TRIZ*fest*-2021

Semantic Web Modelling of TRIZ System Evolution Concepts

Tom Strempel*, Hans-Gert Gräbe*

*Computer Science Institute, Leipzig University, Germany

Abstract

Since Altshuller's formulation of eight evolution laws of technical systems [1], the topic has developed into a weighty TRIZ tool, primarily on a speculative-empirical basis [3,4], to classify or predict development directions of technical systems. In [6], Shpakovsky presents a systematic attempt to assemble such development germs into the more complex structure of an evolution tree. In this paper, we report on the transformation of these approaches and results into a formal syntax based on RDF. The required formal remodelling especially of concrete examples clarifies the strengths and weaknesses of the concepts developed in [6]. The modelling project is part of our contribution [10] to the TRIZ Ontology Project [11].

Keywords: *Evolution of Technical Systems, Evolution Tree, Semantic Web, TRIZ Ontology Project*

1 Aim of the work

The aim of this paper is to present an ontological modelling of the areas of *TRIZ System Evolution Concepts* based on the approaches in [4] and [6] and further own considerations. The work fits into the activities of the TRIZ and WUMM Ontology Projects [10,11] to model core TRIZ concepts using modern Semantic Web means. We provide two kinds of result – a formally described SKOS based body of notions and formal models of three examples taken from the literature (available in the github repository [8]), and this paper, in which the background and motivation of the modelling decisions are informally described in more detail. Due to space restrictons, we can only take up some essential moments of the modelling and discuss some aspects of two of the three examples. For a more detailed version of the material see the seminar paper [9].

7 Evolution Concepts for the Development of Technical Systems

2.1 TRIZ and the Evolution of Technical Systems

The basic practical approach of TRIZ is the development of technical systems (TS) through transformation in the course of problem solving. These practices of changing real-world systems (including their design, implementation and reconstruction) for dedicated special purposes generate a spectrum of highly contradictory real-world developments, for the systematisation of which appropriate principles must be developed. With the formulation of *8 development laws* of TS [1], Altshuller proposed principles of such a systematisation, which have

largely resulted from the study of patent documents and other practical engineering experience on the background of system-theoretical approaches. The authors of [4], where further developments of this approach are discussed, are more cautious and call the laws *trends* in view of their low theoretical foundation. 10 trends are discussed in [4] based on a wealth of examples, where each of the trends is related on a one-dimensional *less-more* scale of a "trend direction". Assuming the possibility of quantifying each of the scales, a 10-dimensional "development space" emerges with multiple questions not addressed in [4], of which the following is one of the simpler ones: Can progression in one of these 10 dimensions be related with regression in another?

2.2 Evolution Trees

Similarly argues Shpakovsky in [6]. He also extracts – largely from experience of the differences in the usefulness of TRIZ principles in practical inventive projects – 10 basic patterns of evolution of TS. Each of these basic patterns is further refined into a sequence of modification subpatterns of graded intensity (see [6: ch. 3]), and this conceptual toolkit (*basic evolution tree*) is applied as a *methodological basis* to the systematisation of real-world technological development in the form of *special evolution trees*.

Unlike [4], in [6: ch. 4] a clear principle is proposed according to which TS are brought into an evolutionary context in such considerations: The basis for any such investigation is a sufficiently general *elementary technical function* (ETF) [6: ch. 4.1]. Only those (sufficiently generalised, [6: p. 122]) TS are included in the investigation that realise this ETF as an emergent function. I refer to this selection in the following as *class of technical systems* (CTS). Its elements are called *objects* in [6]. Each CTS delimited on this basis is doubly contextualised by the choice of the ETF and the degree of generality of the technical systems under consideration. The delimitation should fulfill the following conditions [6: p. 122]:

1 To organise information, a tree-like structure is used that allows visual presentation of descriptions of *all basic* known versions of an object under examination.

2 The evolution tree is an organized set of objective evolution patterns based on the analysis of the evolution of many technical systems. Hence the construction of evolution trees suggests use of an *objective classification criterion*.

3 Every evolution pattern includes a set of generalized descriptions of transformation versions and transitions between them and may be illustrated by a transformation example of a specific technical object. Hence the *requirement of generality and specificity is satisfied*.

4 Information presentation in the form of a tree-like structure allows a designer to see all the basic transformation versions simultaneously and to distinctly *trace their structure*.

5 The availability of the basic tree allows foreseeing all *significant transformation versions* even if the information available on the versions of a system under consideration is scant or fragmentary.

This delimitation is the first step in a sequence of 8 steps [6: ch. 4.3] that Shpakovsky proposes to base his construction of evolution trees on.

1 Determining the elementary function performed by the object of interest, clarifying and formulating its role in the performance of this function.

2 Collecting information on similar objects which either are known to perform the same role in the realization of the same elementary function, or can be adapted to the per-

formance of this function. Making a short description of each modification of the object, paying special attention to the essence of the transformation which resulted in the appearance of this modification. Finding the initial transformation version of the object, the simplest one in terms of the technological evolution.

3 Selecting the main evolution pattern — the trunk of the future tree. It may be any of the evolution patterns, but using those patterns where transformations of components are especially significant, such as «Segmentation of Objects and Substances» or «Mono-Bi-Poly» would be more convenient. Building the main evolution pattern, the frame of the future Tree by placing cards with the description of corresponding versions of the object under consideration.

4 Constructing of second-order evolution patterns keeping to the following rule: constructing dynamization patterns of object modifications if possible; if it is impossible to obtain dynamization resources, first building the patterns which provide resources — «Mono-bi-poly», «Segmentation» and «Expansion».

5 Checking whether it is possible to build second-order patterns which describe transformations of object's shape, surface and internal structure. These patterns are: «Geometrical Evolution», «Internal Structure Evolution» and «Evolution of Surface Properties». To optimize the tree structure, it is better to add these patterns only if they reflect object transformations which are important for subsequent analysis.

6 Checking whether it is possible to build third-order patterns — «Dynamization» — after the «Mono-Bi-Poly», «Segmentation», «Expansion» patterns. Constructing these patterns in significant and indicative places of the tree.

7 Constructing the «Increasing Controllability» patterns placing them after the Dynamization patterns. These patterns should only be built for characteristic and significant cases of controllability. For all other cases, the controllability of objects is clarified by analogy. Building the «Increasing coordination» patterns in characteristic and indicative places of the Tree.

8 Carrying out an additional information search, supplementing and specifying the tree structure.

2.3 Objective of our work

Shpakovsky thus proposed a systematic-methodical approach to the study of the evolution of TS, which goes beyond previous approaches, and demonstrates the practical performance of this approach in a number of examples.

The aim of the work presented here is to prepare this methodological approach for a Semantic Web formalisation within the scope of the WUMM Ontology Project [10]. With regard to the explanations in [2], we limit ourselves to a formalisation of the taxonomy (conceptualisation, basic tree as *evolution tree ontology* – ETO) and show how this can be used in the formalised representation of special evolution trees. In [2], also the reasons are explained in more detail why we base this work on SKOS as meta model and not on OWL.

8 Conceptualisation

The conceptualisation to be developed follows the basic assumptions and settings that are explained in more detail in [2]. In particular, the following namespace prefixes are used:

- ex: – the namespace of a special CTS to be modelled.
- tc: – the namespace of the TRIZ concepts.

- od: – the namespace of WUMM's own concepts.

Furthermore the SKOS ontology is used to model labels and definitions of the object.

Our central task is to model the nodes and edges of a given CTS evolution tree. The full graphical representation of that evolution tree as an edge-marked graph then can be reconstructed from that set in the usual way (actually, the two representation forms are equivalent to each other.).

The interaction between the special CTS modelling and the basic constructs of the ETO is explained here using code from the CTS *DisplayDevelopment*. An edge in a CTS evolution graph has the typical shape of an RDF sentence, e.g.

```
ex:TVWithLargePixels ex:decreasePixelSize ex:TVWithMediumPixels .
```

This sentence addresses the development from *TV with large pixels* to *TV with medium pixels* that have a better performance in brightness and sharpness of images. The code of the two nodes is not presented here, we only note that the introduced URIs have nothing directly to do with the semantics of the represented TS except that – following the modelling recommendations of the Semantic Web – "speaking names" are used. To the RDF predicate `ex:decreasePixelSize` further information is attached.

```
ex:decreasePixelSize a rdf:Property, skos:Concept ;
      od:usesPattern tc:SegmentationPattern ;
      skos:prefLabel "Decrease pixel size"@en ;
      skos:definition """Decrease pixel size by segmentation of one
            big pixel in several smaller ones"""@en .
```

SKOS label and definition describe the transformation in the CTS in more detail, `od:usesPattern` refers to the pattern from the ETO that was applied in this transformation.

Although RDF graphs are an important RDF concept and multiple RDF graphs can be stored in and retrieved from an RDF store, it is difficult to represent graphs as delimitable objects at the level of RDF triples. We therefore store each specific CTS graph in a separate file. The file contains the description of the nodes and edges of this graph as well as an instance of `tc:EvolutionTree` with the global properties of the graph. Each such graph also has its own namespace, which can also be used to identify the parts of the graph.

9 Modelling the Evolution Tree Ontology (ETO)

This section describes how the concepts from [6] are modelled in our ETO.

The input of an ETO modelling of a CTS is the CTS itself, which is delimited according to contextual parameters (*goal* and *scope* of the modelling, determination of the *ETF*, determination of the *level of abstraction* of the TS to be included in the CTS, see 2.2) and the given methodology [6: ch. 4.3]. This delimitation is taken as given in our modelling. Essential context parameters can be stored in the global object of the graph file.

```
Ex:DisplayEvolution a tc:EvolutionTree;
      rdfs:label "Evolution of TV and Computer Displays"@en;
      dcterms:source "Shpakovsky's book"@en ;
      od:hasETF "visualize information"@en ;
      rdfs:comment """A display is an artificially created object
```

```
        specially designed as a tool to realize the function
        «To visualize information»"""@en .
```

9.1 Evolution Patterns and Modification Subpatterns

As essential structuring elements for evolution trees, ten *basic evolution patterns* and modifying subpatterns were introduced in [6]. These ten basic patterns are:

1 Mono-Bi-Poly
2 Trimming
3 Expanding-trimming
4 Segmentation
5 Geometrical evolution
6 Object structure evolution
7 Evolution of surface properties
8 Dynamization
9 Increasing the controllability
10 Increasing the coordination of the elements

For each of these basic evolution patterns, a sequence of more specific *modification subpatterns* is specified. The state of the development along the basic evolution patterns 1–4 constrain the application of other evolution patterns. For example, there is no possibility for dynamization on an unsegmented monolith. The structure of the object is addressed by patterns 5–7. Patterns for dynamization, controllability, and coordination are applied at points that seem reasonable. It is not required to follow the sequence of modification subpatterns of a basic pattern to its end before applying a different basic pattern.

In [6] it is mentioned several times that evolution deals with the development of an object from the CTS. We follow the usual approach in TRIZ ontology modelling that distinguishes between old and new object instead of working with object modifications. All evolutionary transformation steps are therefore modelled according to the pattern

OldObject → isTransformedInto → NewObject.

Note that the concept of an evolution tree is a self-similar concept. An evolution tree thus can be related to an evolution tree with of one of its objects as root expanding this object to another CTS at a different abstraction level. E.g. the evolution tree of the display can be related in such a way to the evolution tree of a plasma screen, which could be analysed further.

9.2 Modelling Evolution Tree Concepts

The file *EvolutionTree.ttl* [8] contains the formal description of the basic evolution patterns and thus the basic evolution tree as developed in [6], which is in a second step – as application of the formalization and proof of concept – applied to create formal models of three special evolution trees.

Each basic pattern and modification subpattern is represented by a special URI. Conceptual relations between these patterns and subpatterns are modeled using the (inverse to each other) predicates `od:subConceptOf` and `od:hasSubConcept`. E.g. the segmentation pattern is represented by the URI `tc:SegmentationPattern` and has the following code in Turtle notation.

```
tc:SegmentationPattern a skos:Concept, od:AdditionalConcept ;
      od:subConceptOf tc:BasicEvolutionPattern ;
```

```
od:hasSubConcept tc:Monolith, tc:TwoParts, tc:ManyParts,
      tc:Granules, tc:Powder, tc:Paste, tc:Liquid,
      tc:Foam, tc:Fog, tc:Gas, tc:Plasma, tc:Field,
      tc:Vacuum, tc:IdealObject ;
skos:prefLabel "Segmenting objects and substances"@en ;
skos:example "Segmentation of an aircraft propulsion unit"@en .
```

In the given example `tc:SegmentationPattern` is a subconcept of `tc:BasicEvolution-Pattern`. Different modification patterns like `tc:Liquid` are also formalised in that way.

```
tc:Liquid a skos:Concept, od:AdditionalConcept ;
      od:subConceptOf tc:SegmentationPattern ;
      skos:prefLabel "Liquid"@en .
```

Different to [6] certain subpatterns as `tc:FlatSurface` and `tc:CylindricalSurface` of the generic evolution pattern `tc:GeometricalEvolutionPattern` are not put in a mutual subconcept relation since transformations in both directions appear in specific examples. E.g., some modern monitors use curved displays instead of flat ones, whereas older CRT displays have a cylindrical surface due to constrains in manufacturing. Using better glass newer CRT displays have a flat surface. Shpakovsky also introduces the MATChEM-Operator from the wider TRIZ context as extra pattern, not listed in the basic ones.

9.3 Construction of Evolution Trees

Shpakovsky emphasises in [6] that the construction of an evolution tree is mostly an iterative process in the course of which the goal, ETF, scope and degree of abstraction of modelling the CTS are gradually refined. Our tools for formal descriptions support this iterative process, as new objects can easily be added as nodes and transformation steps can be added or modified as edge descriptions.

With the description elements presented so far, some of the more advanced concepts from [6] cannot yet be adequately represented. This is especially the case for the concepts *trunk* and *branch* of a CTS tree, which, however, remain vague not only from a graph-theoretical perspective.

The concepts *trunk* and *root* attempt to address the development in a CTS from simpler to more complex forms, which is mainly oriented towards the unfolding of the ETF and associated with the basic patterns 1–4. However, since the modification sequences for each basic pattern define *branches* in the tree, even in such a linear context it is unclear which of these branches is the trunk. In the ETO, a language element can easily be added that identifies transformation edges as belonging to the trunk. However, it is not clear that this results in a linear rather than a branched structure.

However, this is a general conceptual problem – the basic constructs only guarantee that the evolution is described by a *directed graph*. Even the property that the emerging graph is *acyclic* requires additional preconditions. An acyclic graph is characterised by the fact that its nodes can be placed in a linear order that coincides with the edge directions. This can be achieved, for example, assigning timestamps to the objects, but this poses restrictions on the abstraction principle applied in the constitution process of the objects of the CTS.

It also remains largely unclear why evolution graphs should necessarily be *trees*. Major advances in general technical development are characterised precisely by the fact that there meet

several lines of development. Such phenomena cannot be conceptualised with a pure tree based approach alone.

9.4 Determination of yet Unknown Versions

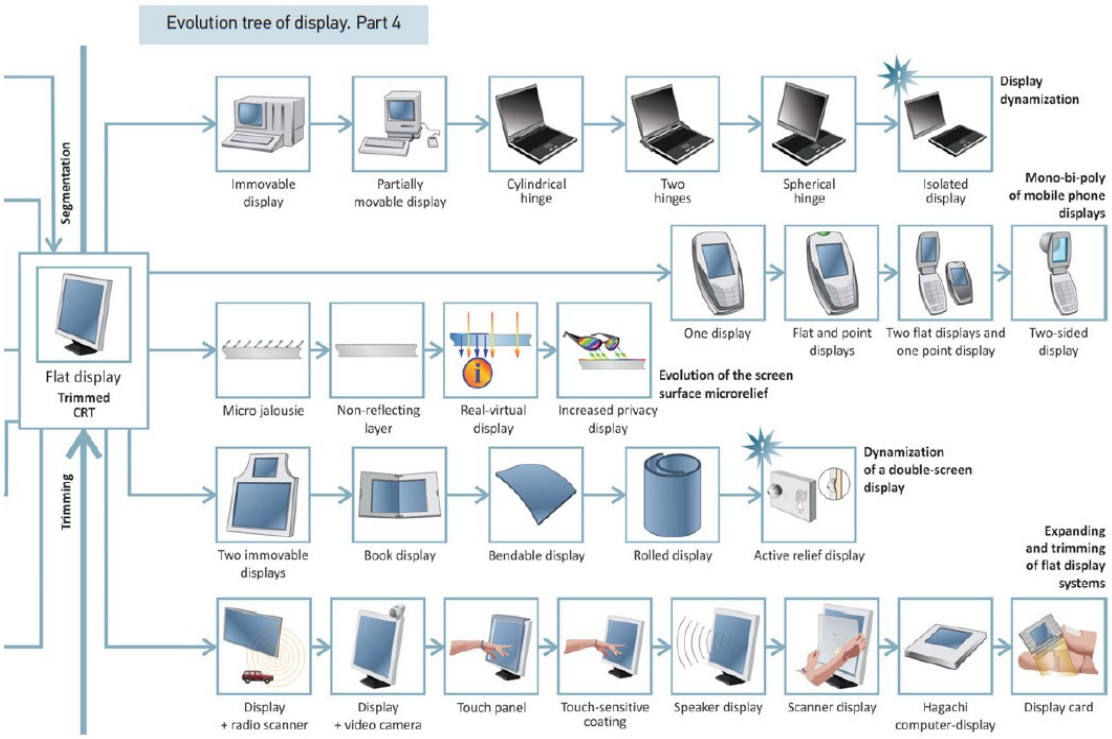

Figure 1: Section of the specific tree of *DisplayEvolution* [6].

For the analysis of a CTS, both the basic (see [6: Fig. 4.78]) and the specific evolution tree (see Fig. 1) must be created. By comparing the two trees, gaps as well as not yet realised evolution patterns can be discovered.

The highest modification level of the dynamization pattern is a complete decoupling of the individual components. For a laptop, this means to separate display and peripherals. Around 2002, at the time the evolution tree of the display [6] was created, this version did not yet exist in the CTS but the gap could be identified and the evolution option formulated. Nowadays, complete dynamization is achieved integrating the computing technology into the display and connecting the peripherals via Bluetooth. This shows that evolution tree analysis is in principle capable to predict such future technological developments.

10 Modelling Examples of Specific Evolution Trees

10.1 Modelling the Evolution Tree of the Display

Shpakosvky modelled the evolution tree of the display with *To visualize information* as ETF. The abstraction level to include objects in the CTS is given by the definition of a display as an *artificially created object specially designed for the role of a tool in the realization of the elementary function* [6], thus ruling out a sheet of paper with information written on it. The main axis of development, i.e. the trunk of the tree, runs along trimming transitions from the cinematographer, trimmed cinematographer, CRT TV set to the flat display. Further transitions apply the segmentation pattern. The evolution tree trunk is marked adding a

```
od:usesPattern tc:EvolutionTreeTrunk
```

statement to the corresponding transition edges. As the granularity of this specific evolution tree is very finegrained some transition patterns can be applied multiple times for object transformations.

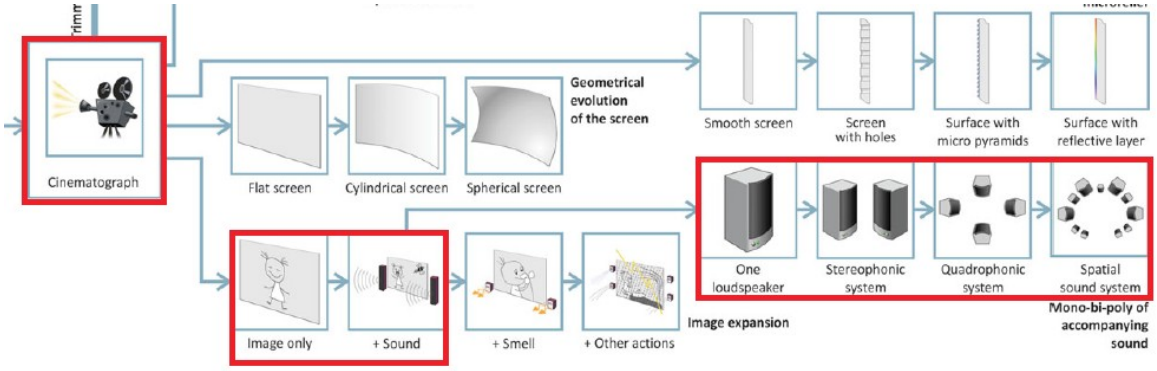

Figure 2: Pattern of adding audio [6]

We describe the code of the transformation for adding sound to the display (see Fig. 2) as an example for the structure of the modelling done in *DisplayExample.ttl* [8]. `ex:` is used as the namespace because a real-world example is described. We choose the cinematographer from the evolution tree trunk as the starting point of our example.

```
ex:Cinematograph a ex:Screen ;
      ex:transitionsTo ex:ImageOnly, ex:FlatScreen,
            ex:SmoothScreen, ex:ImmovableScreen ;
      ex:trimCinemaBuilding ex:MechanicalTVSet ;
      skos:prefLabel "Cinematograph"@en .
```

A word about the Turtle notation used here, which compactly combines all RDF triples containing the RDF subject `ex:Cinematograph`. This code contains, among other things, the transition triples starting at the cinematograph object which describe the transitions into the different branches in Figure 2. They expand into the RDF triples

```
ex:Cinematograph ex:transitionsTo ex:ImageOnly .
ex:Cinematograph ex:transitionsTo ex:FlatScreen .
ex:Cinematograph ex:transitionsTo ex:SmoothScreen .
ex:Cinematograph ex:transitionsTo ex:ImmovableScreen .
```

The transitions are all described by `ex:transitionsTo` and represent the transition of the same object *cinematograph* into the initial positions of the different branches. A uniform predicate is used here, since the respective transformation does not change the object, but only its perception for the further development in the respective branch – the ImageOnly perception is extracted for merging with audio, the FlatScreen perception for further development of curved surfaces, the SmoothScreen single-layer perception for the addition of further layers and the ImmovableScreen perception for further development towards portable units (not shown in figure 2).

Each of these transformations defines a new object in the CTS that may serve as root of a evolution subtree, so that we can also interpret the situation as merging four evolution trees into one with the new root in the *cinematograph*. However, such transformations of whole evolution trees and thus also of the contextualisations given by their CTS are neither discussed in [6] nor so far conceptually supported by our semantic modelling.

Somewhat different is the fifth transformation

```
ex:Cinematograph ex:trimCinemaBuilding ex:MechanicalTVSet .
```

of the cinematograph into a mechanical TV, which also includes a transformation of the technical object itself.

Further development in the addSound branch of the tree is described by the transformation

```
ex:ImageOnly ex:addSound ex:ImageSound .
ex:addSound a rdf:Property, skos:Concept ;
      od:usesPattern tc:MonoBiPolyPattern, tc:BiSystem ;
      skos:prefLabel "Add sound"@en .
```

adding to the image a sound track thus transforming the `ex:ImageOnly` object into the `ex:ImageSound` bisystem. The further development yields another branching

```
ex:ImageSound ex:addSmell ex:ImageSoundSmell .
ex:ImageSound ex:transitionsTo ex:OneLoudspeaker .
```

adding a smell track to the bisystem on the one branch and joining bisystem with the audio development track thus refocussing on audio development on the other. However, the latter is problematic for the concept developed in [6], because it softens both the ETF and the contextualisation.

In a similar way the whole evolution tree of the display is transformed into a formal model. One particularity must be explained concerning the further segmentation of the display. Shpakovsky uses in that example not only generic evolution patterns but also specific ones. This was modelled introducing additional model-specific patterns (`ex:ManyParts`, `ex:Sand` etc.) in the `ex:` namespace and the model-specific `ex:segmentation` predicate.

10.2 Modelling the Ship Propulsion Evolution Tree

Souchkov describes in [3] another evolution tree using the example of the boat evolution, see Fig. 3. The terms boat and ship are used interchangeably here even if a ship is assumed to have some other characteristics as a boat, e.g. being ocean-going and having a higher displacement.

This graph representation of an evolution tree differs from the example in 5.1 in that it was not created on the special conceptual basis [6]. Nevertheless there was no problem to prepare the material according to Shpakovsky's principles, enriched and transferred into a formal model (see *BoatExample.ttl* in [8]). The nodes labelled in Fig. 3 in italics are “dead ends” whose development was not continued and which are no longer in active use today. They are marked model-specific as `ex:DeadEnd` in our modelling.

A *main axis of development* is already given by the nodes labelled in bold, which thus forms the *tree trunk*. The tree heavily branches and also contains parts in which the boat function is no longer dominant, but is used in combination with other functions in bi- and polysystems. The end of the development line *Mono-Bi-Poly* is the transition to a “monosystem on a higher level” [6: p. 184] through integration of the partial functions in the polysystem to a new emergent ETF on the level of the supersystem (listed as T*rend of Transition to the Supersystem* in [4: ch. 4.4]). Such developments, for example from the boat to the military boat in Fig. 3, are not modelled in this complexity in [6], because from the specific context perspective of the CTS, it is not the emergent new ETF that is of interest, but only the contribution that the old ETF makes to it. However, Souchkov's diagram has probably also to be understood in this

way, because in that context only the boat property of the military boat is of interest, but not its combat properties, which emerge from the interaction of many sub-functions. However, this is only our assumption; details are not explained in [3].

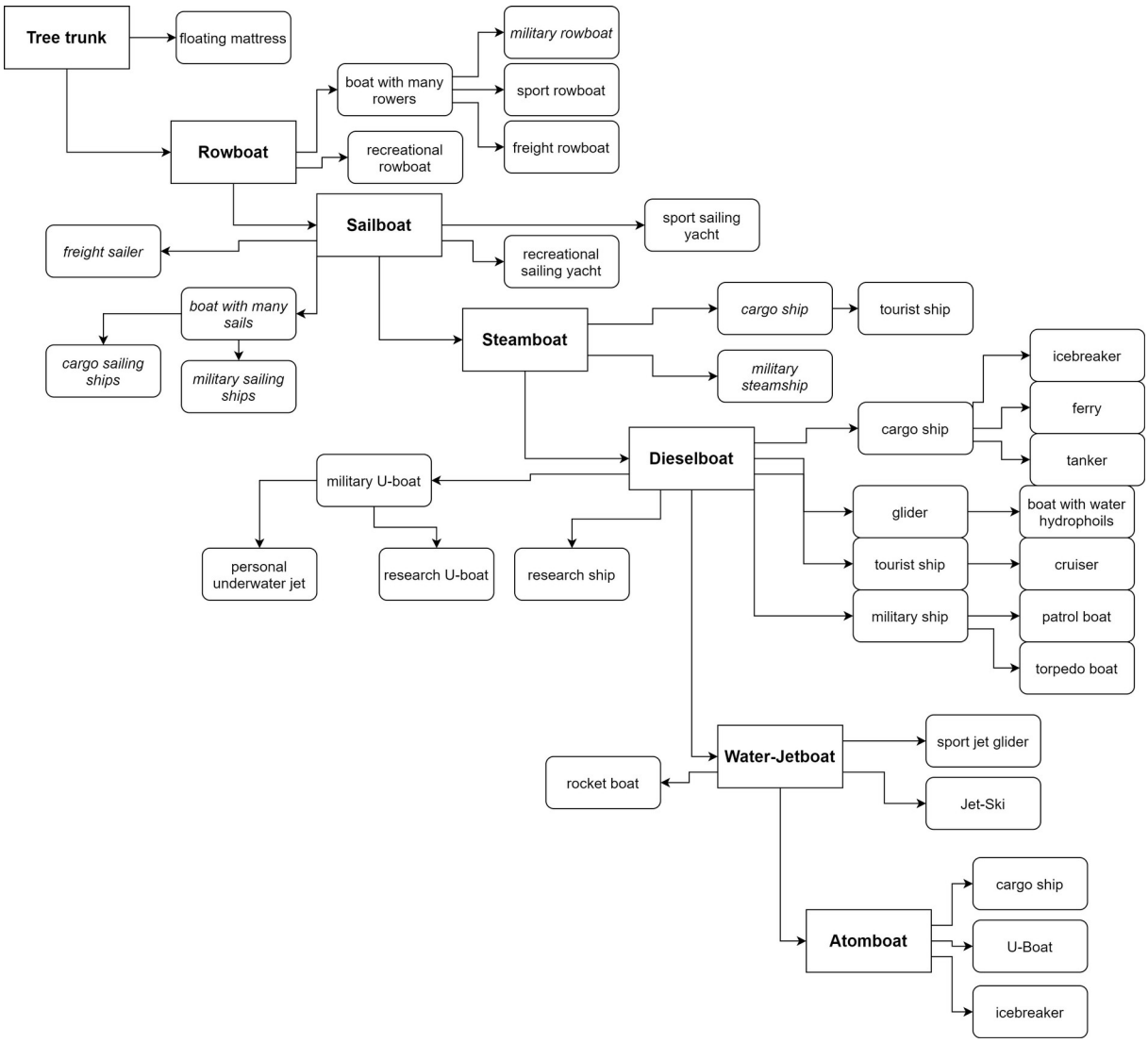

Figure 5: Souchkov's boat evolution tree [3: p. 162], own diagram

The next step is to *specify the ETF* of the CTS model. The objects grouped in this CTS cover a wide range of functionalities and transformations, making it difficult to identify the goal, scope and ETF of the modelling. Souchkov splits the transformations into three categories: New transformations for delivering the main function, existing transformations that could be developed further and completed or discontinued transformations. We are interested in the new transformations for delivering the main function as this defines the main axis of development – tree trunk, rowboat, sailboat, steamboat, dieselboat, waterjetboat and atomboat. Hence we define the ETF as to *provide the boat with engine and power source* since all these transformations, with one exception, focus on the engine and power source. A tree trunk has no power source, a rowboat uses muscle power, a sailboat the wind, a steamboat a steam machine and so on. As the waterjet is a means of propulsion and thus the transformation does not focus on the power source, but how the power is used for propulsion (e.g. propeller, paddle wheel), it should probably be skipped from the evolution tree trunk and a direkt edge between dieselboat and atomboat should be added.

The granularity of this tree is very coarse thus imposing a high degree of abstraction in the definition of the objects of the CTS. This abstraction is also not oriented towards temporal sequences (even in the age of atomboats, there are still rowboats), hence a timestamp based acyclicity condition as mentioned above cannot be implemented in such a context. Moreover, branching from the trunk does not necessarily follow Shpakovsky's modelling rules, because, for example, the transition steam boat → cargo ship seems to be more as a taxonomic relation general → special than a real technological evolution (according to the Mono-Bi-Poly pattern; the boat receives the additional function "transport of goods", but this is a function from the supersystem). We can already see from these considerations which problems arise with the specification of an initially vague CTS modelling of an evolution tree, as it is given with Figure 3 alone ([3] does not contain any further explanations on the background of the modelling decisions). In the many repetitions of nodes with label "U-Boot" (probably better translated as "submarine") or cargo ship in fig. 3, another complex evolution pattern becomes evident: The evolutionary lines of TS, which are components in a common supersystem, are closely connected through synergy effects, resulting in a close interrelation structure between the evolutionary trees of these two TS, whose specific networking effects cannot be grasped from the perspective of one of these trees alone. This requires new conceptual approaches of an integrative view of technology development.

A particular difficulty is the modelling of the transformations on the trunk of the evolutionary tree, as different new propulsion technologies are introduced in each case and this technological development does not take place at the level of the boat, but at the level of one of its components. The change within the component is accompanied by a complete reorganisation of the way how the interaction between the sub-functions of the different components of boat components works to deliver the emergent ETF of the boat. A corresponding complex reconstruction pattern is missing in Shpakovsky's list. We therefore again use a model-specific pattern

```
ex:BoatMATChEMOperator a tc:SpecificEvolutionPattern ;
      skos:prefLabel "Boat specific MATChEM operator pattern"@en .
```

at this point. The details cannot be presented here, see [9]. It is quite possible to generalise such a suitably defined model-specific pattern, to adopt it for addition to the ETO in the course of further ontology development and to declare the model-specific pattern as subconcepts of such a more general pattern.

11 Summary

In this paper we have shown how the concepts of building and refining evolution trees presented by Shpakovsky in [6] can be formalised and transformed into a Semantic Web format. We restricted ourselves to the formal modelling of the tree structure, the process dimension of the methodological system proposed in [6] remains informal. The reasons for this are explained in [2] in more detail.

Nevertheless, this processual methodological dimension is also to be applied in the formal remodelling of concrete examples. The strict requirements of formal modelling are predestined to reveal inconsistencies and weaknesses in the conceptual system of evolution trees. This is demonstrated by two examples from [6] and [3]. In particular, the formal refinement of the vague specifications in the latter example, which is given as a graphic only and was (presumably) not created according to Shpakovsky's methodology, proves both the applicability of Shpakovsky's methodology to problems from other sources and shows which detailed questions have to be solved in the systematic design of coherent evolution trees if one wants to go beyond the limits of purely speculative compilations.

12 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). The WUMM Project on a TRIZ Ontology. Basic Concepts. https://wumm-project.github.io/Texts/WOP-Basics.pdf.

3 Karl Koltze, Valeri Souchkov (2017). Systematic innovation methods (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 (2003). One of the evolution trends of an aircraft propulsive device. http://www.gnrtr.com/Generator.html?pi=211&cp=3.

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

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

8 Tom Strempel (2021). Code of the RDF Modelling of Evolution Trees. Directory Ontologies/EvolutionTrees in the github repo https://github.com/wumm-project/RDF-Data.

9 Tom Strempel (2021). A Proposal for Modelling TRIZ System Evolution Concepts. https://wumm-project.github.io/Texts/WOP-EvolutionTrees.pdf.

10 The WUMM Project. https://wumm-project.github.io/.

11 The TRIZ Ontology Project. https://wumm-project.github.io/Ontology.

Paper category: Research

Corresponding author: Hans-Gert Gräbe, graebe@informatik.uni-leipzig.de