A Proposal for Modelling TRIZ Functional Analysis

Tarek Stelzle

April 11, 2021

Contents

1 Aim of the work

The aim of this paper is to elaborate a proposal for an ontological modelling of the areas of *TRIZ Functional Analysis* based on the approaches in [3], [11], [12] and further own investigations. The work fits into the activities of the *WUMM Ontology Project* [7] to model core TRIZ concepts using modern semantic web means. The work consists of two parts – a *turtle file*, in which the semantic modelling is performed based on the SKOS framework [6], and *this elaboration*, in which the backgrounds and motivations of the concrete modelling decisions are detailed.

The paper is structured in the following way. In section 2 the information sources are mentioned and further explained. In the following section the conceptualisation will be shortly explained. In section 4 the Functional Analysis as described in [3] will be summarized. The next section introduces Python tools which have been implemented to make the *turtle*-file creation easier. Following it will be shown how and why the *turtle*-file has been created in that way. In extension to the last section will be some example implementations of the *turtle*-file for Functional Analysis in section 7. In the last section a conclusion about the paper is given.

2 Starting point

The starting point for the building the ontolog for Functional Analysis is going to be the book *Systematische Innovationsmethoden* [3]. The definitions, demenstrations and explanations will be used to implement the ontology.

Other definitions for Functional Analyis will be picked from *The Glossary Of Tritz and TRIZ-Related Terms* by Valeri Souchkov [12]. As this is a great summary with good definitions.

These two information sources and the following more detailed explained Webinar from Nikolai Shchedrin are the starting point in building the ontology for Functional Analysis.

2.1 Webinar of Nikolai Shchedrin

Nikolai Shchedrin made a talk about building an ontology for *Function* and *Function Analysis* [11]. He has presented some insights on how to structure the ontolgy.

2.1.1 Function Classification

First he introduced, that there should a be a new classification for the functions. There should be three types:

1. Function of the subsystem
2. Function of the upper system
3. Function of the surrounding objects

2.1.2 Model Of A Function

Furthermore he introduced the *Model of a function*. With this a function is further described. Not only does it show the Action and the two components which interact, but it furthermore shows the parameter, the type of the function and the degree of execution.

The *Functional Model* is a graph representation of the system which is analyzed. Every node in the graph is a component or a subsystem of the system. The functions are represented by edges. This will be further explained in 4.3.

The *Model of a function* can be helpful for building the *Functional Model*.

2.1.3 Objectives Of A System

As explained by Nikolai Shchedrin the objective of a system can be divided into three objectives.

1. Primary Objective
2. Secondary Objective
3. Auxiliary Objective

The Primary Objective is the objective, which the system was build for.

Secondary Objectives of the system are functionalities which are offered by the system, but for which it was not mainly built.

And the Auxiliary Objectives fulfill the purpose to get the Primary Objective to work.

2.1.4 Primary Function

Furthermore a function can be classified in one of these three functions.

1. Primary Function
2. Core Function
3. Auxiliary Function

The Primary Function is the Primary Objective and its technical execution.

The Core Function represents a function, which directly helps executing the Primary Function.

The Auxiliary Functions describe the set of functions which help run other subsystems.

3 The Conceptualisations

The conceptualisations to be developed follow the basic assumptions and positings that are elaborated in more detail in [2]. In particular, the following namespace prefixes are used:

- `ex:` – the namespace of a special system to be modelled.
- `tc:` – the namespace of the TRIZ concepts (RDF subjects).
- `od:` – the namespace of WUMM's own concepts (RDF predicates, general concepts).

The concept will be to gather various definitions from the mentioned sources. With these sources the important TRIZ triple will be implemented.

From there the necessary connection will be created for the various triples. Furthermore missing concepts will be implemented to fully show the Functional Analysis and the belonging concepts.

For the organizing of the turtle file and to structure the building process of the turtle file, first of all the Functional Model will be built. This is done by following the five steps, which will be explained in 4.3. With the given model the Functional Analysis components can be added to attach all important concepts.

In the end the implementation will be supported by different example, which are going to use the implemented concepts. In figure 1 an implemented function of an example is shown.

```
ex:Steer
  a tc:function ;
  tc:SubjectActionObject ex:DriverTurnSteeringwheel ;
  tc:QualityOfFunction tc:UsefulFunction ;
  skos:prefLabel "Steer"@en ;
  skos:Definition "Turning the steering wheel by the
    driver to change the direction of the vehicle."@en .
```

Figure 1: Example For Function *Steer*

4 Functional Analysis

In this section a summary about the terms and definitions explained in [3] will be given. This is going to help explain in the following sections, why the *turtle*-file was modeled in that way.

4.1 Concept

In Functional Analysis the idea is to represent a system by various functions. This representation helps in organizing and structuring the system. To tackle the optimization problem a more precise look is taken at the non-useful and contradictory functions in the system.

The two main objectives of Functional Analysis in TRIZ are formulated in the following.

1. Recognizing Of Problems
   Hereby the objective is to find as many as possible non-useful or contradictory functions in the system.
2. Trimming Of Components
   To optimize the system some components have to be redesigned. During this process the functionality of the component must not be changed.

With these two main objectives there are five tasks to handle in Functional Analysis.

1. Recognizing Interactions Between Objects
2. Recognizing Problems Within The System
3. Formulating Open Problems
4. Innovative Redesign
5. Optimizing Systems By Trimming
6. Bypassing Patents

4.2 Quality Of A Function

To further analyze the functions it is recommended to give each function a level of quality. This quality of a function makes it easier to find problems within the system.

Accorind to the book there are five different quality levels.

1. Useful Function:
   A function works as intended and the result is only positive.
2. Useful, But Insufficient Function
   A function has a positive impact but the result is not satisfieing.
3. Useful, But Bad Controllable Function
   A function with a positive impact but is not satisfieing as the result cannot be controlled. Hence it is wrongly timed.
4. Useful, But Excessive Function
   A function with a positive impact but a bigger result than necessary.

5. Harmful Function
   A function which has a negative impact.

More precise definitions can be found in the book [3] or in the Glossary of Souchkov [12]. As both of these are merged into the *turtle*-file the defintions can now also be found there.

Nikolai Shchedrin mentioned also a new function quality in his web-seminar. This is called *Useful Function With Disadvanteges*. As mentioned on the website [13] this class includes Redundant Functions, Insufficient Functions, Bad Controllable Functions and Missing Functions.

As there is no source mentining the Definition of a Redundant Function and a Missing Function, these will be interpreted in the following way.

1. Redundant Function:
   A function with a positive result, which is implemented a second time in the system.
2. Missing Function:
   A useful function which is not implemented in the system and therefore missing.

4.3 Functional Model

The Functional Model structures the function and the components in the system. In the book the following steps are recommended when building a Functional Model. It is also mentioned that steps two to four can be made in one step.

1. Making A List Of Components
2. Determine Interactions
3. Linking Functions To Components (Subject)
4. Determining The Direction Of Function (Arrow)
5. Define The Quality Of The Function

With this a graph-like structure is built. Every node in this graph represents a component or subsystem within the system. An edge represents a function. Different styles and forms of an edge represent the quality of the function.

For easier reading and analyzing the Functional Model, it is possible to group components according to self-defined properties.

4.4 Positive And Negative Aspects

Using the Functional Analysis helps you to fully understand the system you are working with. Furthermore you can exchange knowledge between colleagues and clarify misunderstandings.

A Functional Analysis makes also sense, when no problem is known. This is good if you want to improve your system without knowing specific complications.

A negative attitude of the Functional Analysis, is that only known systems can be analyzed.

5 Implementation Of Tools

5.1 Creating Turtle File

For easier and faster creation of the "Matrix 2003" a small python tool was implemented. This tool reads the matrix from a *csv*-file and generates a *turtle*-file.

5.1.1 The CSV-File

First of alle *csv* file needs to be in the following syntax, because otherwise the matrix is wrongly interpreted. Every line in the csv file is a row in the matrix. Furthermore every comma seperates a column entry in a a row. The different entries in the matrix field are seperated by dashes.

For reading the matrix the tool "tabula" was used [9]. With this tool the Matrix2003 from the book "Systematische Innovation" was interpreted as a txt file. As this did not have the right syntax the small script, named *convertTxtMatrixToCsv.py*, was created. With this the matrix is converted into the right syntax, as mentioned above.

Using The Script Python will need to be installed on the computer. The command to run the script will then be:

python convertTxtMatrixToCsv.py <csv-file-name>

The output *csv*-file will be called *createdMatrix.csv*. The file *temporary.txt* is only created for storing the information temporary. It can be deleted afterwards.

5.1.2 Creating The Matrix

The following explains the implementation of the *createMatrix.py* script.

The script uses three other files. The first is the *principles.txt*. In this file all the forty principles are written down, with their belonging number at the beginning. Hence the script can map an entry index to the belonging principle name.

The second file, which is also used is called *parameters.txt*. In there are all the forty-eight parameters which represent the column and rows in the matrix. These names are similar to those for the Altshuller Matrix. The nine new parameters, which were introduced for the Matrix2003 are the following.

As this matrix introduced upper-parameters, a third file for these is needed. This has been called *upperParameters.txt*. The upper-parameters are linked by writing the upper-parameter number with a dot before the parameters in the *parameters.txt*. The python script then handles the rest.

1. QuantityOfInformation
2. FunctionEfficiency
3. Noise

4. HarmfulEmissions
5. CompatitbilityOrConnectability
6. Security
7. Vulnerability
8. AestheticsOrAppearance
9. ComplexityOfControl

The layout for the Turtle file is the same as the layout for the "Altshuller Matrix". As this makes it easier for later adjustments or improvements of the matrices.

At first the script creates the header for the Matrix2003, which is the same as in the "Altshuller Matrix". Then it defines the owl entry for the Matrix. Afterwards the script iterates over the entries in the *csv* file and builds the matrix in the following way.

```
<http://opendiscovery.org/rdf/Matrix/E.06.34>
  od:upperDecreasingParameter tc:Non-Functionality ;
  od:decreasingParameter tc:EaseOfUse ;
  od:upperIncreasingParameter tc:Physical Parameters ;
  od:increasingParameter tc:SurfaceOfTheStationaryObject ;
  od:recommendedPrinciple tc:PreferredAction, tc:Asymmetry,
      tc:Mediator, tc:SelfService ;
  a od:MatrixEntry .
```

Figure 2: Entry of the Matrix2003 Turtle File

Using The Script For running the script python will need to be installed on the computer. The script can then be started with the following command:

python createMatrix.py <csv-file-name>

Afterwards there will be a file called *created_matrix.ttl*. This will be the result of the script.

5.2 Translating

For translating the various tags a python script was implemented, which takes a turtle files as input and adds the missing language tags. Therefore the script checks each line which already has a language tag. Then it makes an api call with the first tag as the source language. It adds the language tags for English, German and Russian. Each tag is a separate api call. For translation the Google Translator is used.

Using the Script Python has to be installed on the computer. Furthermore does the script need the *deep_translator* library [10]. The library can be installed with the following command:

pip install deep-translator

Afterwards the script can be started. <*turtle-file-name*> specifies the file for which the missing language tags should be added.

python translateText.py <turtle-file-name>

This will output a file name *translated_<turtle-file-name>.ttl*. This is the new turtle file with the added language tags. Furthermore all translated words or sentences will be shown in the terminal output.

6 Creating And Modeling the Turtle-File

As explained in the book "Systematische Innovationsmethoden" the Functional Model for the Functional Analysis is built in five steps. These steps have been explained in 4.3. The explanation of the implementation will also be structured in these five steps.

Each of these steps is also a new WUMM concept in the RDF graph.

1. od:ListOfComponents
2. od:ListOfInteractions
3. od:FunctionForComponents
4. od:DirectionOfFunction
5. od:DefiningQualityOfFunction

For each step the necessary implementation steps will be explained.

In the following all implementation examples will only include the English language tags. In the Turtle File itself there are also tags in Russian and German.

6.1 Making A List Of Components

Within the first step for building the Functional Model all components which are part of the system are listed. Hence the system concept is implemented.

A system has different objectives, mentioned in 2.1.3. These objectives are implemented as TRIZ components. The *hasPrimaryObjective*, *hasSecondaryObjective* and *hasAuxiliaryObjective* WUMM concepts link a system to the belonging objective.

The system consists of subsystems and components. As a system can itself be a subsystem of another system, also an upper system is implemented.

The relations between these are created via new WUMM triples. These are *hasSubsystem*, *hasUppersystem*, *hasComponent* and *hasComponent*.

The objective of this step is to create a *Component Model*. In the component model all component and subsystems of the system are listed. As this is a Functional Model without the functions, it is implemented as a sub-concept of the Functional Model. The additions to the Functional Model will be added in the next steps.

Creating the Component Model is done with the Component Analysis. This is implemented as a sub-concept of the first step of Building the Functional Model.

The link to the Component Model is done via the *domain* and *range* properties. As the Component Analysis starts with a system and returns a Component Model.

The implementation of the Component Analysis is shown in figure 3.

```
tc:ComponentAnalysis
  a skos:Concept ;
  od:subConceptOf od:ListOfComponents ;
  od:SouchkovDefinition "A step in Function Analysis that identifies
    components of a technical system being analyzed and its supersystem."@en ;
  rdfs:domain tc:System ;
  rdfs:range tc:ComponentModel ;
  skos:prefLabel "Component Analysis"@en ;
  skos:altLabel "Component and Structural Analysis"@en .
```

Figure 3: Implementation Of Component Analysis

6.2 Determine Interactions

In the next step components, which in some kind interact with each other are listed. These are called Interactions and are implemented as a TRIZ concept.

The linking is done with a WUMM concept called *hasInteraction*. The RDFS property domain and range are both Components and hence an interaction is created.

All the interactions can be shown with an Interaction Matrix. This is mentioned in the implementation as a TRIZ concept.

The Interaction Matrix can be created with an Interaction Analysis. The Interaction Analysis is implemented, see figure 4, as a TRIZ concept. And as this leads to the result of the belonging interactions it is described as a sub-concept of the second step.

```
tc:InteractionAnalysis
  a skos:Concept ;
  rdfs:domain tc:System ;
  rdfs:range tc:InteractionMatrix ;
  od:subConceptOf od:ListOfInteractions ;
  od:SouchkovDefinition "A part of Function Analysis that identifies
    interactions between components included in a Component Model."@en ;
  skos:prefLabel "Interaction Analysis"@en ;
  skos:altLabel "Structure Analysis"@en ;
  skos:altLabel "Funtion Analysis of the Process"@en ;
  skos:altLabel "Process analysis"@en ;
  skos:example:  "The coffee production process is technically implemented
    by the components of the coffee machine."@en .
```

Figure 4: Implementation Of Interaction Analysis

6.3 Linking Functions To Components

In the next step functions will be linked to the belonging components.

Therefore a action is defined. Furthermore a WUMM concept is added which attaches an action to an interaction.

A function is defined as having a direction, but as this is going to be added in the next step the term function is going to be used without these functionality in this section.

The allowed values for constructing a function will be the TRIZ concept Subject-Action-Object. As this already includes the direction, this will be explained in the following section (6.4).

The sub-concepts of a function are the various specifications of a function. These are all summed up in the TRIZ concept *QualityOfFunction*, which is going to be explained in section 6.5.

The following three functions are also implemented as a sub-concept, but are not representing a quality of the function. These three ordering where introduced by Nikolai Shchedrin and have been explained in section 2.1.4.

- Main Function
- Auxiliary Function
- Core Function

Nikolai Shchedrin introduced the Model Of A Function, which will also be implemented. This model represents the function with two components, an action, a parameter, the type of the function and the degree of execution. The not already implemented *Parameter* and *Degree Of Execution* will be added as TRIZ concept. Furthermore WUMM concepts will be added, which attach these TRIZ concepts to a function.

As mentioned in section 2.1.1 Nikolai Shchedrin also added three new types for classifying a function. These types were also added as a TRIZ concept. With the WUMM concept *hasFunctionType* a function is connected with the according function type.

6.4 Determining The Direction Of The Function

The representation for the function is done with the concept of Subject-Action-Object. This is implemented as a TRIZ concept. Therefore the *Function Carrier* and the *Object of the Function* is needed. Both are also embedded as a TRIZ concept.

To connect an object of a function with a function carrier, the new WUMM concept *hasFunctionCarrier* is introduced. This is done similar for the opposite direction.

An action can be attached to a direction with *hasDirection*. This maps a action to a Subject-Action-Object triple.

6.5 Define The Quality Of The Function

The fifth and last step in building the Functional Model within the Functional Analysis defines the quality of the given function.

Therefore, as mentioned before, the TRIZ concept *QualityOfFunction* is introduced. This is a sub-concept of the function and in that way linked to the function.

```
tc:SubjectActionObject
  a skos:Concept ;
  od:subConceptOf tc:FunctionModeling ;
  od:hasSubConcept tc:FunctionCarrier, tc:ObjectOfFunction, tc:Predicate,
    tc:DirectionOfFunction ;
  od:SouchkovDefinition "A triad which identifies a Function Carrier,
    its specific Function, and Target Object."@en ;
  skos:prefLabel "Subject - Action - Object"@en ;
  skos:altLabel "Tool - action - product"@en .
```

Figure 5: Implementation Of Subject-Action-Object

The following items define the quality of the function and are thus allowed values in the *QualityOfFunction*.

- Additional Function
- Basic Function
- Excessive Function
- Harmful Function
- Ideal Function
- Insufficient Function
- Neutral Function
- Providing Function
- Technical Function
- Useful Function
- Transport Function
- Bad Controllable Function
- Redundant Function
- Missing Function

The explanation of those can be found in the book "Systematische Innovationsmethoden" [3] and in the paper from Nikolai Shchedrin [11].

The following two are introduced by Nikolaid Shchedrin and are a aggregation of quality of functions from the above list. These are also implmented as TRIZ concept.

- Function Disadvantage
- Useful Function Disadvantage

After running all five steps on the system the Functional Model is created. As the Component Model is small part of the Functional Model, this is mentioned as sub-concept. Furthermore the Function Modeling describes a process and rules for building the Function Model. Hence this is also a sub-concept of Function Model.

Furthermore for completion *process* is added as a TRIZ concept.

6.6 Optimizing The Function

After creating the Functional Model a Principle has to be applied to the function which need optimization. This is done by choosing an increasing and decreasing parameter from the chosen Matrix. These parameters have been implemented as *rdf:property*.

In a next step these properties are added to a chosen function and form a new concept, named *functionWithMatrixParameter*.

With the concept *ChooseMatrixEntryByFunction* the parameters are then checked and the suitable matrix entry is returned. As there are various Matrix, which differ slightly in their entries a new property *chosenMatrix* is added as an allowed value to *ChooseMatrixEntryByFunction*. With this the chosen Matrix can be mentioned.

This results in the *MatrixEntryByFunctionParameter* concept. With this and the implemented WUMM concept *ChoosePrincipleFromMatrixEntry* a Principle from the matrix entry is chosen. Finally the chosen principle is connected with the function by *PrincipleByMatrixEntryByFunctionParameter*.

6.7 The Matrix 2003

The Matrix 2003 has been created with the help of the python script, which has been explained in 5.1.2. To connect the principle id's from the Matrix 2003 with the implemented principles in the file *Principles.ttl* an id has been added. The property has been called **od:Matrix2003Id**.

6.8 The Parameters

As for the created Matrix 2003 more parameters and upper-parameter were added, the already created *Parameters.ttl* was updated.

The nine new parameters were created as TRIZ concepts.

Furthermore the six upper-parameters were also added as TRIZ concepts. For those the new WUMM concept *od:UpperParameter* was implemented.

7 Example For Functional Analysis

In the following section for four different systems the Functional Model will be implemented. With the given Functional Model the problems of the system can be obtained. Afterwards with the help of the various Matrices and the Principles solutions for these problems can be found. For some optimizable functions a principle will be looked up in the matrix, which will be visible in the examples. Some functions will not be optimized on purpose, as this should help to figure out own ideas and solutions.

For the following examples always one implementation of a function is described. The creation of the other functions in the example works analogously.

7.1 Example Washing Machine

The first examples describes the system of a washing machine. This example was used by Nikolai Shchedrin.

First the important system components *Washing Machine Drum* and *Laundry* were defined as an component. Furthermore the action *Turn* is implemented. With the two components the interaction *Washing Machine Drum - Laundry* is created.

At point three, as mentioned before, the function *CirculateLaundry* is created.

In the next step the direction for the function is added. This is done, by defining the Subject-Action-Object triple *Washingmachinedrum - Turn - Laundry*, with the interaction and the action. In this case *Washing Machine Drum* is the Subject and the *Laundry* is the Object. Afterwards this attached to the function.

In the final step we create the quality of the function. At this point the creator has to analyze the function and reason why this quality was chosen. Here, the quality was chosen as a *Useful Function*, as the turning of the laundry helps the cleaning. Furthermore it is assumed that the washing machine works perfectly in this aspect. Otherwise the function could have a harmful addition. The quality is then added as another triple to the function.

At this point the function has all important triples. The function in figure 6 then represents one part of the Functional Model.

```
ex:CirculateLaundry
  a tc:Function ;
  tc:SubjectActionObject ex:WashingmachinedrumTurnLaundry ;
  tc:QualityOfFunction tc:UsefulFunction ;
  skos:prefLabel "Ciruclate Laundry"@en .
```

Figure 6: Implementation Of The Function *CirculateLaundry*

7.2 Example Coffee

The next example is mentioned in the book "Systematische Innovationsmethoden". The example describes the system of coffee interacting with a cup when poured into it. A small

part of the Functional Model is implemented as a RDF graph.

At first the two components *Cup* and *Coffee* are implemented in *ex* namespace. With the two components the interaction *CupCoffee* is created. Afterwards the action *Rest* is added.

With the components and action defined, the function *Pollute*, which can be seen in figure 7 is created. This describes that the coffee leaves residues at the cup.

As before, the direction of the function is added with the Subject-Action-Object triple. In this case this is named *CupRestCoffee* and the Subject is the *Coffee*.

As the residue is a negative effect this would be a *Harmful Function*. Hence this function would be a starting point to find a Principles in the Matrix to solve this problem.

With the creation of further functions the Functional Model is fulfilled. At last all the functions with problems can be analyzed and a suitable Matrix Entry can be identified. From there a Principle is chosen, which optimizes the function.

```
ex:Pollute
  a tc:Function ;
  tc:SubjectActionObject ex:CoffeeRestCup ;
  tc:QualityOfFunction tc:HarmfulFunction ;
  skos:prefLabel "Pollute"@en .
```

Figure 7: Implementation Of The Function *Pollute*

7.3 Example Car

In the third example the system of a car is analyzed. Hereby the function *SitUncomfortable* is explained.

First of all the two components *UncomfortableSeats* and *Inmates* are created. The action is in this case *Touch* and the Subject-Action-Object triple will be *InmatesTouchUncomfortable-Seats*. The *Inmates* are the Subject as these are touching the seats.

The quality of the function is a *Function Disadvantage*, as the inmates are able to sit, which is a positive or useful aspect.

On the other hand the seats are uncomfortable for the inmates. This can be harmful as the driver can be disturbed by this. Furthermore, as the feeling of a uncomfortable seat differs for each person, this function is bad controllable.

```
ex:SitUncomfortable
  a tc:Function ;
  tc:SubjectActionObject ex:InmatesTouchUncomfortableSeats ;
  tc:QualityOfFunction tc:FunctionDisadvantage ;
  skos:prefLabel "Uncomfortable Sitting"@en ;
  skos:Definition "By touching the occupants with the uncomfortable seats of
    the car, a useful function forms with disadvantages, since the occupants
    on the one hand can sit but only uncomfortably"@en .
```

Figure 8: Implementation Of The Function *SitUncomfortable*

7.4 Example Mechanical Computer Mouse

The last example is taken from the book "Systematische Innovationsmethoden". Picture 9 shows an overview over the Functional Model. This picture was created by the book authors [3]. The different arrows represent different kind of qualities for the function. The black arrows are Useful Functions, the red ones describe Harmful Functions and the dotted lines represent a Bad Controllable Function.

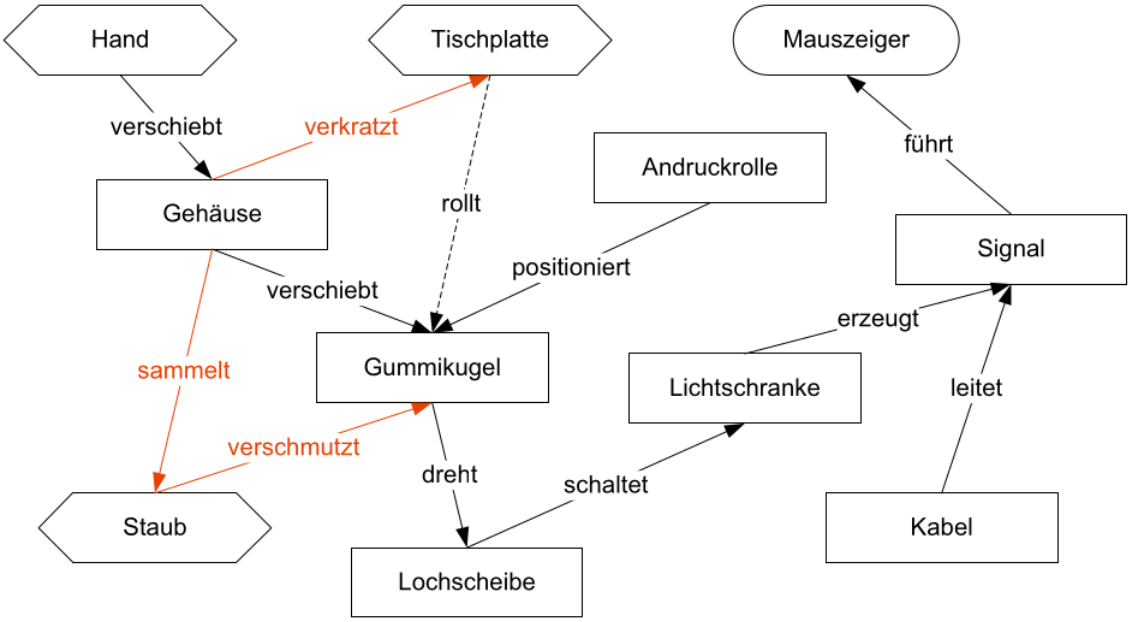

Figure 9: Functional Model For A Mechanical Computer Mouse In German

The implementation of the function *Scratches* is going to be explained. This describes the arrow from *Case (Gehäuse)* to *Tabletop (Tischplatte)*.

The interaction is therefore *CaseTabletop* with the two components *Case* and *Tabletop*. All of these are in the ex namespace as explained in section 3.

Furthermore the action *Move* is defined. And with that the direction *CaseMoveTabletop*, by

the Subject-Action-Object triple created.

Finally the quality of the function is added. The book defined this as a *Harmful Function*.

As this is an Harmful Function an optimization is necessary. Therefore we define *Scratches-Parameter* which is a *functionWithMatrixParameter* concept. Hence we have to choose the decreasing and increasing parameters. As in this paper also the Matrix 2003 was implemented. This is the one chosen to find a suitable principle.

As an increasing parameter *Power* is chosen, because the moving of the mouse increases the working possibilities of the mouse. The decreasing parameter is *Loss Of Material*, as the tabletop looses material during the moving of the mouse.

With the chosen parameters the matrix entry can be obtained. In this case this would be *<http://opendiscovery.org/rdf/Matrix/E.18.25>*.

From there the principle *tc:FlexibleCoversOrThinFilms* is picked.

This should lead to an optimization for the function. Here it could be adding a layer between the tabletop and the mouse, like a mouse-pad.

8 Conclusion

This paper implemented a suggestion for an ontology for Functional Analysis. Most of the information were taken from the book "Systematische Innovationsmethoden" and the talk from Nikolai Shchedrin. This gives a good summary over all the concepts which are need for a Functional Model and the Functional Analysis. In a next step other sources could be added and the ontology could be advanced.

Furthermore the examples could be made more specific, which would help for a greater understanding on the topic. An interesting aspect would also to see multiple approaches on how to optimize a function with different principles.

As there is another update of the Matrix another implementation for this would also be helpful.

In conclusion this implementation should give a good starting point for the ontology for Functional Analysis, but could be advanced with more knowledge and ideas from other sources.

References

[1] Genrich Altshuller (1979). Creativity as an exact science (in Russian). English version: Gordon and Breach, New York 1988.

[2] Hans-Gert Gräbe (2021). About the WUMM modelling concepts of a TRIZ ontology. https://github.com/wumm-project/Leipzig-Seminar/blob/master/Wintersemester-2020/Seminararbeiten/Anmerkungen.pdf.

[3] Karl Koltze, Valeri Souchkov (2017). Systematische Innovationsmethoden (in German). Hanser, Munich. ISBN 978-3-446-45127-8.

[4] Alex Lyubomirsky, Simon Litvin, Sergei Ikovenko et al. (2018). Trends of Engineering System Evolution (TESE). TRIZ Consulting Group. ISBN 9783000598463.

[5] Nikolay Shpakovsky (2016). Tree of Technology Evolution. English translation of the Russian original (Forum, Moscow 2010). https://wumm-project.github.io/TTS.html

[6] SKOS – The Simple Knowledge Organization System. https://www.w3.org/TR/skos-reference/.

[7] The WUMM Project. https://wumm-project.github.io/

[8] Matrix2003: https://triz-journal.com/wp-content/uploads/2018/04/Screen-Shot-2018-04-30-at-15.20.25.png.

[9] tabulapdf: https://github.com/tabulapdf/tabula.

[10] Deep Translator: https://pypi.org/project/deep-translator/.

[11] Nikolai Shchedrin (2020). Webinare des TRIZ-Ontologie-Projekts "Funktion" und "Funktionsanalyse".

[12] Valeri Souchkov (2018). GLOSSARY OF TRIZ AND TRIZ-RELATED TERMS .

[13] Nikolai Shchedrin (2020). https://triz-summit.ru/onto_triz/mod/metod/triz/fa/model_fa/func_syst_model/func/func_type/disadv_f/.