Improving Taxonomy Maintenance: Automated Splitting and Merging of Taxonomies

Moritz Flöter[2], Pascal Reuss[1,2], Johannes Ude[2], and Klaus-Dieter Althoff[1,2]

[1] German Research Center for Artificial Intelligence
Kaiserslautern, Germany
http://www.dfki.de
[2] Institute of Computer Science, Intelligent Information Systems Lab
University of Hildesheim, Hildesheim, Germany
http://www.uni-hildesheim.de

Abstract. In Case-Based Reasoning (CBR), taxonomies are often used to model similarities. For complex domains and tasks such taxonomies tend to increase in size making them hard to model and maintain. This especially holds true if a group of people is working on the same taxonomy simultaneously. In this paper, we propose a solution by dividing larger taxonomies into sub-taxonomies, which can be regarded as individual and independent taxonomies. Through compilation, they can be merged and put back into their original context.

1 Introduction

For structured CBR systems, taxonomies are a relatively efficient way of defining similarities between different values of a certain attribute. This is achieved by ordering the possible values in hierarchical form as nodes of a tree and assigning similarity values to each node [2][3]. Beginning from the most general descriptor for the attribute in the root node, the values on each child node get more and more specific towards the leaf nodes of the tree. The similarity values follow that pattern usually starting at 0.0 for the root node and ending at 1.0 for the leaves. The value of each node serves as its unique identifier as the calculation of similarities has to be consistent. Once the tree is complete, the similarity between two different known values of the same attribute is calculated by choosing the closest common predecessor in the tree-structure. In case the attribute-values are identical or if the query contained a more general attribute-value that is a direct predecessor to the value in a case from the case-base, the resulting similarity is 1.0. It should be noted that the same concepts for similarity-calculation remains applicable in our approach for a compiled taxonomy.

Modeling taxonomies can be difficult and can cause a high effort, especially when modeling hundreds of values in a single taxonomy. For example, a taxonomy of aircraft systems may contain several thousand values from components (e.g. cabin) to systems (e.g. in-flight entertainment system), to system parts (e.g. audio system) to so-called Line-Replaceable-Units (LRU, e.g. display, speaker). Modeling each system or system part in an individual taxonomy would reduce

In: M. Leyer (Ed.): Proceedings of the LWDA 2017 Workshops: KDML, FGWM, IR, and FGDB.
Rostock, Germany, 11.-13. September 2017, published at http://ceur-ws.org

the modeling and maintenance effort, but similarity computation between values of different taxonomies will become very difficult. Therefore, splitting a big taxonomy into smaller sub-taxonomies solves just a part of the effort problem. To compute the similarities between values of different sub-taxonomies, the relevant sub-taxonomies should be merged during retrieval. This way, the modeling and maintenance effort is reduced, while still allowing a comparison of values from different sub-taxonomies.

The motivation for this work has its seeds in the OMAHA research project[4]. In this project we modeled taxonomies for aircraft systems with several thousand nodes. A taxonomy with three subsystems has 7668 nodes. Maintenance of this taxonomies is only possible with high effort. Therefore, the idea is to split up the big taxonomies into several smaller taxonomies with only dozens of nodes. This way we could maintain the smaller taxonomies with less effort. To have the same effect on similarity as the origin taxonomy, the smaller taxonomies have to be merged for similarity measurement.

For the rest of this paper, we will regard taxonomies from their structural point of view, namely trees. When splitting up this structure, there are some obstacles that need to be overcome. It is important that the similarity measure of the sub-taxonomies can be used even when the respective attribute values are put back into their original context. Furthermore, because we are splitting up a structure, that could mean that some parts of the structure are reused at more than one single point. This offers the chance of eliminating redundant work on repetitive parts of the taxonomy, but also forces us to consider means to keep the nodes uniquely identifiable.

This paper is structured as follows, in Section 2 we describe some related work and distinguish our approach from previous research. Section 3 describes in more detail the approach of splitting and merging taxonomies and its implementation within the tool myCBR[1]. We then give an argumentative evaluation and conclude with an outlook on future work.

2 Related Work

There is plenty of research on taxonomies and on combining and merging them, that is not directly related to CBR. Several algorithms have been described to merge different taxonomies and to unify them, such as RCC-5[10] or the Extended Integrated Taxonomy Generation algorithm[8]. These algorithms merge taxonomies from different sources into a target taxonomy. During the merge process duplicate values are removed, sub-taxonomies and leaves are combined, and sometimes unused nodes are removed. The intention of these algorithms is to unify the values of different taxonomies to put them into a common context. Our approach tends not to unify the values of smaller taxonomies, but allows the integration of several taxonomies via reference nodes into another taxonomy. This way we also put the taxonomies into a common context, but with the focus on similarity computation during retrieval.

There is also some research in the context of dynamic similarity measures. Approaches range from dynamic similarity on databases[6] via their use in data mining and clustering[9] to adaptive similarity measures in CBR[7]. These approaches use similarity measures to retrieve data from data sources while dynamically adapting these measures depending on the content of the query. Our approach also allows a dynamic similarity measure, using the dynamic composition of taxonomies for retrieval. This can be done by the user before the retrieval or depending on the content of the query during the retrieval. This allows us to compose a taxonomy as similarity measure for attributes depending on the required information from sub-taxonomies.

3 Splitting and Merging of Taxonomies

In this section we describe our approach for splitting and merging taxonomies and the current implementation in the open source tool myCBR.

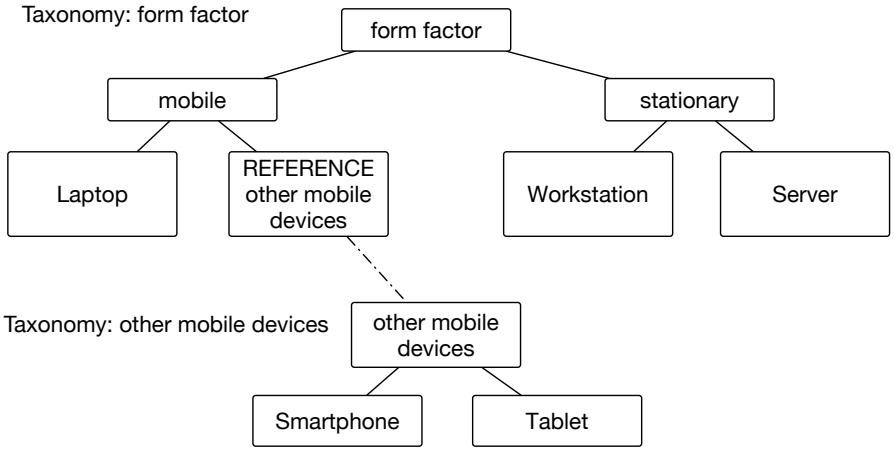
3.1 General introduction to the concept of splitting taxonomies

Before splitting up taxonomies and being able to combine them dynamically, it is necessary to model an interface between taxonomies that defines their relation. By assigning a unique identifier to each sub-taxonomy, it is possible to reference them when needed. A node in one taxonomy could for example reference an entire separate taxonomy by its name. As trees contain one single root element, the referencing node therefore gets replaced with the root-node of the referenced taxonomy. If a referenced sub-taxonomy can not be found, for example because it was deleted, the referencing node will be treated as a leaf node.

Because the important part in the context of CBR is the use of taxonomies as similarity measures, the similarity values of the sub-taxonomies have to remain consistent. Maintaining the original values of the original single taxonomy would be the most obvious solution but does defeat the purpose of splitting the taxonomy in the first place. If all similarity-values of the nodes are simply kept, the sub-taxonomies continue to be dependent on the context from which they are referenced from. It should be possible to model the similarity values of each sub-taxonomy individually without that kind of dependency to use a sub-taxonomy as an individual similarity measure. This also allows the same sub-taxonomy to be reused in multiple contexts by multiple different taxonomies - even across different projects - that reference it.

Considering that a typical taxonomy has the similarity value of 0.0 in its root node and that similarity-values have to increase from the root towards the leaves of the taxonomy, a solution to this problem has to adjust the similarity values of the referenced sub-taxonomy when inserting it back into the project-context. When inserting a sub-taxonomy into another taxonomy, the important parameter to define the context is the similarity-value in the referencing node. Therefore, it is also important that a referencing node keeps the value that was originally in place at its position when a taxonomy is split.

Fig. 1. Taxonomy with Reference



3.2 Merging of taxonomies

Once taxonomies are separated it should be possible to merge them together into a single taxonomy that can directly be used in the context of CBR. For the merging procedure we propose the compilation of a combined taxonomy, based on a formula that adjusts the similarity-values of a sub-taxonomy when inserted into a taxonomy that references it.

In the following formula the similarity value of the referencing node is represented by $sim_{referencing}$ while $sim_{referenced.root}$ represents the value stored for the root node of the referenced taxonomy. Given the old similarity value of any node of the referenced sub-taxonomy sim_{old}, it is possible to calculate a new similarity-value $sim_{calculated}$ for the node in the project-context.

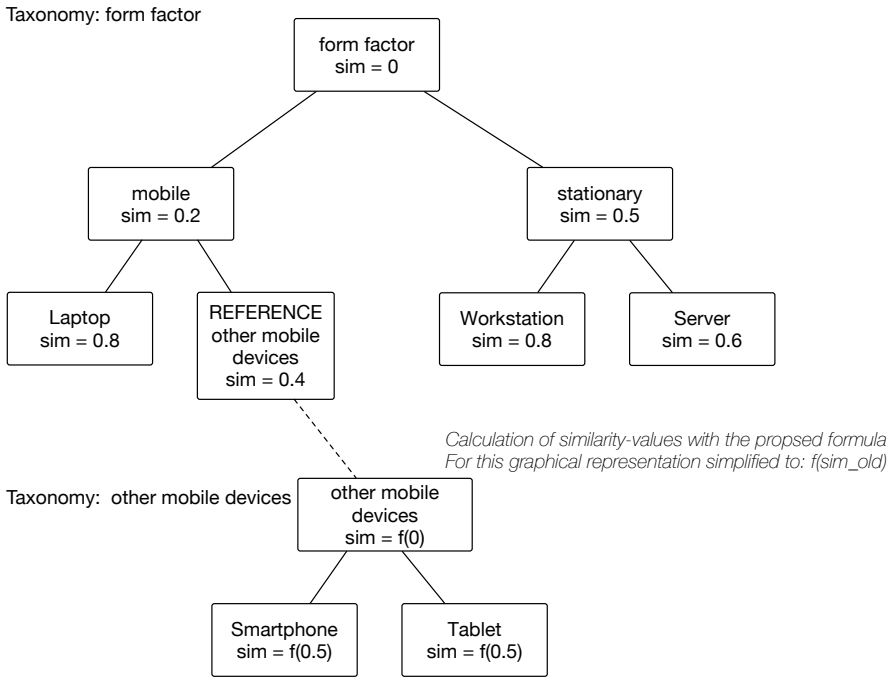
$$sim_{calculated} = sim_{referencing} + \frac{sim_{old} - sim_{referenced.root}}{1 - sim_{referenced.root}} (1 - sim_{referencing})$$

This formula proportionally maps the similarity values of the referenced sub-taxonomy from the interval $[sim_{referenced.root}, 1]$ to the interval $[sim_{referencing}, 1]$. If $sim_{referencing}$ or $sim_{referenced.root}$ is already 1.0, all nodes of the sub-taxonomy are inserted into the project-context with the similarity-value of 1.0. The mapping is achieved following the pattern of the mathematical mapping from a source-interval to a target-interval as shown later in this section with x_{mapped}.

Depending on the conventions of a project, one might assume that $sim_{referenced.root}$ always carries the value 0.0. This does however not have to be the case. Some projects might choose to keep their existing values when they split taxonomies because the people working on the different parts of the taxonomy are already

used to seeing specific values for their part of the taxonomy. Using the proposed version of the formula, that remains possible while assuming $sim_{referenced.root} = 0.0$ would offer the possibility to further simplify it.

Fig. 2. Calculation of Similarity-Values



Given that all nodes below the referencing node have higher or equal calculated similarity-values compared to it, the condition to have increasing similarity-values from top to bottom is fulfilled.

In the following, we describe the merge process of taxonomies. We call the referencing taxonomy $TaxA$ while the referenced taxonomy is called $TaxB$. The similarity-values of the nodes in $TaxA$ remain the same as before. $TaxB$ initially starts with a root-node that carries the same similarity-value as the referencing node. All other nodes are assigned the same similarity-values as before.

Because the goal is to be able to look at the taxonomies as separate and independent constructs, the values of $TaxB$ are now mapped proportionally from the original interval $[sim_{referencing}, 1]$ to $[0, 1]$. $TaxB$ now carries similarity-values that are detached from the original project-context.

Finally and most importantly, the resulting values calculated for the project-context are actually **identical** to the values of the original taxonomy. In fact it does not matter to which interval $[b, 1]$ we mapped the original values to as long as $b \neq 1$.

Proof (Calculated similarity-values are identical to original after merge). Assuming the original values of a sub-taxonomy were mapped from [a,1] to [b,1], $a, b \neq 1$ and a given mapped similarity sim_{mapped} of an arbitrary element from that sub-taxonomy, we have to prove for that the calculated similarity based on sim_{mapped} is the same as the original similarity $sim_{original}$ ($sim_{calc} = sim_{original}$)

Note: b=a is not excluded

Inserting into the formula, we get

$$sim_{calculated} = a + \frac{sim_{mapped} - b}{1 - b}(1 - a)$$

As proportional mapping from an interval [g, h] to another interval [i, j] follows the scheme

$$x_{mapped} = i + \frac{x - g}{h - g}(j - i)$$

and we originally mapped from [a,1] to [b,1], we get

$$sim_{mapped} = b + \frac{sim_{original} - a}{1 - a}(1 - b)$$

$$\Rightarrow sim_{calculated} = a + \frac{b + \frac{sim_{original} - a}{1 - a}(1 - b) - b}{1 - b}(1 - a)$$

$$= a + \frac{\frac{sim_{original} - a}{1 - a}(1 - b)}{1 - b}(1 - a) = sim_{original}$$

□

3.3 Compilation of taxonomies

When implementing the compilation algorithm for sub-taxonomies, the basic structure follows the patterns used for conventional trees. Each node contains a descriptive id which matches an attribute value of the CBR application and a similarity value. A non-leaf node may have children nodes attached directly to it or via a reference to another taxonomy.

In our implementation the decision between children nodes and a reference is exclusive in order to keep the overall structure as simple to understand as possible. This, however, is a pure design decision and does not impose a technical restriction. The decision was made in order to achieve a clear distinction between the two different ways in which children may be included. Nodes that reference another taxonomy are called referencing nodes while others are called non-referencing nodes.

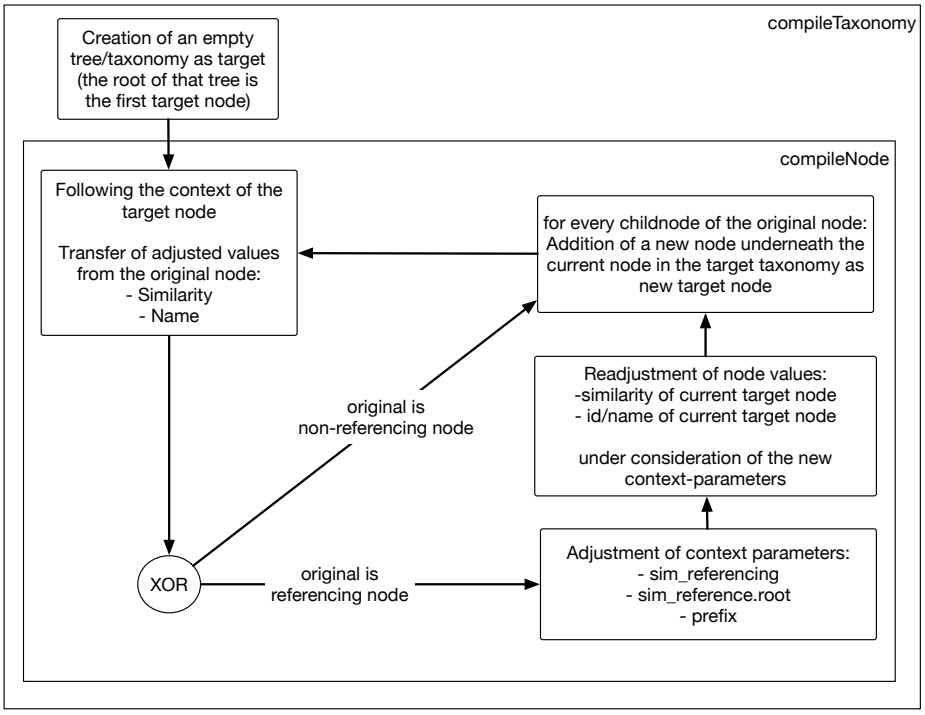
The compiler gets passed a set of sub-taxonomies as resources for the compilation as well as the information on which sub-taxonomy defines the root of

the desired output. Because all of the sub-taxonomies are referenced by their id/name, it makes sense to use a key-value mapping for efficient access to the requested sub-taxonomy.

When inserting the same taxonomy multiple times into one big taxonomy at different referencing nodes, the identification of the different instances of the sub-taxonomy needs to remain unambiguous. For those use-cases, the compiler offers the option to generate unique names for each of the nodes by inserting the name of the referencing nodes parent as a prefix before the names of the nodes in the referenced taxonomy. Using ':' as a separator symbol between the names, the names for the taxonomy-nodes from 'other mobile devices' as shown in Figure 2 would result in 'mobile:other mobile devices', 'mobile:Smartphone' and 'mobile:Tablet'.

Figure 3 shows an overview of the algorithm executed in order to compile the taxonomy. It should be noted that the algorithm treats directly attached children and children that are inserted by reference in exactly the same way. A referencing node gets replaced by the node that it references and the children of that referenced node are considered its children.

Fig. 3. Overview of the Compilation-Algorithm



3.4 Runtime depending on input size

We define the number of nodes of the resulting taxonomy as input length n. This definition of the input length is somewhat unconventional as it assumes information about the resulting taxonomy. It still makes sense though when put into the context of our concept. The alternative to this algorithm and concept is to have large singular taxonomies with all their disadvantages. The concept of subtaxonomies offers the possibility to represent the same taxonomies in a simplified way - therefore taking the resulting singular taxonomy as input size is not as far fetched as it might seem at first. The basic processing of each node happens in $O(1)$ as the only action performed for each node is the adjustment of its values. Each node needs to be processed one time resulting in $O(n)$ for the average, worst and best case. The part where a non-linear increase of the runtime is possible would be the process of finding the according taxonomy for a reference. As mentioned before, an access-optimize storage model for access from a given name/identificator of a taxonomy would make sense here. The expected effort for finding elements by using hashing is within $O(1)$ depending on the amount of elements in the hashtable compared to the size of the hashtable [5, Chapter 6.4]. While the worst case can be $O(n)$, optimized hybrid implementations like Java 8's HashMap offer a runtime of $O(log(n))$ for the worst case by using search trees.

The algorithm is able to efficiently compile a taxonomy within an expected runtime of $O(n)$. In the end even the worst case runtime of the algorithm is in $O(n * log(n))$. It should be noted that this would assume that the largest part of nodes are referencing nodes, which does not make sense in real life scenarios for obvious reasons. Assuming a low (compared to the number of nodes in total) and constant number of referencing nodes we further approach $O(n)$ even when hitting the worst case for the hashtable.

3.5 Implementation within myCBR

myCBR is an open source tool and library for CBR modeling and retrieval tasks that integrates structural CBR into an easy to use interface, the myCBR workbench and can also be accessed through an API. It is a joint effort of the Competence Center CBR at DFKI, Germany, and the School of Computing and Technology at UWL, UK. Various industry partners like Airbus and Lufthansa Industry Solutions work together with those institutes to explore new possibilities and approaches to CBR systems[4].

Before the integration of sub-taxonomies into myCBR, the tool already offered support for traditional taxonomies that was used as the basis for the integration. As a matter of fact, the integration was achieved while keeping the existing calculations for similarities between cases intact. Sub-taxonomies get compiled into a construct that remains largely unchanged from the original structure which was already used before for the calculation of similarities between cases.

While adding the functionality for sub-taxonomies along with cycle-detection[3] during the compilation of sub-taxonomies to the SDK, the user interface also had to be adjusted.

We created a separate perspective for sub-taxonomies in the workbench, where all sub-taxonomies are collected. The tool myCBR separates attributes and similarity functions in a way that different types of similarity functions can be assigned to an attribute. The addition of the function-type 'Sub-taxonomy' allows to assign a single sub-taxonomy to an attribute that is then compiled to a complete taxonomy using all other sub-taxonomies that exist in the project.

Traditional taxonomies that existed in older myCBR projects can be imported as sub-taxonomies. The export of single sub-taxonomies is also possible to transfer them to other myCBR projects. Those taxonomies can be split according to their logical parts (e.g. every system inside of a plane gets its own sub-taxonomy for the plane-taxonomy). However, it serves the purpose of lowering the effort for maintenance to pay special attention to duplicate (sub)parts inside of existing taxonomies.

Processing a traditional taxonomy to gain a set of sub-taxonomies that eliminates duplicate information is a workflow which consists of three steps. First we check if at least one of the imported taxonomies can be simplified. A taxonomy can be simplified if there is repetitive information across several taxonomies or within the same taxonomy. If there are several taxonomies that can be simplified, we use the new myCBR functions to *split* the repetitive part from the original taxonomy. Then we use *set reference* to replace the according part in all other taxonomies. After this process we check again if the taxonomies can be simplified even further. If no further simplification is possible, the workflow is finished. Figure 4 shows an event-driven process chain for this workflow.

Fig. 4. Workflow to simplify taxonomies with myCBR Workbench

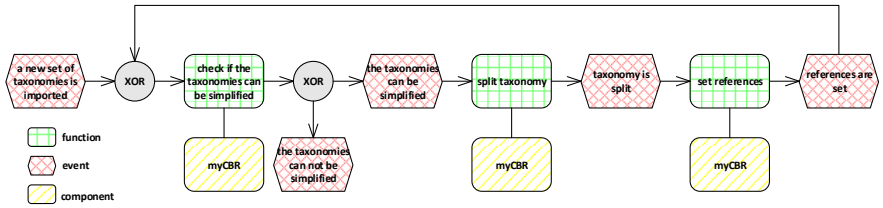
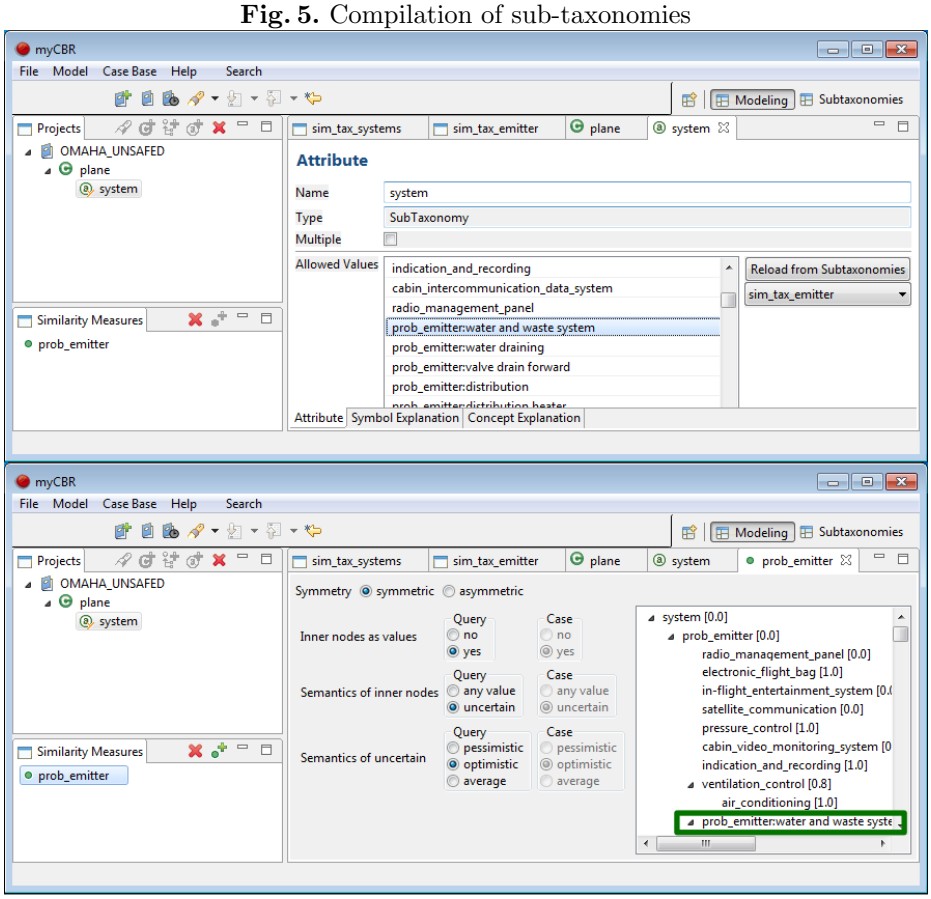


Figure 5 shows the interface of the myCBR-Workbench in the 'Sub-taxonomies'-perspective.In order to use a sub-taxonomy for the calculation of similarities for an attribute, the user has to switch to the 'Modeling'-perspective and add a Sub-taxonomy-Function as a similarity function to an attribute inside of the

[3] For example: TaxA references TaxB and TaxB references TaxA resulting in a circle

Fig. 5. Compilation of sub-taxonomies



current project. By assigning a Sub-taxonomy-Function as similarity function, the user can now choose any of the taxonomies contained in the collection of sub-taxonomies inside of the project. That taxonomy will be used as the root taxonomy for the compilation of a single taxonomy out of all referenced sub-taxonomies.

As a result of the selection, the taxonomy gets compiled and myCBR can now perform the same similarity-calculations on that taxonomy that were possible before, using traditional taxonomies. In contrast to traditional taxonomies, this compiled version of sub-taxonomies can not be modified directly. All modifications have to be performed directly on the sub-taxonomies which it consists of. After modification of one or multiple of those sub-taxonomies, the user can choose to recompile this taxonomy at any time by clicking "Reload from Sub-taxonomies".

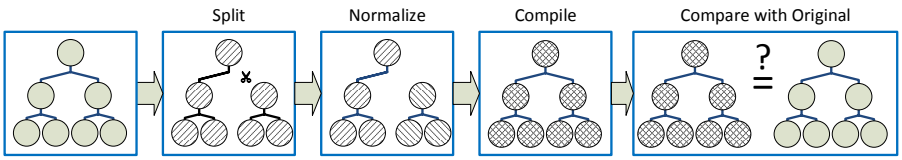
4 Evaluation

Following the principle of separation of concerns it makes sense to split up a taxonomy into multiple meaningful parts. Being able to regard each resulting part separately allows for individual modeling thereby making work easier for the ones working on the different parts. Without using the concept of sub-taxonomies

proposed in this paper, a change in a parent node needs to be reflected in all the nodes that succeed it. In our concept, each sub-taxonomy that is integrated under the according node through compilation does not need to be touched. In instances where parts of a taxonomy are reused across multiple projects or multiple taxonomies in the same project, there is also less modeling effort as the changes do not need to be copied over through multiple taxonomies anymore and can instead be implemented by one single sub-taxonomy.

While we already demonstrated the feasibility of using sub-taxonomies through mathematical means earlier in this paper, we also computed multiple random splits of taxonomies provided by airbus at 20 % of each taxonomies nodes, mapped each resulting sub-taxonomy from $[sim_{root}, 1]$ to $[0, 1]$ [4] and finally compared the resulting compiled taxonomies to the original taxonomy checking for deviations.

Fig. 6. Evaluation Method



This process was repeated 10 000 times for the following taxonomies of the OMAHA research project resulting in the processing of 1 020 000 nodes in all iterations combined:
tax_engine_type was split 6 times (Taxonomy size:30 nodes)
sim_tax_ata was split 10 times (Taxonomy size:52 nodes)
sim_tax_emitter was split 3 times (Taxonomy size:18 nodes)
default function was split 0 times (Taxonomy size:2 nodes)

Alongside consistency checks on the general structure of the taxonomy (naming, tree structure), we also measured the average and maximum error between the nodes of the compiled taxonomy compared to its original counterpart. As expected, the results showed no measurable deviation from the original taxonomy.

5 Summary and outlook

In this paper, we presented a concept that offers the functionality to combine separately created and maintained sub-taxonomies into a single taxonomy. The feasibility of this concept has been mathematically proven and demonstrated

[4] Mapping → Normalization of the sub-taxonomy

with the integration into the myCBR-project. It was further shown that the concept does not impose any noteworthy computational overhead.

During the design we aimed for a concept of taxonomies that allows for efficient maintenance and we assume that the adaption of this concept would bring benefits to any area where complex and hard to maintain taxonomies are prevalent. It is possible to separate complex taxonomies with relatively low effort and risk as similarity values can be kept identical before and after this process. On top of that our concept makes sharing parts of a taxonomy across one or multiple projects possible and comfortable.

For the future an evaluation of this concept with respect to the maintenance effort after and before introduction together with industry partners would be desirable.

References

1. Bach, K., Sauer, C.S., Althoff, K.D., Roth-Berghofer, T.: Knowledge modeling with the open source tool mycbr. In: Proceedings of the 10th Workshop on Knowledge Engineering and Software Engineering (2014)
2. Bergmann, R.: On the use of taxonomies for representing case features and local similarity measures. In: Proceedings of the Sixth German Workshop on CBR. pp. 23–32 (1998)
3. Bergmann, R.: Experience Management, Foundations, Development, Methodology and Internet-Based-Applications. Springer Verlag Berlin Heidelberg New York (2002)
4. BMWI: Luftfahrtforschungsprogramms v (2013), http://www.bmwi.de/BMWi/Redaktion/PDF/B/bekanntmachung-luftfahrtforschungsprogramm-5,property=pdf,bereich=bmwi2012,sprache=de,rwb=true.pdf
5. Knuth, D.: The art of computer programming. Addison-Wesley Pub. Co, Reading, Mass (1973)
6. Köppelmann, M., Lange, D., Lehmann, C., Marszalkowski, M., Naumann, F., Retzlaff, P., Stange, S., Voget, L.: Scalable similarity search with dynamic similarity measures (2012)
7. Long, J., Stoecklin, S., Schwartz, D.G., Patel, M.K.: Adaptive similarity metrics in case-based reasoning
8. Raunich, S., Rahm, E.: Target-driven merging of taxonomies. CoRR abs/1012.4855 (2010), http://arxiv.org/abs/1012.4855
9. Ruiz-Miró, M.J., Miró-Julià, M.: Dynamic similarity and distance measures based on quantiles. In: Computer Aided Systems Theory – EUROCAST 2015 (2015)
10. Thau, D., Bowers, S., Ludäscher, B.: Merging taxonomies under rcc-5 algebraic articulations. In: Proceedings of the 2Nd International Workshop on Ontologies and Information Systems for the Semantic Web. pp. 47–54. ONISW '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1458484.1458492