
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



21.07.2021

Contents

Abstract	4
Introduction	5
Motivation	5
Problem Analysis	8
Concept	10
Program component design	10
User experience design	10
Command Line Interface design	11
Implementation	13
Importer interface	14
Exporter interface	15
Remarks	16
Verification	17
Summary	19
Appendix	20
References	24

List of Figures

1	Program component integration	10
2	User experience design	11
3	Command Line Interface design	12
4	Structure Command Line Interface	13
5	Sequence diagram Importer interface	15
6	Sequence diagram Exporter interface	16
7	Example HAZOP Case RDF graph	21
8	Graph ontology in Excel	22
9	HTML coverage report example	23

Listings

1	Test coverage report	17
2	HAZOP ontology in turtle format	20

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 a key part of such a computer aided HAZOP system. This key part is able to handle HAZOP data in the Excel and RDF format and allows for an easy-to-handle, back-and-forth transformation. Furthermore, a verification of the HAZOP data is integrated and can be configured further.

The idea for this system has its root in a college project task. After choosing the task to create such a “HAZOP transformation program” which should include an importer, exporter and a Triplestore we quickly began to do research about the main elements of the task.

We started by reading about how HAZOP studies are conducted and how ontologies can be formed with RDF. After the first success with a powerful python library named `rdf2lib` we kept adding on elements like a verification algorithm for the Excel HAZOP, a command line interface for easy access to the components of our software and a Fuseki Server integration for allowing a better storage of imported HAZOP RDF files.

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. Within the HAZOP analysis, information about risks of a plant is stored and can be made accessible for further conclusions.

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 are essential part of that chain. Fading 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 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 Iso-propyl Alcohol [Ranney, Griffeth, and Jha \(2020\)](#). 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, should be conducted on all parts of every plant. This study is vital and costly engineering work that must be done meticulously. Errors and oversights can cost lives and damages that with ranges into the billions of Euros¹.

HAZOP studies today are still created analogous to the stationary plants they are covering. They are single purpose and offer little 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

¹Article: [The Guardian](#)

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 Kletz (1986) in the 1980s. Although even earlier mathematical concepts like Fussell (1973), who described automated fault tree analysis by piecing together “mini fault trees,” can be found, Kletz (1986) is regarded as the first major work for modern HAZOP studies. This early start of the HAZOP study methodology eventually developed into a risk analysis method used by many companies today. Since the 1980s, 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 contain utilities 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.

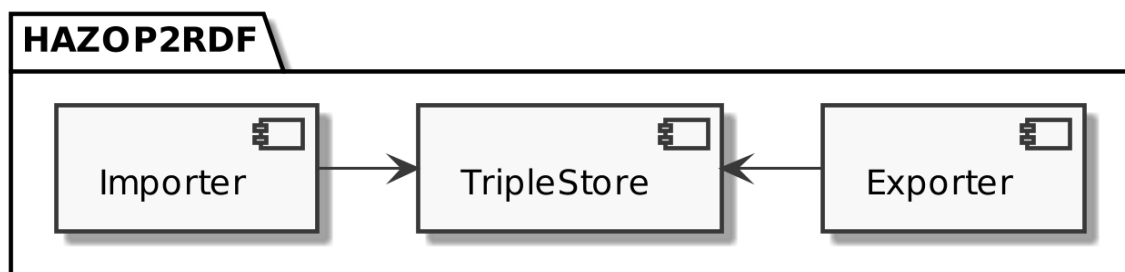


Figure 1: Program component integration

User experience design

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.

On the following diagram we show the concept of the user interaction with the program. Using a program interface the full cycle of HAZOP2RDF transformation should be accessible to the user.

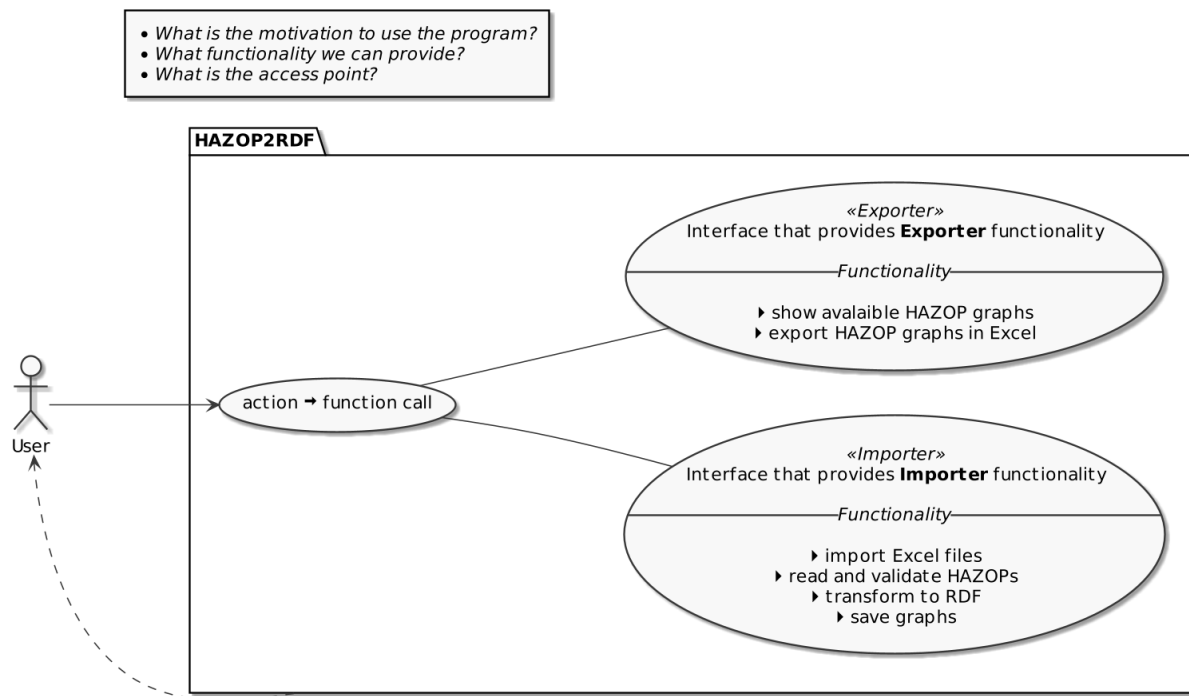


Figure 2: User experience design

Command Line Interface design

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

The diagram below shows the main features of the Command Line Interface.

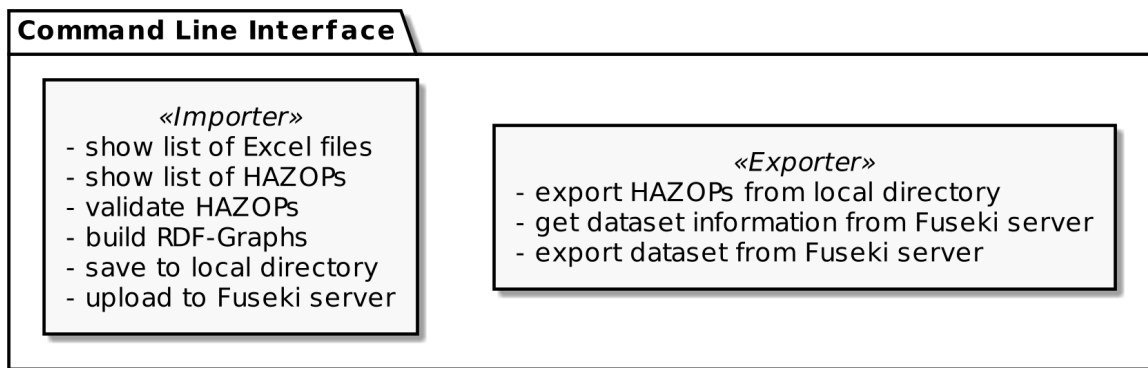


Figure 3: Command Line Interface design

Implementation

We implemented the Command Line Interface using Click² 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.

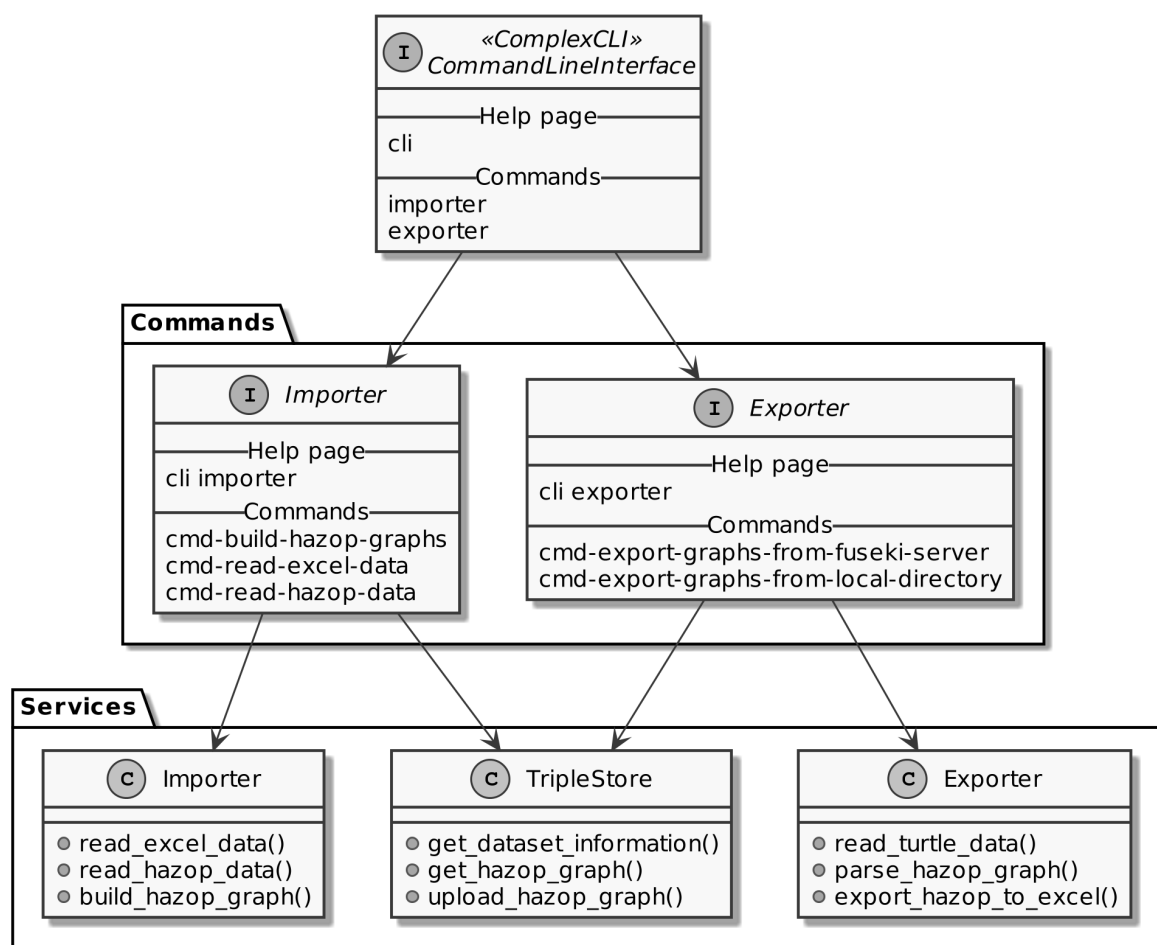


Figure 4: Structure Command Line Interface

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

²Python package: [Click](#)

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 read, filter and validate the incoming HAZOP data. 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.

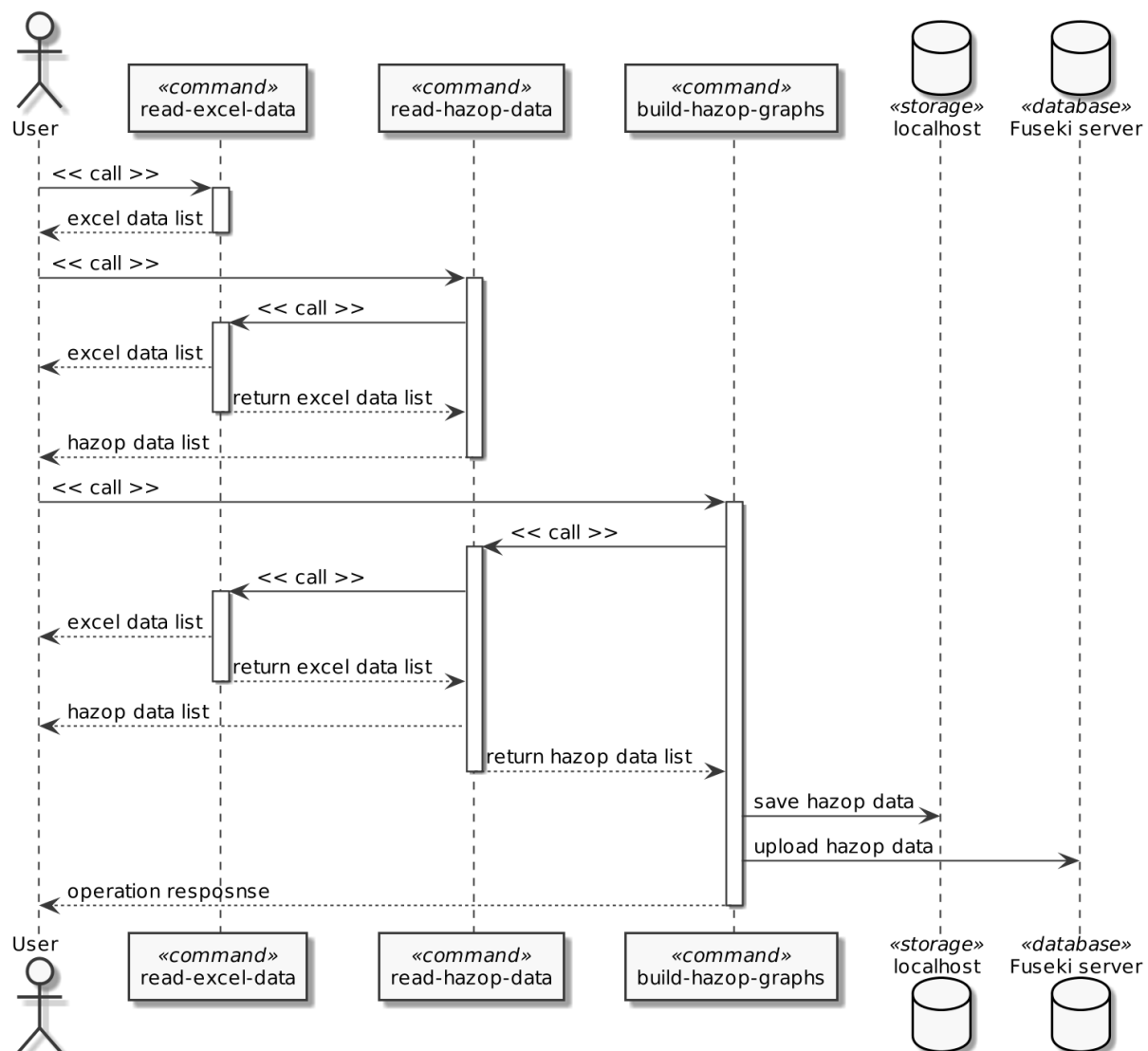


Figure 5: Sequence diagram Importer interface

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.

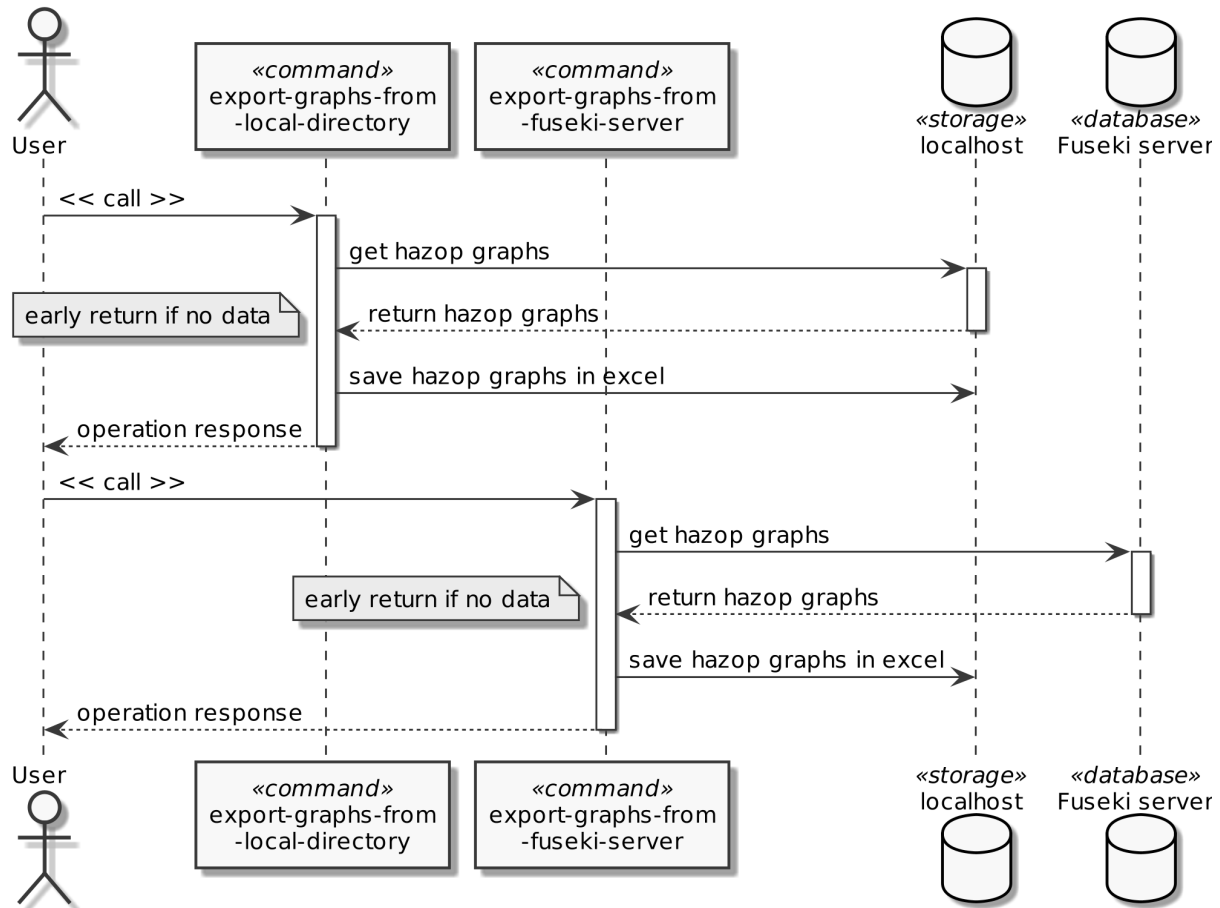


Figure 6: Sequence diagram Exporter interface

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: Test 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/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

Our software lays the groundwork for our vision of a modular HAZOP for a modular plant. It is easily integrable into the traditional HAZOP workflow, by allowing the engineering team to continue conducting the HAZOP studies in Excel. However, it solves the many follow up problems associated with that format. The user is able to overcome the storage and reusability issues, while also having his HAZOP studies available in a machine-readable format. The queryable nature of the RDF format allows for the separate future building of a HAZOP-to-HAZOP interface.

The user is provided with a Command Line Interface that enables the input validation of Excel HAZOP data and the lossless back-and-forth transformation of Excel and RDF formats. A possible storage solution, in the form of a Fuseki server, is also provided. Alternatively, all work can be conducted in the local directory.

The application provides the 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 constraints. 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 scripts³ to work with SPARQL. This API can be extended with the Requests⁴ or SPARQLWrapper⁵ 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.

With HAZOP data now readily available in RDF graphs, working on a HAZOP-to-HAZOP interface can begin.

³Set of scripts: [SOH \(SPARQL over HTTP\)](#)

⁴Python package: [Requests](#)

⁵Python package: [SPARQLWrapper](#)

Appendix

Listing 2: HAZOP ontology in turtle format

```

1 @prefix blanknode: <http://www.hazop2rdf.de/hazop/blanknode/> .
2 @prefix hazopcase: <http://www.hazop2rdf.de/hazop/hazopcase/> .
3 @prefix predicate: <http://www.hazop2rdf.de/hazop/predicate/> .
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5
6 hazopcase:1 blanknode:cause [ predicate:description "Zugeführtes
   Prozessmedium zu heiß (>200°C)" ;
7     predicate:guideword "Mehr" ;
8     predicate:hazopnode "In 1 - Feed-Eingang" ;
9     predicate:parameter "Temperatur" ] ;
10 blanknode:consequence [ predicate:description "
   Materialversagen der Dichtungen, Leckage" ;
11     predicate:guideword "NaN"^^xsd:double ;
12     predicate:hazopnode "Speicherbehälter" ;
13     predicate:parameter "NaN"^^xsd:double ] ;
14 blanknode:deviation [ predicate:description "Überschreitung
   der zulässigen Temperatur im Behälter" ;
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 ] .

```

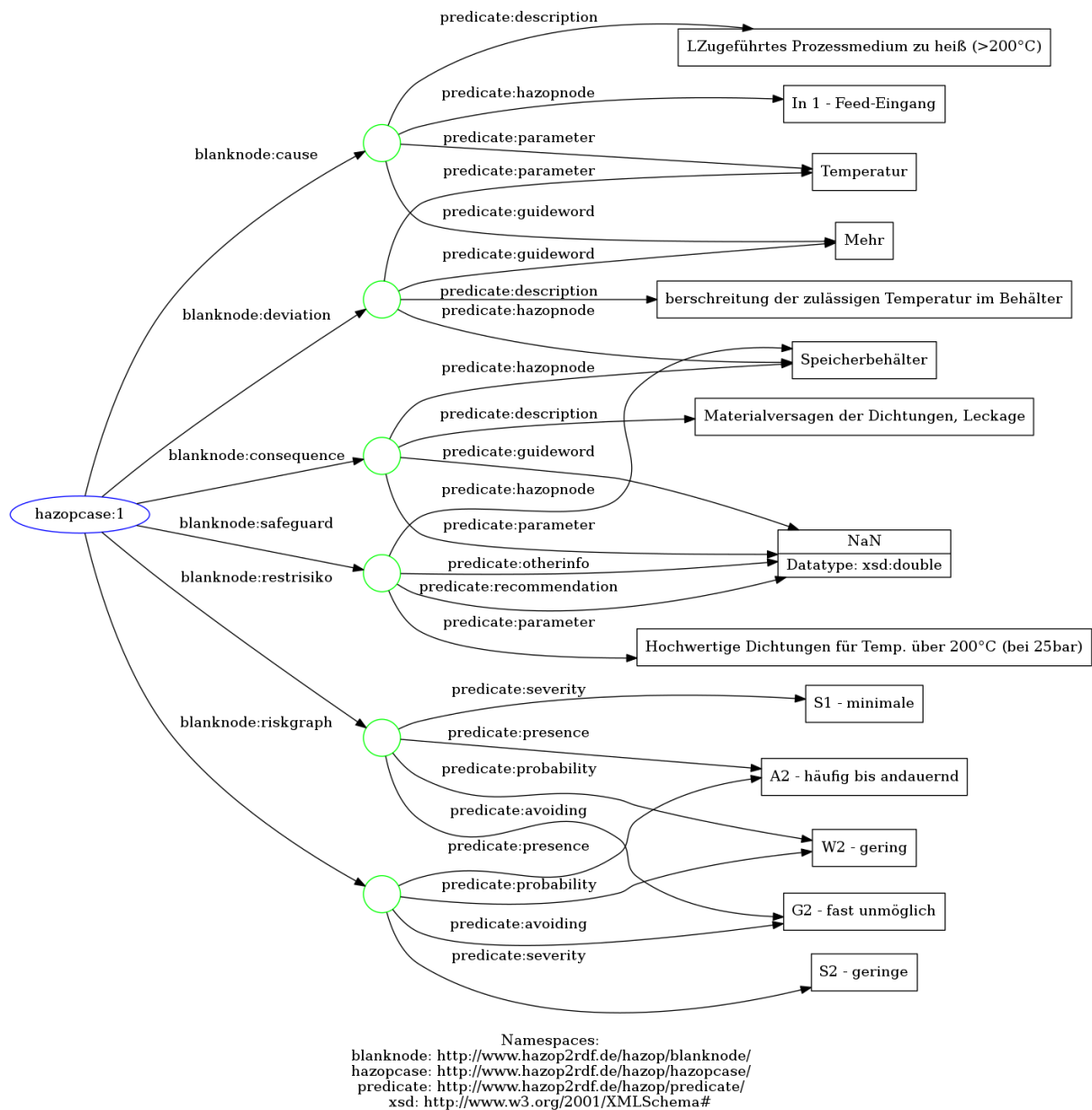


Figure 7: Example HAZOP Case RDF graph



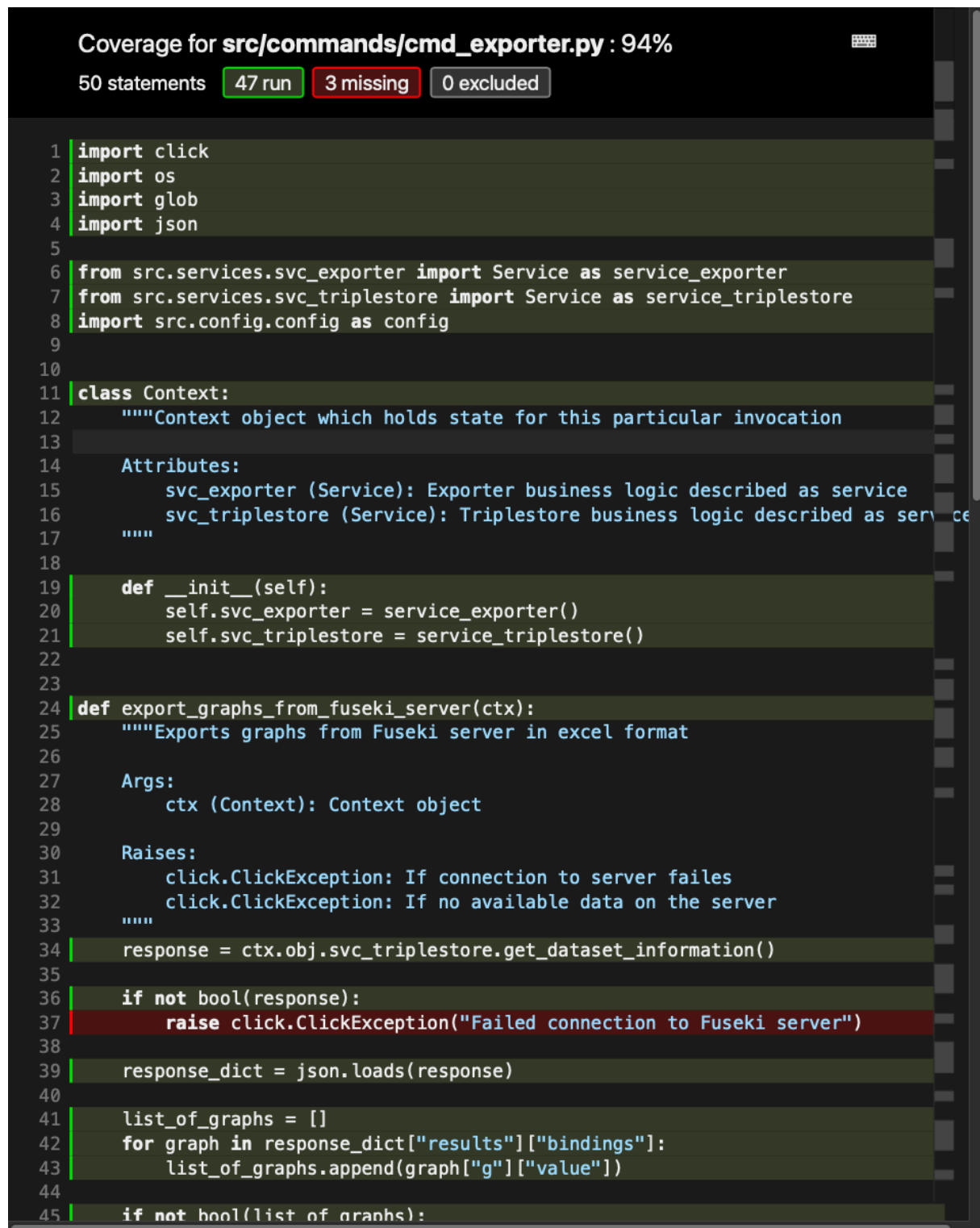


Figure 9: HTML coverage report example

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>.
- Kletz, Trevor A. 1986. *Hazop & Hazan: Hazard Workshop Modules ; Notes on the Identification and Assessment of Hazard*. Institution of Chemical Engineers.
- 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>.
- Ranney, Megan L., Valerie Griffeth, and Ashish K. Jha. 2020. "Critical Supply Shortages — the Need for Ventilators and Personal Protective Equipment During the Covid-19 Pandemic." *New England Journal of Medicine* 382 (18): e41. <https://doi.org/10.1056/NEJMp2006141>.
- 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>.