Project CAE-PA: HAZOP2RDF

Subtitle

Dmytro Kostiuk
Marcus Rothhaupt
Artan Kabashi
Vincenz Forkel

22.22.2022

Contents

List of Tables

List of Figures

Introduction

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.

As a result of our research we conclude that handling HAZOP data in RDF Format is much more convenient than working with excel data because the descriptive format largely increases compatibility between different modular plants. With RDF we can build graphs, examine dependencies between single HAZOP cases and store the RDF files in less disk space.

Problem Analysis

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 nor 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.

?? * When can it help? * Whom can it help?

Concept

- How did we solve the problem(framework)
- How did we get to the concept?
- Why did we choose this method of implementing

Design program components

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 0.1: Program components

User interaction scenario

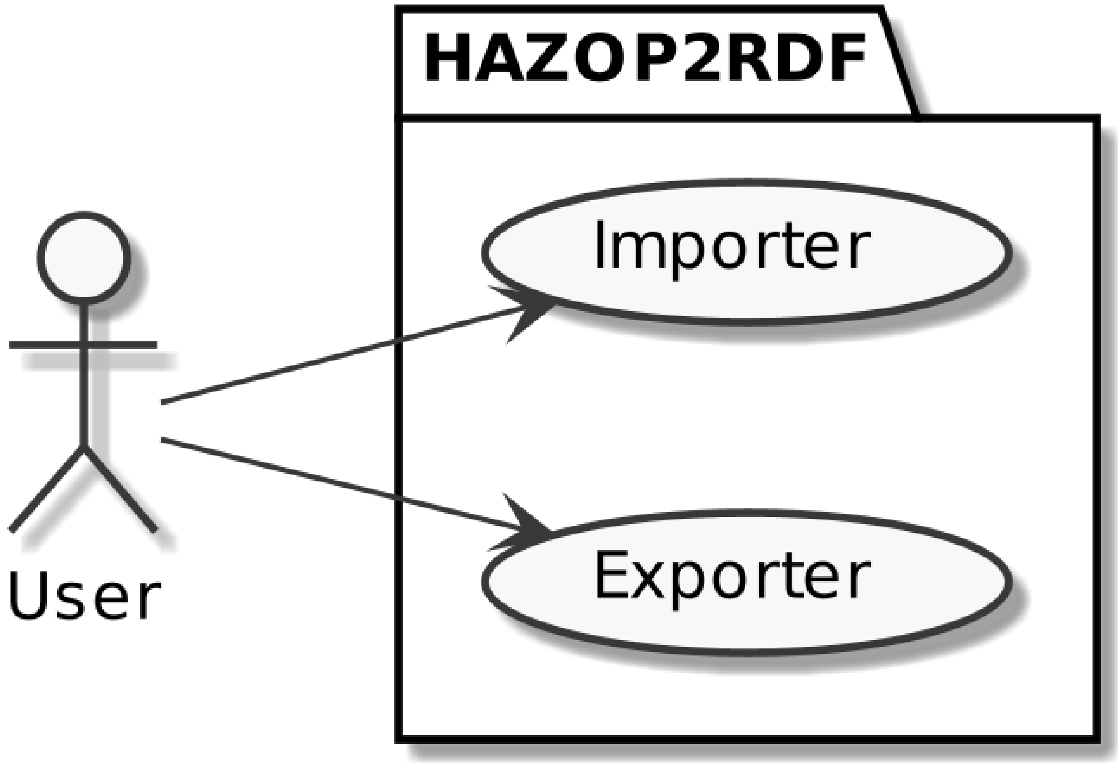

Figure 0.2: User interaction

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.

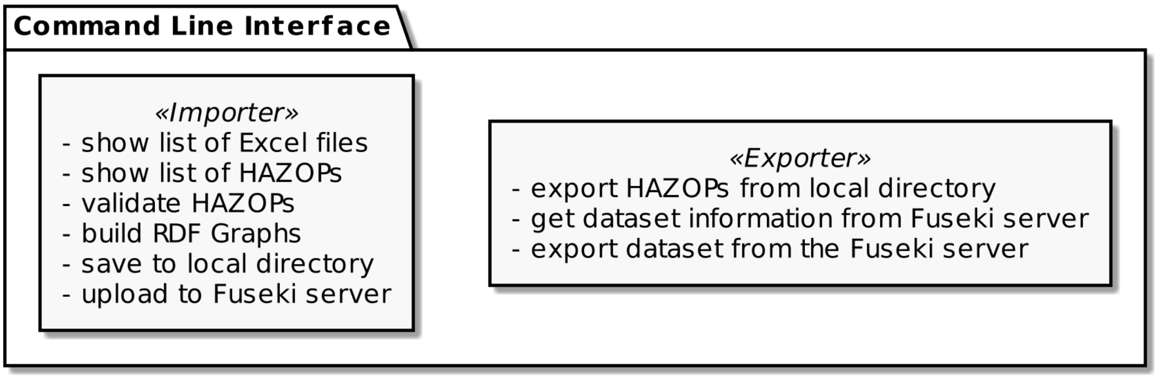

Figure 0.3: Design: Command Line Interface

Implementation

We implemented the Command Line Interface using Click 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.

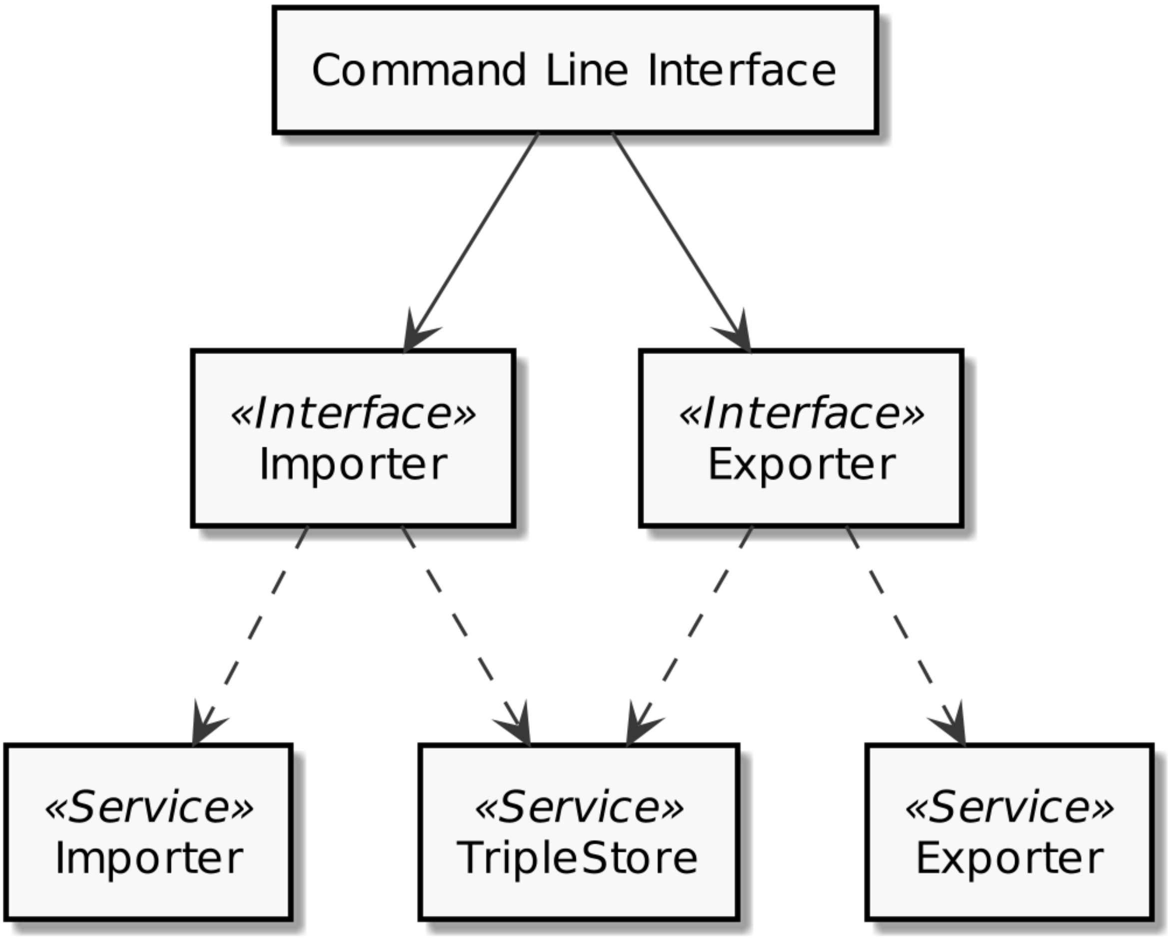

Figure 0.1: Structure of Command Line Interface

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.

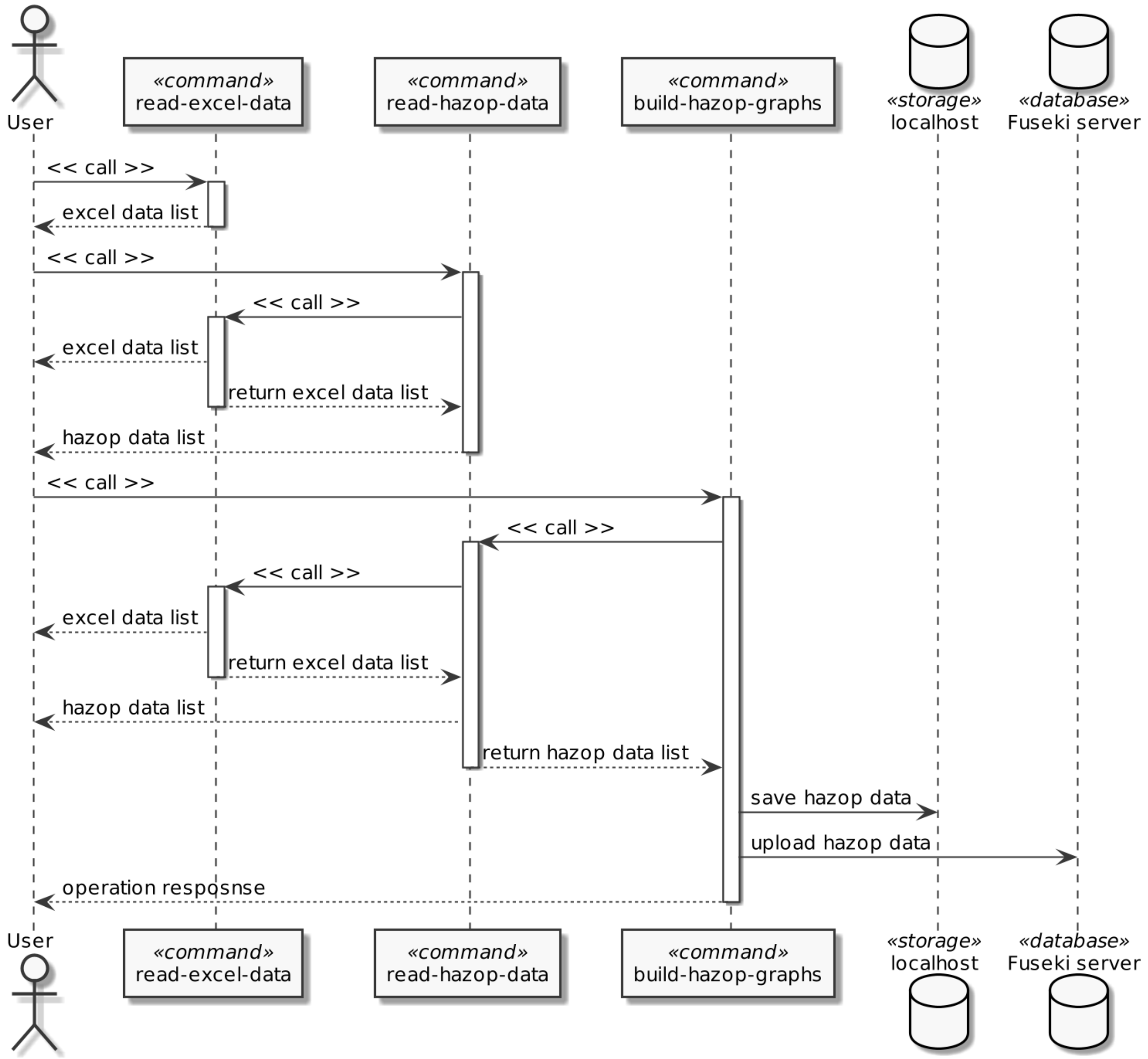

Figure 0.2: Sequence diagram of Importer interface

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.

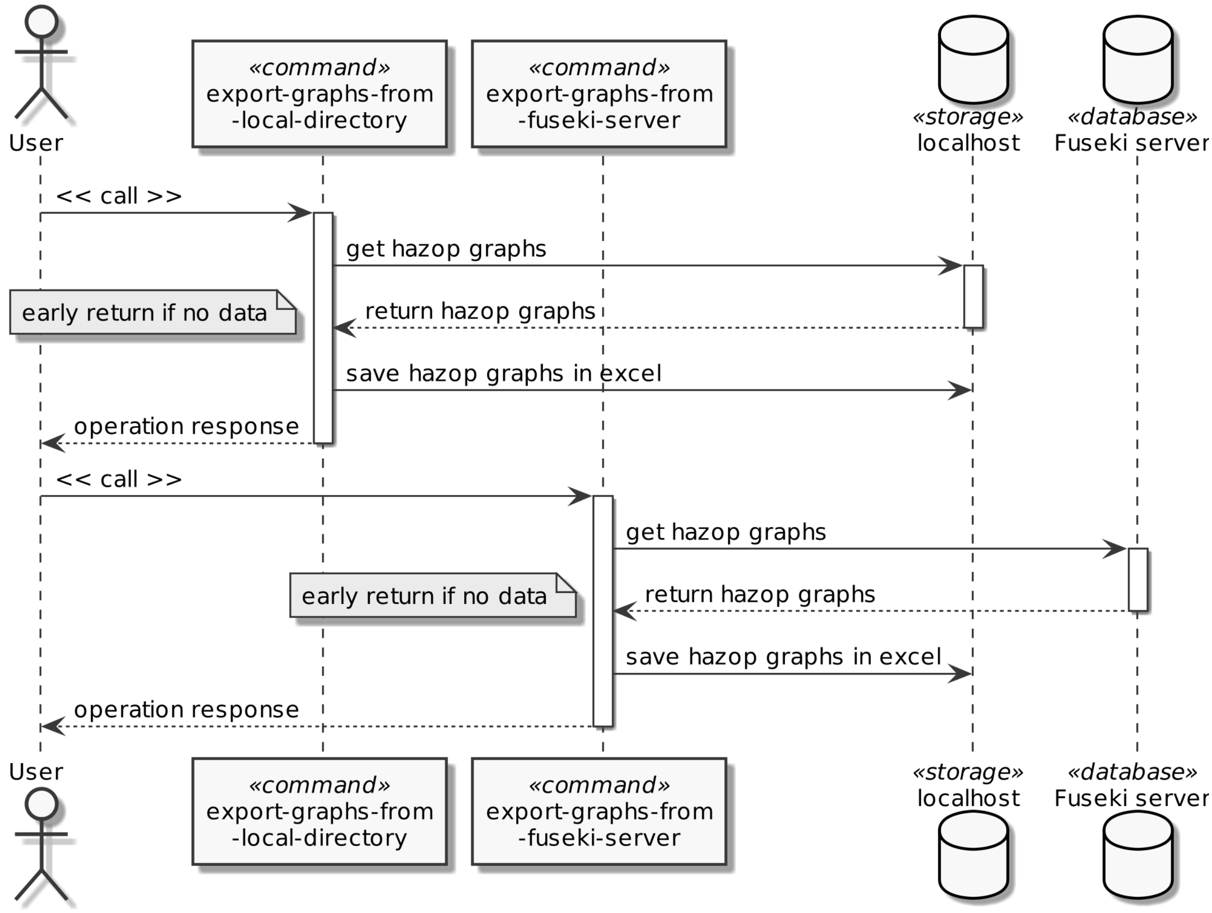

Figure 0.3: Sequence diagram of Exporter interface

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 important part of the project is verification. We designed a test pattern to verificate the results of the program execution. This pattern is simple and extendable and covers the main parts of the program.

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

- `test_cli.py` - test command line interface initialization
- `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.

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

The user can although generate a coverage report in HTML format and easily discover the missing statements.

There are predefined commands in the Makefile for quick access.

Summary

- Discussion
- Future Work
- Future Projects
- Problems to be solved

PROBLEM: We use a fixed schema to validate the input data, by pattern mismatch the data will be skipped.
TODO: Dynamic schema for accepting dynamic changes.
NEED: Dynamic changes constrains.

PROBLEM: We don't validate the logic of the HAZOP data.
TODO: Validation schema.
NEED: Rules, what are acceptable in HAZOP and what are not.

IMPROVEMENT: TripleStore SOH API (SPARQL over HTML) to an API using Requests or SPARQLWrapper packages.
ADVANTAGES: better control over request/response operations, customization.

Appendix

- turtle data, output ontology in excel, documentation

```
@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 unmöglich" ;
            predicate:presence "A2 - häufig bis andauernd" ;
            predicate:probability "W2 - gering" ;
            predicate:severity "S1 - minimale " ] ;
    blanknode:riskgraph [ predicate:avoiding "G2 - fast unmöglich" ;
            predicate:presence "A2 - häufig bis andauernd" ;
            predicate:probability "W2 - gering" ;
            predicate:severity "S2 - geringe" ] ;
    blanknode:safeguard [ predicate:hazopnode "Speicherbehälter" ;
            predicate:otherinfo "NaN"^^xsd:double ;
```

```
            predicate:parameter "Hochwertige Dichtungen für Temp. über 200°C (bei 25bar)" ;
            predicate:recommendation "NaN"^^xsd:double ] .

hazopcase:10 blanknode:cause [ predicate:description "reines Lösungsmittel wird zugeführt" ;
            predicate:guideword "Kein" ;
            predicate:hazopnode "In 1 - Feed-Eingang" ;
            predicate:parameter "Konzentration" ] ;
    blanknode:consequence [ predicate:description "kein" ;
            predicate:guideword "NaN"^^xsd:double ;
            predicate:hazopnode "Speicherbehälter" ;
            predicate:parameter "NaN"^^xsd:double ] ;
    blanknode:deviation [ predicate:description "nur reines Lösungsmittel in Behälter" ;
            predicate:guideword "Kein" ;
            predicate:hazopnode "Speicherbehälter" ;
            predicate:parameter "Konzentration" ] ;
    blanknode:restrisiko [ predicate:avoiding "NaN"^^xsd:double ;
            predicate:presence "NaN"^^xsd:double ;
            predicate:probability "NaN"^^xsd:double ;
            predicate:severity "NaN"^^xsd:double ] ;
    blanknode:riskgraph [ predicate:avoiding "G1 - möglich" ;
            predicate:presence "A2 - häufig bis andauernd" ;
            predicate:probability "W3 - relativ hoch" ;
            predicate:severity "S1 - minimale " ] ;
    blanknode:safeguard [ predicate:hazopnode "In 1 - Feed-Eingang" ;
            predicate:otherinfo "NaN"^^xsd:double ;
            predicate:parameter "keine Aktion erforderlich" ;
            predicate:recommendation "Normalzustand" ] .
```

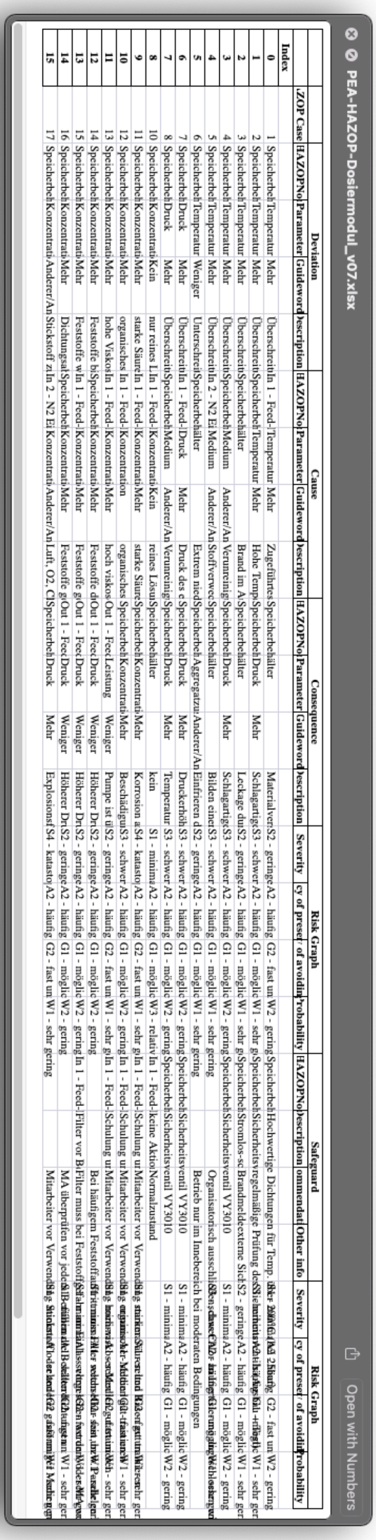

Figure 0.1: Graph ontology in Excel

References