How Trees and Traps Become Part of the Language Vocabulary – Postulating the Need for a New Meta-Concept for Composite Elements

Victoria Döller[1,2]

[1]Research Group Knowledge Engineering, Faculty of Computer Science, University of Vienna, Vienna, Austria
[2]UniVie Doctoral School Computer Science DoCS, University of Vienna, Vienna, Austria

Abstract

This paper discusses the need for a new meta-concept termed *compound type*, which represents composite structures in models that cannot be captured by traditional object types. Compound types allow for the expression of overt concepts that emerge from the mere coexistence of other modeling elements. This can be, in particular, seen in the example of trees that are instantly perceived by humans as a unit. The constituting nodes and edges are subordinate to this structure. The pervasiveness and relevance of compound types are illustrated in several more examples from various modeling languages and domains, including process modeling, Petri Nets, and ER diagrams, where concepts like processes, traps or ERD constructs are essential but not directly represented in the language's metamodel. By introducing the meta-concept of compound types on the meta^{2}level, the paper aims to extend the expressiveness and functionality of modeling languages, ensuring they can handle these complex structures. This allows for "speaking" about compound instances within the vocabulary of the language, making them available to actors not exhibiting human cognition, i.e. machines. An attempt is made to neatly integrate the new meta-concept in the meta^{2}model and put it in context to the established meta-concepts. In the end, composite types are determined as a means of operationalizing the model.

Keywords

Metamodeling, Meta^{2}model, Compound Type, Model Operationalization

1. Introduction

When creating models, we intend to capture information by using a modeling language, instantiating concepts provided by the language, and composing instances according to the information in our mind. When talking about the content of a model, there can be some curiosities that most experienced modelers are not even aware of. We speak about elements that are not explicitly mentioned in the model. This experience was made by the author when designing the metamodel for a modeling method in math education. The central concept, a frequency tree, was not eligible to become an element in the metamodel because it is not an element instantiated by the modeler. For the implementation this enforced some workarounds to be able to capture all the necessary information because this information is related to the unit of the tree and cannot be derived from the single constituting elements.

This observation is strongly reminiscent of the principle that "the whole is greater than the sum of its parts" as used in Gestalt psychology, compare to the triangle in Fig. 1. In this figure, the prominent element perceived first of all is a white triangle pointing downwards, although there is not a single line of its contour drawn explicitly. It simply exists because of the coexistence of the other elements. There is another analogy to metamodeling: By the creation and concrete arrangement of three black notched circles (instantiation and positioning) we can model triangles without having a concrete triangle shape at hand (no "triangle" concept in the language).

This lack of a corresponding concept is not a big problem when using models in knowledge exchange between humans, and models have proven highly valuable in this case because human cognition

Companion Proceedings of the 17th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling Forum, M4S, FACETE, AEM, Tools and Demos co-located with PoEM 2024, Stockholm, Sweden, December 3-5, 2024
victoria.doeller@univie.ac.at (V. Döller)
0000-0002-5770-635X (V. Döller)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-3855/forum4.pdf

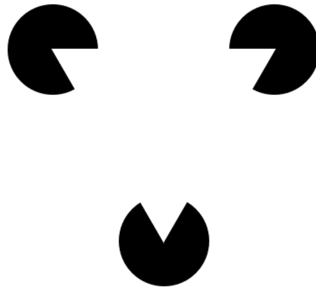

Figure 1: In this picture, we primarily perceive a triangle. The notched circles are subordinated to that.

preferentially perceives the sum of elements as unity. Nevertheless, when the models shall be understood by an entity (probably a machine) not capable of human cognition, only capable of "speaking" the language defined by the metamodel, this entity will fail in retrieving any information that is so obvious for the human model consumer.

The motivation of this work is to discuss the need and realization of a means to represent composite elements as described above. This must be a meta-concept, i.e., a concept of the meta^2-model, we will call it *compound type*. Compound types shall capture the unique nature of these compositions, allowing for suitable and more meaningful representations of their instances within models. By incorporating compound types, we can extend the expressive power of modeling languages, enabling them to describe and reason about these complex structures.

Although a precision of the meta^{2}level where the building blocks of languages are defined is a prerequisite for understanding and developing conceptual modeling languages, investigations on this level are rare, e.g., [1, 2, 3]. This work is part of an endeavor to reconsider meta-concepts and approach a state-of-the-art meta^{2}model for conceptual modeling research [4].

In this paper, we will demonstrate the need for a new meta-concept on a row of examples in Section 2, we investigate existing related concepts in Information Systems Modeling in Section 3, search for compound types in the language design process in Section 4, analyze rules for the derivation of such composite elements in Section 5, spend some thoughts on the necessary adaption of the meta^{2}model in Section 6, summarize the potential of compound types and their role for model operationalization in Section 7, and summarize our thoughts in a conclusion in Section 8.

2. From Trees to Traps: Exploring the Occurrence of Compound Types in Diverse Modeling Languages

In this section, we will examine some well-known modeling languages for the occurrence of composite concepts and begin with trees as mentioned above:

A rooted tree diagram is a composition of several nodes (representing events, sets, types, ...) given that no loops in the event sequence occur. Additionally, the tree diagram holds information about the depth of the tree and about correctness of attribute values taking into account neighbours, predecessors, etc.. Consider Fig. 2. What we perceive is a frequency tree, not a set of nodes that are related. Even the fact that the set of nodes and edges is a tree is only verifiable by considering all nodes and connections at once. We observe that the tree is binary. It has depth two and obviously an error in the represented data, as the sum of the two subgroups in the left branch of male and female professors exceeds the number of all professors. Similar to the triangle in Fig. 1, we see that the actual cognitive artifact that we consider in our mind when looking at this model is not a set of connected nodes but a solitary object of a tree with several attributes deduced from the mere cooccurrence of other elements. In this case, we don't even need a metamodel to know the attributes, as tree diagrams are commonly well-known and easy to comprehend. What we also see is that all this information (or technically speaking attributes) cannot be captured as information of the single nodes as none of them is aware of, e.g., the conformance

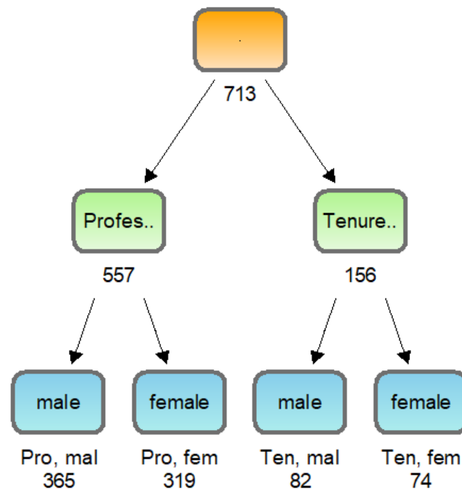

Figure 2: A tree depicting the distribution of professorships and tenure track positions on male and female academics

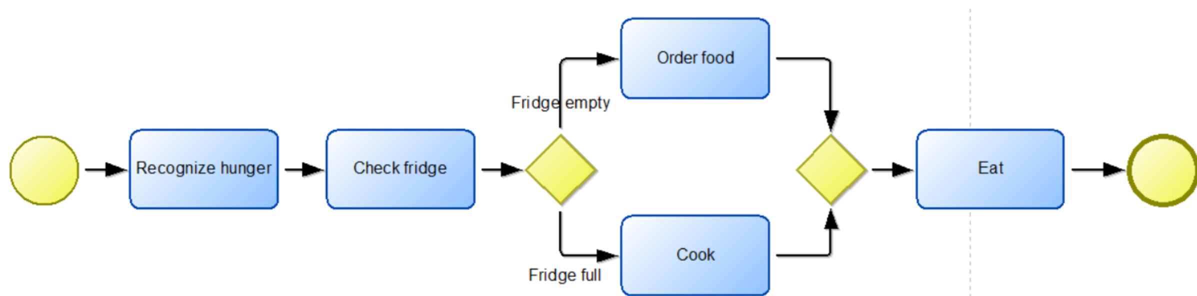

Figure 3: A process model about someone's eating habits

to the tree-structure, the depth of the tree, or the binarity. Therefore, a metamodel lacking a concept for the tree cannot capture this information (apart from inelegant, technical workarounds). With the meta-concept of a *compound type* in the meta^{2}model and the compound type *tree* in the metamodel, we could solve this issue.

A prominent example in enterprise modeling is the investigation of processes. In projects on process optimization, participants might drop statements like "The process for product delivery is so inefficient. The average duration is three weeks. This branch of the process has too much delay." Surprisingly, the central term in process modeling – the process – is not part of the vocabulary of most common process modeling languages, e.g. BPMN. A process is a composition of a start event, several subsequent tasks, and one or more end events. At the same time, the process carries much more information than every single element constituting it, e.g., knowledge about the duration of the whole process, costs, bottlenecks, etc. But also subordinate concepts are composited, like a branch of the process. Looking at Fig. 3, we see a short process about someone's eating habits. We see that there are no parallelities or loops so we do not need to fear any deadlocks or endless loops. If we had information about the average duration of the single tasks (usually captured in attributes not shown directly in the model) we could furthermore calculate the expected faster process instance, *cook* or *order food*, an information not captured in any of the single tasks. With the meta-concept of a *compound type* in the meta^{2}model and the compound types *process, process branch,* etc. in the metamodel we could solve this issue.

An example for the actual realized support of tasks belonging together, or in other words a composite element, is the visual clamping of tasks by a swimlane. For process modeling, it is also noteworthy that as soon as some sophisticated functionality like simulation is executed on the model indeed, compound concepts are introduced in the language, e.g., BPEL does have a concept *process* in its metamodel.

We proceed with the next example out of the row of established modeling languages and standards –

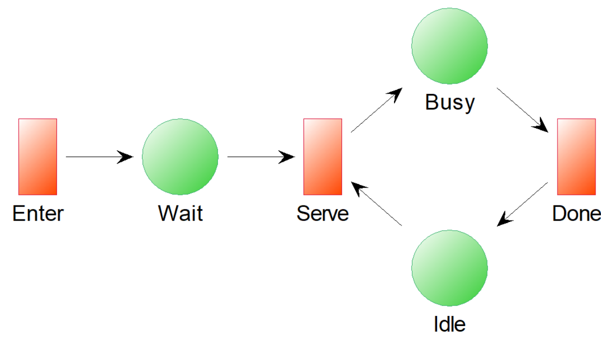

Figure 4: A Petri net with traps depicting a barber shop

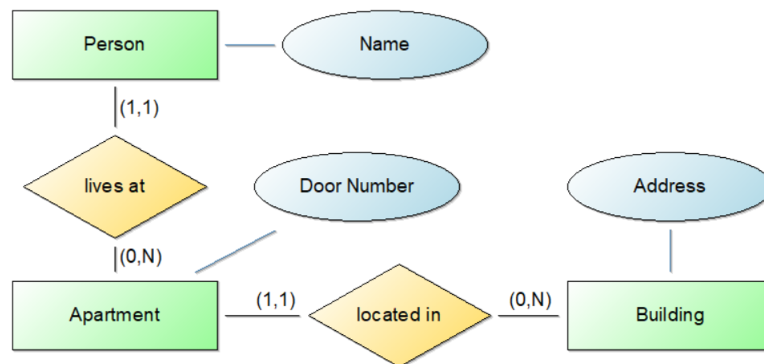

Figure 5: An ER model about the housing situation

Petri Nets: Traps and cotraps in Petri Nets are substructures that consume or emit tokens and allow for conclusions on, e.g., deadlocks. Traps are subsets of places with concrete behavior that can be roughly described as follows: If a transition takes something out of the trap, it also puts something back. Such traps can only be spotted by analyzing the net as a whole, considering the interdependence of places. When we look at the example of a barber shop in Fig. 4, we can observe that (for instance) the places *Idle* and *Busy* form a trap, and the places *Wait* and *Idle* do not. A trap is a full-value concept in a net and its information and value exceed the information captured in a mere set of places. This compound type is different in nature from the last two examples of trees and processes, as traps are not obvious for us to see in a model. So, in this case, the mere existence and identification of traps is of interest rather than the calculation of additional attributes. Furthermore, as traps are hard to detect, automation is desirable, but to integrate this functionality directly in the modeling language, the metamodel must be extended with a compound type *trap* so that the machine is even capable of making statements about traps.

As last example of the established modeling languages in computer science will serve the Entity-Relationship modeling language: The triple of two entities in ER connected to the same relation element form a unit that is the minimal structure of an utterance about the entities' relation and is a prerequisite for cardinality constraints to make sense. This unit was coined by Chen *ERD constructs* in analogy to a natural language sentence [5]. This is further underpinned by the fact that only such triples can be transformed into SQL statements. In Fig. 5, we find the ERD constructs *Person lives at Apartment* and *Apartment located in Building*. We can explicate some cardinality constraints: A person can live at exactly one apartment and an apartment can be located in exactly one building. All this information can only be deduced by a triple of connected elements, two entities and a relation in between.

Also, in models beyond the boundaries of computer science, we can find the concept of compounds (due to the lack of the author's deep knowledge in these areas, we will omit a discussion of a concrete example):

The structural formula of a chemical compound represents its molecular structure. Each detail,

starting from the number of atoms to the exact chemical bonding, is determinative for the identity of the concrete chemical compound, whether it is water or caffeine.

In circuit diagrams, the interrelationship of power sources, consumers, resistors, wires, etc., determines if the electric circuit, a composition of elements, works properly.

We see that the principle of conceiving a composition of elements that expose a concrete structure or behavior as a solitary item is ubiquitous. It is not restricted to a concrete domain or use case. To face this requirement within a modeling method, we have to extend the language vocabulary to even be able to "speak" about this solitary item, may it be a process, a frequency tree, a trap, or any other concept mentioned above. Otherwise, the language syntax is not powerful enough to allow for the postulation of statements concerning compositions of elements. These composite concepts must become part of the metamodel, nevertheless, in another way as classical instantiable object types. We need a compound type that matches the specifics of the types described above.

2.1. Current Practices to Work Around the Problem

For the examples of *processes, traps*, and *ERD construct*, one might argue that these modeling languages are very successful without having the mentioned compound concepts in their metamodel. This is valid but simply reflects the fact that the concepts were not relevant to the primary purpose of the languages. As long as the machine does not need to simulate a process it has no need for the concept of a process. As long as no one searches for traps in nets, there is no need for a trap concept. As long as an ER diagram is only used for communication, not for SQL generation, there is no need to consider ERD constructs in the language. But as soon as we add the mentioned functionality, the machine needs a concept of a process, a trap, and an ERD construct.

Indeed, they do but this knowledge is often outsourced to software solutions detached from a modeling tool or language. So this is also a kind of a workaround: As the modeling language does not provide the needed concepts they are detached from the modeling language and realized beyond the metamodeling paradigm. However, this practice weakens the actual potential of modeling languages and limits the model value because the determination of such implicit information constitutes a central goal of modeling. It belongs to the major functionality of many languages and model functionality should be an integral part of a modeling method [6]. Indeed, compound types are a way of explicating this intrinsic information of models and are, therefore, means to incorporate information-processing functionality into the language specification. By not exercising the authority to define the functionality in the language specification, it becomes arbitrary, tool-specific, and indefinite. Furthermore, by capturing this knowledge in external software, we miss the possibility of raising the level of abstraction for the system under study by working with a conceptualization of the domain instead of solving the problem on code level.

3. Similar Concepts in Information Systems Modeling

An akin idea to a compound type, the so-called *derived type*, was introduced by Olivé in the book *Conceptual Modeling of Information Systems* [7, Chap. 8]. A derived type is a type whose instances are computed or derived from other data or elements in the information system, database or others, rather than being directly instantiated by users. The derived data is a result of applying logical rules or functions to the base data within the model. This already mentions a further relevant conception for our considerations, the derivation rule: Each derived type is defined by a derivation rule, which specifies how the instances are computed based on other elements. These rules can involve mathematical functions, logical operations, or even more complex algorithms that compute derived data from base data stored in the system. Examples of derived types are the *square* type in a population of *rectangles* given the side lengths or the *grandparent* relation type deduced from the relation type *parent*. Derived attribute values are, for example, an employee's total annual salary calculated from their base salary and bonuses. Indeed, derivability is not an intrinsic characteristic of an entity or relationship type and depends on the available vocabulary of the modeling method.

The fundamental difference to the considerations done above is that the conceptual models considered here are schemata of information (and the book prototypically uses UML as the language for these models). The derivation does not affect this modeling level but the instance level of concrete information or data. However, this information and data is not the content of the model. For this reason notions like tree or process are not of relevance in this approach.

4. Compound Types and the Design of the Metamodel

The necessity for including compound types in the vocabulary, i.e., the metamodel of a language, is backed by the guidelines and macro processes for domain-specific-modeling language (DSML) development as proposed in [8]. In this work a six step (circular) process of language specification comprising the steps *create basic/extended glossary*, *develop concept dictionary*, and *design draft metamodel* is postulated. Thereby, the glossary should be a collection of all key terms that are frequently used in the domain of discourse. Out of these terms, the concepts for the metamodel have to be chosen. For this election, there are several decision criteria proposed: *invariant semantics* requesting a stable meaning of the term/concept throughout the domain and in time, *relevance* distilling the necessary concepts from the optional or rarely used ones, *variance of type semantics* requesting instances of the concept not to be semantically identical or overly similar.

Checking the concept of a *frequency tree* in the domain of math education against these criteria shows: the concept of a *tree* does have invariant semantics – trees are an established concept in mathematics – they are supremely relevant for the domain – and the tree instances have varying semantics, e.g., a tree depicting the partition of academic employees and another one showing the distribution of blood groups. The same is valid for the concepts *process* in process models or *trap* in Petri Nets. This affirms the actual relevance of the concepts for the corresponding languages, and without them, the languages are not capable of making statements about trees, processes, or traps. Still, the concepts *process* or *trap* are not part of the metamodels of (e.g.) BPMN and Petri Nets, and this is for a plausible reason: A concrete process and trap are not instantiated as single, stand-alone elements by the modeler, and they are technically not realized in a modeling tool. However, this is not a criterion on the checklist mentioned above. They are composite elements emerging through the combination of other subsidiary elements in the model. So we need the concepts *process* and *trap* as part of our metamodel but not as object types, the most familiar meta-concept. We need another more sophisticated meta-concept for this case, the bespoken *compound type*.

Returning to the design guidelines above, we could extend the list of decision criteria to determine the concrete character of a term – regular or compound – in our glossary:

- Do the instances of the concept dependent on the existence of other elements, on their togetherness? Are these elements in any form part of or participating in the concept instance?
- Does it seem deviant to show instances of the concept as a stand-alone object? Is it the structure, the connectedness, or the proximity of other objects that marks the essence of the concept?
- Does an instance of the concept automatically come into being when a couple of other elements fulfill some preconditions?

If the first two points can be answered with *Yes*, the concept most probably has the nature of a compound type. If also the third point is answered with *Yes*, these compound elements (instances of the compound type) might even be automatically deduced from the model, compare to the concept of *derived type* as introduced by Olivé [7] (see Section 3 for a description). In this case, we can postulate a derivation rule enabling the automatic determination of all instances of the type. Examples of such derivable types are frequency trees, processes, or traps. In the latter case (and other cases), the derivation of the compound element is even the target of language functionality. Surely, this is not always the case, and compound elements might only be deduced by interpreting their semantics or names. On occasion, the modeler has to express the compoundness of elements by hand. A common practice is to

use visual markers. This can be realized by containing boxes like swimlanes in BPMN or packages in UML, or colours, e.g. the identification of teams or positions in soccer strategy modeling.

Furthermore, it makes sense to give thought about terms relevant to functionality:

- Are there any additional terms appearing when operationalizing the model, terms that were not relevant when considering the model only as representation of a system? Does the functionality bring into use some extra attributes of new or existing types?

This might result in the extension of the vocabulary with concepts carrying information needed in intermediate processing steps, e.g., parallel subpaths in process models, or attributes holding the results of the functionality, e.g., information about the average duration of a process.

We have to mention that these questions might be answered differently in different domains or considering different purposes. A process might indeed be a plain object type if we simply want to mention it in a model but are not interested in the details of concrete activities, e.g., for an overview of the processes of an enterprise.

While the catalog of questions above guides us in the decision if a type is a compound type, the concrete disposition of this type is not definite. It is a matter of design and depends on the purpose. Are we interested in the structure of the elements participating in the compound, or is it just their mutual existence? Based on this we might perceive the compound type as a structured tuple like it is the case for process instances, or a set like it is the case for traps, including or excluding also relation elements.

5. Constructive Constraints Revisited

Besides the use of stand-alone compound elements, derived knowledge can also appear in small pieces of information about modeling elements, i.e., attributes of compound or standard types. For this, we recall the notion of constructive constraints as introduced in [9]. *Constructive constraints* impose some dependencies between elements and values and can be delimited to *restrictive constraints*, confining the creation, connection, and attribution of models (this delimitation is inconclusive).

They play an important role in the context of composite elements and deduced information because they are a tool for prescribing derivable knowledge, for defining the *derivation rule* of a compound type, and for ensuring the correct behavior of its instances. Thereby a derivation rule is different to a usual constraint as it induces the existence of elements (causes the "creation" of all elements that fulfill the derivation rule to be precise), while traditional rules only apply to elements created by the modeler. Furthermore, constructive constraints are the dedicated tool to specify the correct behavior of compound types as these types can become quite complex beyond their basic structure, i.e., a tree or tuple consisting of elements that are neatly bound to another type.

Another use case is the provision of answers to queries or at least the basis for query execution, as this can be captured in dedicated attributes. The answer to the query for the root of a tree diagram might be encoded in an attribute. An attribute holding the costs of a whole process can be used for determining the most expensive process.

6. Implications for the Meta²model and Tooling

An audacious first try to integrate the meta-concept *compound type* into the meta^2-level is done in Fig. 6 (legend on the right). The depicted meta^{2}model comprises, besides the compound type, those meta-concepts that can be seen as the commonly used core according to a survey of established metamodeling platforms in [2]. Commonly used in this case means that the mentioned concepts appeared in the survey in at least half of the investigated platforms.

The basis meta^{2}model of Fig. 6 was developed for a metamodeling language called `M2FOL` presented in [10]. We will shortly summarize the concepts in the following: *Object types* are abstractions of sets of elements in a domain with equal features, constraints, and relationships. *Specialization* is a relation between object types where the subtype inherits properties of the supertype and owns some additional

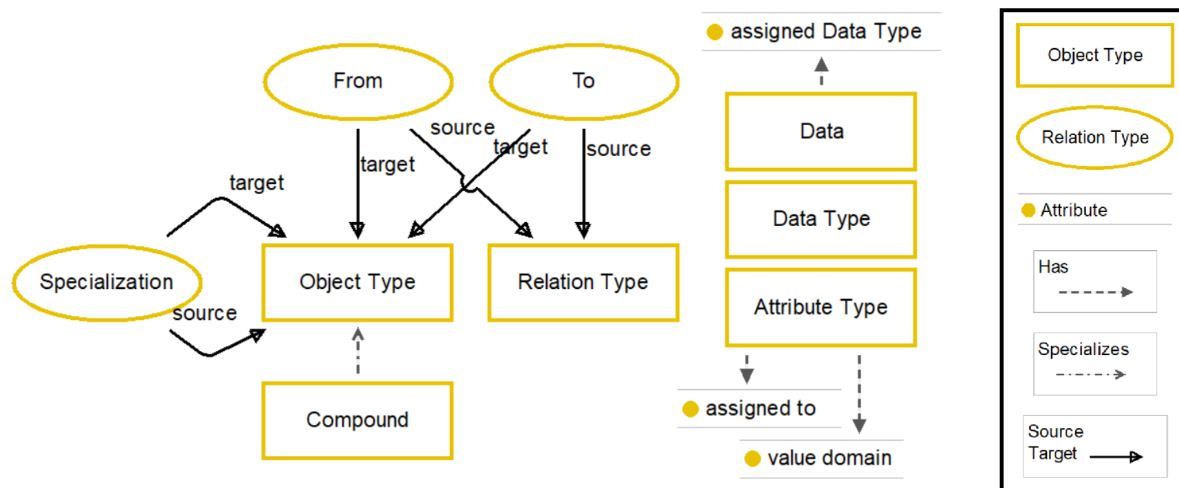

Figure 6: The meta²model including the proposed meta-concept of a *compound type* adapted from [10] (legend on the right)

ones. *Relation types* are abstractions of sets of connections between elements in a model carrying the same features and constraints. They are existentially dependent on the linked objects. In the majority of approaches, relation types appear only in a directed binary form. In the meta²model, relation types are therefore designed with two attributes pointing to the source and target object type. *Attribute Types* capture single features of object- or relation types and hold a concrete value on the model level. The value belongs to the value domain of the attribute, i.e., the admissible set of possible manifestations an attribute might have. To define the value domains in a metamodel, the depicted meta²model contains the meta-concepts *data type* and *data* to name the data type and fill it with possible data values.

Starting from the proposed extended meta² model, we have to raise some questions, some of which can be answered immediately, some of which we mention first considerations on, and some of which have to be considered diligently when further profoundly elaborating the concept of compound types:

- Do compound types behave like object types?

  In a way, they do. They are salient elements in models. They are sets of elements (objects and relations) and are perceived as standalone elements in contrast to relations that are primarily characterized by their connecting role and do not automatically vanish if one element (with exceptions) vanishes. The difference to classical object types is that they do have instances but are not directly instantiable by the modeler.

  At the same time, one could argue that compounds behave like relations because they depend on other elements and describe their relationship, e.g., their connectedness in a tree. They disappear if (all or most of) the constituting elements disappear. Nevertheless, their superordinate role to the constituent elements disputes considering compound types as relation types.

  The third option of compound types as a meta-concept unrelated to object- or relation types seems implausible as they exhibit mostly the same characteristics besides their originator. This is underpinned by the fact that they appear side-by-side in the glossary of a language as mentioned in Sec. 4.
- Do compounds have attributes?

  Yes, they do as outlined on numerous examples above.
- Can compound elements be connected via relations?

  In principle, there are scenarios and purposes where it makes sense to connect such composite elements to other elements, e.g., relate trees to other modeling elements representing the same information. The side effects must be investigated closely, just like the visual realization in a graphic model.
- Can compound types inherit from each other?

This requires further investigation. In principle, for some cases, it makes sense to have binary trees as a subclass of all trees or processes with a reduced set of possible events as a subtype of all processes. This includes a more restricting derivation rule. Also, this point requires a thorough check for side effects.

- Can compound elements be attribute value of another object?

  This needs further investigation. At first sight, there is nothing objecting to this.

Integrating such concepts in the meta^{2}model and the metamodel is a theoretical task. The argumentation for the need for such a meta-concept is based more on a practical experience; therefore, we also have to discuss the modeler's handling of such concepts. For analog models drawn on paper, this does not make a big difference because they are restrained to a pure visualization without the potential of models beyond that, first and foremost, operationalization. So the question has to be raised how the modeler can make use of compound elements in a modeling environment.

The first step is, of course, integration of the meta-concept *compound type* in the meta^{2}model of the platform, which has to be based on the considerations above. A second step must then address the user interaction. How to interact with a compound element? How to simply select such an element? This is necessary to retrieve the information of its attributes, e.g., the average duration of a process or the depth of a tree. Whenever the user clicks on a tree, actually a node or relation is clicked so one of the classical objects. How to "create" compounds in case they are not fully automatically derivable, e.g., mark some elements as belonging together? How to take notice of the automatically generated compounds? This is obvious for trees but nor for traps.

These are relevant questions that might be easily answered with methods from user interaction design. As this is neither the scope of this paper nor the expertise of the author, we acknowledge these questions but leave them unanswered.

7. Résumé and Potential

Compound types are means to extend the expressiveness of conceptual modeling languages. Language specifications become more complete when also incorporating composite notions that are highly relevant for the domain and purpose but not meant to be instantiated by the modeler. This emphasizes their potential to raise the level of abstraction, provide a comprehensive conceptualization of the domain and fosters modeling languages as full-valued scientific artifacts beyond secondary helping tools. If not capable of expressing composite structures like the mentioned examples, we have to outsource this information and its further processing to other methods of processing. This strongly weakens the value of models and modeling languages, hinders them from fulfilling their purpose of providing a comprehensive conceptualization and vocabulary of the domain, and leverages the image of models as tidy but cumbersome, limited representations.

Besides supporting the completeness of the language alphabet, compound types are means for operationalizing models. This operationalization is actually one of the most valuable benefits of state-of-the-art conceptual modeling and, indeed, shall be done using the same principles as the construction of models – the use of abstraction and the increase of efficiency by employing domain concepts for the specification of functionality rather than code.

Compound types are primarily a realization of information-processing functionality on models by explicating implicitly given knowledge. An example of such an information-processing functionality mentioned above is the determination of traps and, therefore, deadlocks. Another prominent example is process simulation of time and costs. Thereby, all the information needed, like duration, likelihood of decision, etc., is already present in the model. With a compound type *process* and some constructive constraints on its attributes, all the information about average duration, costs, etc., can be automatically derived.

A concrete example of the use of compound types in another aspect of model operationalization is model transformation. Usually concepts in different languages differ in the level of granularity and semantics and do not match precisely. In some cases this match can be established by employing

compound types to build composite constructs with sufficiently matching semantics in both languages. An example of model-to-model transformation might be the matching of slightly incongruent concepts of process modeling languages. In model-to-code transformation, the compound concept of ERD constructs can be mapped to SQL statements.

8. Conclusion

In this paper, we have argued for the necessity of introducing a new meta-concept, we call it *compound type*, as a means to capture composite elements in models, i.e., structures of modeling elements exhibiting their own unity. This is strongly reminiscent of the principle that "the whole is greater than the sum of its parts". With this meta-concept, we want to fill a critical gap in the expressiveness and functionality of modeling methods in their role as a language understood and spoken by a machine.

By examining a range of examples across different domains and concepts, such as frequency trees, processes, traps, and ER constructs, we made evident that these concepts are ubiquitous but do not fit the characteristics of a traditional object type. They are not instantiated by the modeler but emerge automatically out of the coexistence of other modeling elements and, in addition, carry their own attributes, such as tree depth, process efficiency, or deadlock conditions in traps. Without compound types, essential concepts are either entirely missing from the model or only captured through inelegant workarounds, but they are needed in the language vocabulary to be able to even "speak" about them. We have shown that compound types allow for the representation of such compositions and extend the expressive capabilities of modeling languages, enabling modeling languages to articulate the relationships and emergent properties that arise from the meaningful coexistence of elements.

We underpinned the right to exist of this meta-concept with a criterion catalog for metamodel design and also discussed constructive constraints as a means to deduce such compound elements. Furthermore, we attempted a redesign of the meta^{2}model, including the newly introduced concept of compound types, and started to outline the implications of this adaption. In the last step, we determined compound types as a means of operationalizing the model, applying the principle of abstraction fundamental for conceptual modeling also on model functionality.

Summarizing, we argued for an imminent need for a new meta-concept, as introduced in this paper, and tried to underpin this need with conclusive examples. The concrete characteristics and behavior of this meta-concept, on the other hand, need further investigation and a collection and contemplation of additional examples and occurrences. Furthermore, the concrete characteristics and implications for the extended meta^{2}model must be examined, as well as their potential in specifying model functionality.

References

[1] B. Henderson-Sellers, On the mathematics of modelling, metamodelling, ontologies and modelling languages, Springer, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-29825-7.

[2] H. Kern, A. Hummel, S. Kühne, Towards a comparative analysis of meta-metamodels, in: Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11, ACM, 2011, pp. 7–12. doi:10.1145/2095050.2095053.

[3] H. C. Mayr, B. Thalheim, The triptych of conceptual modeling, Software and Systems Modeling 2020 20 (2021) 7–24. doi:10.1007/S10270-020-00836-Z.

[4] V. Döller, Meta-concepts revisited: Looking for a common meta2model that meets the needs of the state of the art in conceptual modeling, in: H. C. Mayr, B. Thalheim (Eds.), Fundamentals of Conceptual Modeling, Mini-Dagstuhlseminar at CBI 2024, Springer, in press, 2024.

[5] P. Chen, English, Chinese and ER diagrams, Data & Knowledge Engineering 23 (1997) 5–16. doi:10.1016/S0169-023X(97)00017-7.

[6] D. Karagiannis, H. Kühn, Metamodelling Platforms, in: G. Bauknecht, K.; Min Tjoa, A.; Quirchmayer (Ed.), Proceedings of the Third International Conference on E-Commerce and Web Technologies - DEXA 2002, Springer, Berlin, Heidelberg, 2002, p. 182. doi:10.1007/3-540-45705-4_19.

[7] A. Olivé, Conceptual modeling of information systems, Springer-Verlag, Berlin Heidelberg, 2007. doi:10.1007/978-3-540-39390-0.

[8] U. Frank, Domain-specific modeling languages: Requirements analysis and design guidelines, in: I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, J. Bettin (Eds.), Domain Engineering: Product Lines, Languages, and Conceptual Models, Springer Berlin Heidelberg, 2013, pp. 133–157. doi:10.1007/978-3-642-36654-3_6.

[9] V. Döller, D. Karagiannis, W. Utz, MetaMorph: formalization of domain-specific conceptual modeling methods—an evaluative case study, juxtaposition and empirical assessment, Software and Systems Modeling 22 (2023) 75–110. doi:10.1007/s10270-022-01047-4.

[10] V. Döller, Formalizing the four-layer metamodeling stack with MetaMorph: potential and benefits, Software and Systems Modeling 21 (2022) 1411–1435. doi:10.1007/s10270-022-00986-2.