual requirements.

Appendix

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" ;
            predicate:guideword "Mehr" ;
            predicate:hazopnode "Speicherbehälter" ;
            predicate:parameter "Temperatur" ] ;
    blanknode:restrisiko [ predicate:avoiding "G2 - fast unmö
       glich" ;
            predicate:presence "A2 - häufig bis andauernd" ;
            predicate:probability "W2 - gering" ;
```

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

```
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" ] .
```

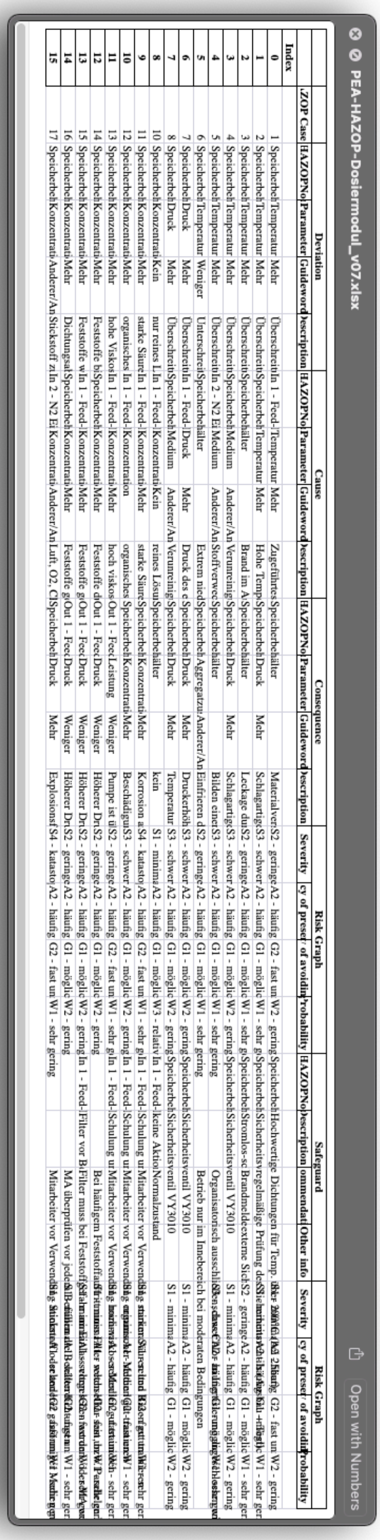

Figure 7: Graph ontology in Excel

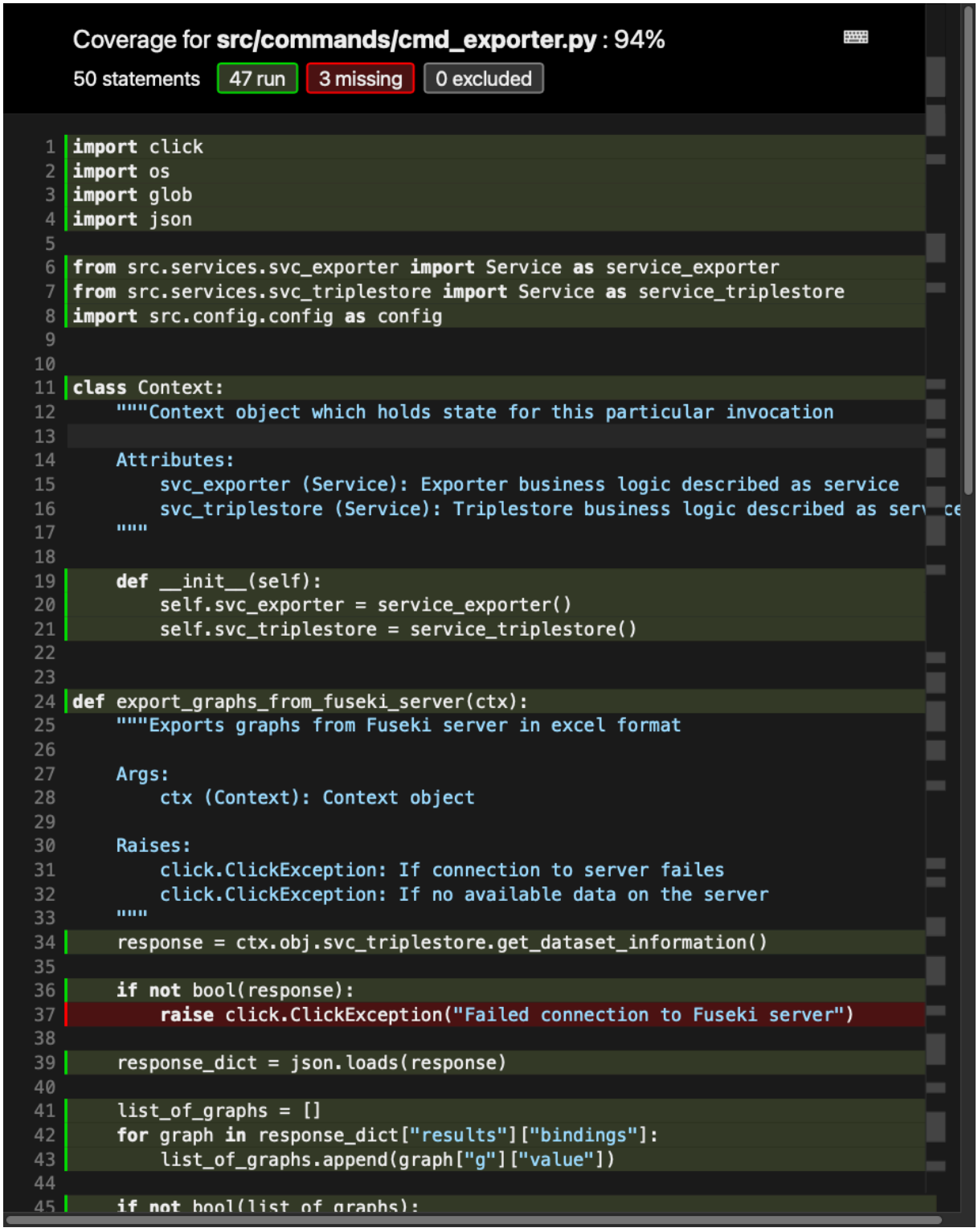

Figure 8: HTML coverage report example

References

Fussell, J. B. 1973. "A Formal Methodology for Fault Tree Construction." *Nuclear Science and Engineering* 52 (4): 421–32. https://doi.org/10.13182/NSE73-A23308.

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.

Rodríguez, M., and Jorge Laguia. 2019. "An Ontology for Process Safety." *Chemical Engineering Transactions* 77: 67–72. https://doi.org/10.3303/CET1977012.

Single, Johannes I., Jürgen Schmidt, and J. Denecke. 2020. "Computer-Aided Hazop: Ontologies and Ai for Hazard Identification and Propagation." In. https://doi.org/10.1016/B978-0-12-823377-1.50298-6.

Single, Johannes I., Jürgen Schmidt, and Jens Denecke. 2019. "State of Research on the Automation of HAZOP Studies." *Journal of Loss Prevention in the Process Industries* 62: 103952. https://doi.org/10.1016/j.jlp.2019.103952.

Taylor, J. R. 2017. "Automated HAZOP Revisited." *Process Safety and Environmental Protection* 111: 635–51. https://doi.org/10.1016/j.psep.2017.07.023.