This is a postprint. The original paper appeared in Proc. 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C 2019), Workshop MULTI 2019, Munich, Germany, © 2019 IEEE, DOI 10.1109/MODELS-C.2019.00019.

# DeepTelos Demonstration

Manfred Jeusfeld
*School of Informatics*
*University of Skövde*
Skövde, Sweden
ORCID 0000-0002-9421-8566

***Abstract*—DeepTelos is defined as a set of rules and constraints that enable multi-level modeling for the Telos metamodeling language. In its ConceptBase implementation, rules and constraints are realized by Datalog clauses. We start with demonstrating first the core functions of Telos, use of simple rules and constraints, then the meta-level rules and constraints defining DeepTelos. A couple of examples show how the DeepTelos rules and constraints are compiled to simple rules and constraints and then realize the desired multi-level modeling environment. The main example is taken from the Galileo satellite domain.**

***Keywords*-multi-level modeling, ConceptBase, Telos, DeepTelos, Datalog**

## I. Introduction

DeepTelos [1] is a simple yet powerful extension to the Telos [2] language to enable a simple form of multi-level modeling. The purpose of this demonstration is to show how this extension was facilitated by the rule and constraint language of Telos, as implemented in ConceptBase [3]. Multi-level modeling (MLM) [4], [5] comprises the following building blocks:

- MLM allows to represent information at more than two abstraction levels, i.e. objects, classes, meta classes, meta-meta classes, etc.
- Classes are objects, too. They can have properties like any other object. Such classes are sometimes called clabjects.
- MLM minimizes the accidental redundancy by defining properties at the right class level and use them for all instances of that class.

There are many proposals for MLM that may exceed the DeepTelos functionality. This paper does not attempt to compare DeepTelos to them. Instead, we focus on the particular strength of DeepTelos, namely that it is a straightforward extension of the Telos language that has been used for metamodeling since the late 1980-ties.

Subsequently, we first shortly introduce the main ideas behind Telos in its ConceptBase implementation. Then, we introduce the rules and constraints that realize DeepTelos as a MLM language. We then demonstrate a number of smaller multi-level modeling examples and show how the DeepTelos rules and constraints are compiled to ordinary two-level Datalog [6] rules. All examples are made available via the web page http://conceptbase.cc/deeptelos2. The examples carry a "Creative Commons Attribution-ShareAlike 4.0 International" license.

This research has been supported in part by the EU ISF Project A431.678/2016 ELVIRA (Threat modeling and resilience of critical infrastructures), coordinated by Polismyndigheten/Sweden, and by KK Stiftelsen Synergy project: Knowledge-driven decision support via optimization.

## II. Telos and ConceptBase

Telos essentially is a metamodeling language that has only a single quadruple fact called "proposition", denoted as P(o,x,n,y). The component x is called the source of the proposition, n is the name, y is the destination, and o is the identifier of the proposition. So, one can say that it is like RDF triples where each triple has its own identifier. For example, P(o1,o1,bill,o1), P(o2,o2,mary,o2,P(o3,o1,likes1,o2) defined two "individual" objects bill and mary by self-referential propositions and links the two by a relation. The same proposition data structure is used to define class-level statements such as P(o4,o4,Person,o4), P(o5,o4,likes,o4), which defines an object Person with a cyclic relation "likes". Objects are their classes are linked by instantiation propositions, for example P(o6,o1,in,o4), P(o7,o2,in,o4), P(o8,o3,in,o5). Here the first two propositions express that bill and mary are instances of Person. The third proposition expresses that the "likes1" link of bill is an instance of the "likes" link of Person. The object identifiers are system-generated and not meant to be human-interpreted. ConceptBase and Telos thus use a frame-like language to define objects, e.g.

```
Person with
  attribute likes: Person
end
bill in Person with
  likes likes1: mary
end
mary in Person end
```

A particular predefined object in Telos is "Proposition". Each proposition is an instance of "Proposition". It defines basic relations such as "attribute":

```
Proposition with
  attribute
    attribute: Proposition;  comment: String
end
```

Since any object is an instance of Proposition, the object Person can use the definition of "attribute" to define the likes relation. In the same manner, each object may have a comment. Note that attributes in Telos are objects, too. They can have no, one or any number of instances. Thus, the comment attribute does not have to be used by all instances of Proposition. Telos defines predicates for instantiation (x in

c), specialization (c isA d), and attribution/relations (x m/n y). In the above example, the predicates (bill in Person), (Person in Proposition), (bill likes/likes1 mary) would be true among others. Note that the attribution predicate in Telos has two labels, one from the class level, and one from the instance level.

ConceptBase supports a rule and constraint language on top of the predicates that have their initial extension from the proposition facts. Consider the following expression:

```
forall x,y/Person (x likes y) ==> (y likes x)
```

It can either be interpreted as a constraint (the likes relation must be symmetric) or as a rule (the likes relation is made symmetric by deriving missing facts). ConceptBase allows to formulate generic (=multi-level) rules such as

```
forall c/Proposition M,n,x,y/VAR
  (c symmetric/M c) and
  (x in c) and y in c) and (x M/n y) ==> (y M x)
```

Such rules range over multiple instantiation levels (Proposition, c, x) and are characterized by variable arguments in the class position of the instantiation predicate (x in c). The formula compiler maps such rules and constraints to a set of formulas that range over two instantiation levels. This is the very basis of defining DeepTelos.

## III. DeepTelos

DeepTelos is defined by five rules and one constraint in ConceptBase. Since they are compiled internally to a set of two-level formulas and then mapped to Datalog, they can be evaluated by the Datalog engine of ConceptBase. The complete definition is as follows:

```
Proposition with
  attribute
    ISA: Proposition; IN: Proposition
end

forall m,x,c/Proposition (x in c) and                (1)
 (m IN c) and not (x isA m) ==> (x ISA m);
forall x,c,d/Proposition (c ISA d) and               (2)
  (x in c)  ==> (x in d);
forall c,d,m,n/Proposition (m IN c) and              (3)
 (n IN d) and (c ISA d)  ==> (m ISA n);
forall m,x,c/Proposition (m IN c) and                (4)
 (x isA m) ==> (x in c);
forall m,mx,x,c/Proposition (m IN c) and             (5)
  :(x isA mx): and (mx ISA m)  ==> (x in c);
forall x,m,c/Proposition (m IN c) and                (6)
  (x in c) ==> not (x in m)
```

Only the last formula is a constraint since it cannot derive new facts. The first rule is the most important one: if there is an object *x* that is an instance of a class *c* for which a most general instance (m IN c) is defined, then (x ISA m) is true, provided that (x isA m) is not true. The relation ISA is for derived specializations, whereas the relation isA is the original Telos specialization. Note that all rules and constraints are reagning over more than two instantiation level. Hence, they are subject to be compiled to a set of two-level constraints by ConceptBase.

Formula (1) is the most interesting from the viewpoint of multi-level modeling: The object *m* in the predicate (m IN c) is the "most-general instance" of the class *c*. The most-general instance of a class subsumes (is super-class of) all instances of class *c*. In DeepTelos, a class has at most one most-general instance as the name suggests. If a class would have two most-general instances, then both would have the same extension (= set of instances).

The predicate (c ISA d) has the classical interpretation of a sub-class relation. One might argue that it is redudant to the Telos predicate (c isA d). We only introduce (c ISA d) here, because ConceptBase disallows the predicate (c isA d) in user-defined rules for historic reasons. The colons in :(x isA mx): means that only explicit (= non-derived) are considered.

The above six rules and constraints on DeepTelos are a variant of the definitions in [1]. They better integrate the Telos predicate (c isA d) into DeepTelos. In particular, user-defined sub-classes of a most-general instance *m* are also an instance of the class of *m*.

## IV. Demonstration

We sketch here the script of the live demonstration of DeepTelos. All steps can be demonstrated directly from this paper by clicking on the hyperlinks. You need to configure your computer to start the ConceptBase tool CBGraph for file-types *.gel, see also the documentation in http://conceptbase.sourceforge.net/CB-Mime.html. You can download and install the free ConceptBase system from http://conceptbase.sourceforge.net/CB-Download.html.

### A. Step 1: The original Telos language

We showcase the object "Proposition", which is the most general object in Telos. It defines instantiation, specialization and attribution/relations:

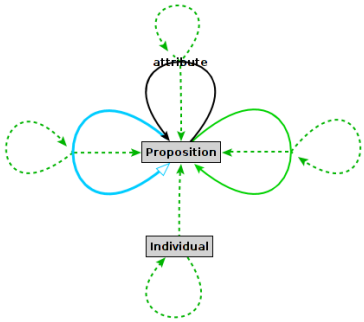


Fig. 1. The object Proposition in Telos.

The central object is "Proposition". On top is the definition of (Proposition attribute/attribute Proposition). The broken line from "attribute" to Proposition expresses that "attribute" is a derived instance of "Proposition". It is also a derived instance of itself. The link on the left is the definition of specialization (Proposition isA Proposition), which also is an instance of Proposition and of itself. In the right is the definition of (Proposition in Proposition), i.e. of instantiation. This link is

again an instance of ”Proposition” and of itself. Finally, there is a predefined object for node-like objects (Individual). These are the core objects of Telos: the so-called omega classes of Telos. The link in the caption of Figure 1 directly starts ConceptBase with the view of ”Proposition”.

### B. Step 2: DeepTelos definition

In step 2, we define the two relations IN and ISA and the 6 formulas defining DeepTelos. We also define some suitable graphical shapes for the two new relations.

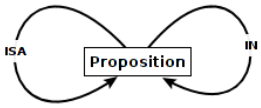


Fig. 2. The definition of DeepTelos.

### C. Step 3: The Product Hierarchy example

This is our first example (figure 3) of using DeepTelos for multi-level modeling. There are two DeepTelos hierarchies here. First, (Product IN ProductModel), (ProductModel IN ProductCategory). You should see these three objects as a unity defining three abstract levels of the concept Product. We define that products can have an owner. As a second DeepTelos hierarchy, we define (Car IN CarModel) where (CarModel in ProductCategory). Car models have a number of doors.

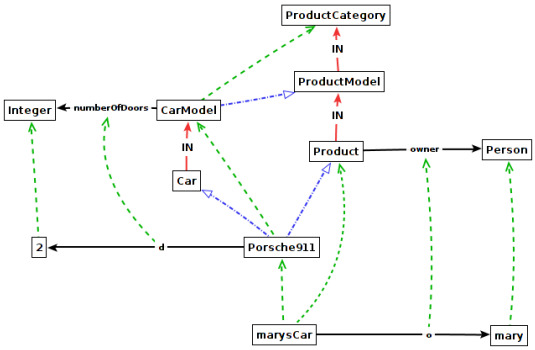


Fig. 3. DeepTelos definition of products and cars.

The specialization links, e.g. between ”Porsche911” and ”Product” are all derived via the DeepTelos rules discussed in the previous chapter. The multiple levels are defined just by the relation (m IN c). A potency-based approach would have a single concept Product and define the owner relation to have a potency of 2. In DeepTelos, this is achieved by defining the ”owner” relation at the ”Product” level. Hence, regard the three objects ”ProductCategory”, ”ProductModel”, and ”Product” as the DeepTelos counterpart of a single object ”Product” in potency-based MLM approaches. In DeepTelos, the potency is selected by picking the right level in the hierarchy (here ”Product” for the ”owner” relation).

As mentioned earlier, ConceptBase compiles the multi-level rules of DeepTelos to a set of two-level rules (Figure 4):

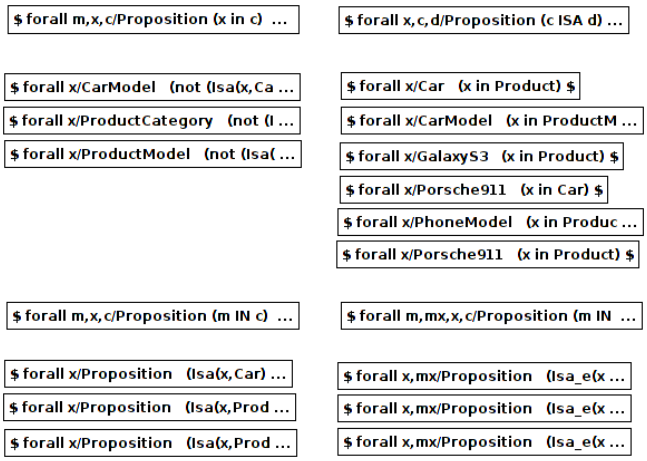


Fig. 4. Compiling DeepTelos to two-level rules.

The first DeepTelos rule leads for the product example to three two-level rules, since we have three facts matching (m IN c). The second DeepTelos rule has six instances. The third DeepTelos rule has no instances since we dont use it in the products example. Finally, rules 4 and 5 of DeepTelos lead to three compiled two-level rules each.

The combination of the compile rules carries all the semantics of DeepTelos for the product model. Hence, we could export the product model just with these compiled rules and they would behave as expected.

### D. Step 4: Entities and Values

The next example shows how DeepTelos can enrich the classical metamodeling of modeling languages, such as Entity-Relationship Diagrams. The classic definition only covers the constructs of entity types, relationship types and entity properties (=attributes). DeepTelos allows the definition of (Entity IN EntityType) and (Value IN Domain). As a consequence, one can query entities and values without using schema constructs such as Projects with a budget. For example, return all entities that are suing the value 1.5 regardless of the schema.

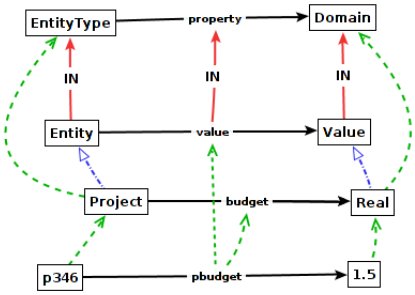


Fig. 5. Entities and values.

The demonstration shall show that the budget attribute is a subclass of the value attribute derived by the first DeepTelos rule. The project p346 is an instance of Entity and 1.5 is an instance of Value, both via the second DeepTelos rule.

### E. Step 5: Multi-level model for Satellites

This example is the most complex in terms of number of levels. We take the Galileo satellite family as an example. This family has several Galileo satellite models, some of them have only one (physical) satellite as instance. The physical satellite has physical components as parts. They are connected by interfaces, e.g. USB interfaces for certain communication connections. The physical satellite must fulfill exactly the design of its model in terms of component models and interface models. For example, the gyroscopic device of a physical satellite must be the model prescribed in the satellite model of the satellite. Satellite models are then grouped into satellite types. For example, the Galileo family forms one satellite type which has common properties such as the orbit, for which they are designed.

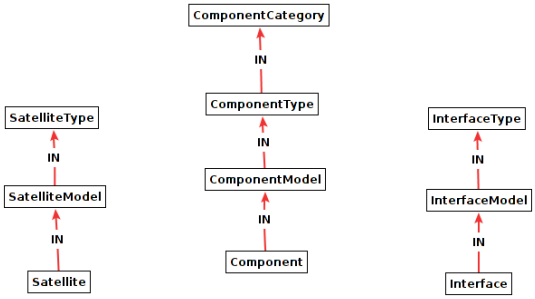


Fig. 6. Multilevel hierarchies for the satellite case.

The component hierarchy in figure 6 is the main trunk. The lowest level (Component) is used to represent actual components (having certain serial numbers). The component model is prescribing which parts a certain component model may have, e.g. a certain Galileo satellite model. Component types are for example used to specify that there are navigational satellite like Galileo satellites.

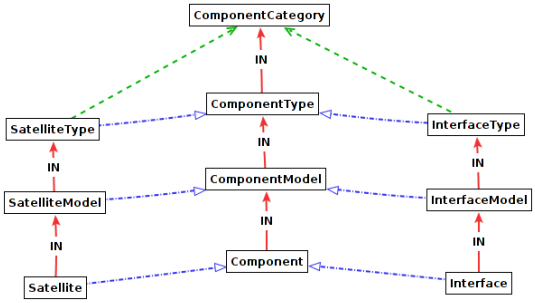


Fig. 7. Adding derived specializations.

Component categories stand for the whole variety of component types used in this example, for example satellite type and interface type. These two example component categories get their own hierarchies in order to be able to define properties like orbit to the right level. Figure 7 shows the (explicit) instantiation of ”SatelliteType” and ”InterfaceType” to ”ComponentCategory”. All specialization links are derived in figure 7 are derived by DeepTelos rules. For example, the fourth DeepTelos rule derives that SatelliteModel (being a specialization of itself) is also an instance of ”ComponentType”. Then, it must be a specialization of ”ComponentModel” as well via the second DeepTelos rule. DeepTelos has no explicit level numbers and no potencies for attributes and relations. Yet, if the multilevel hierarchies are connected by specialization links such as in figure 7, the level numbers can be calculated: instances of ”Component” (and also of ”Satellite” and ”Interface) have level 0, instances of ”ComponentModel” (and also of ”SatelliteModel” and ”InterfaceModel) have level 1, instances of ”ComponentType” have level 2, and instance of ”ComponentCategory” have level 3.

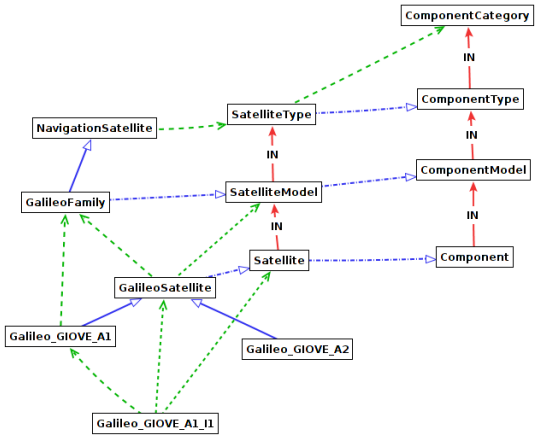


Fig. 8. Adding derived specializations.

The example in figure 8 shows the instance I1 of the Galileo Giove A1 satellite model. The solid specialization links are explicit (user-defined) specializations. So, a navigation satellite is a satellite type and the Galileo family is a specialization of it (inherits all properties). Giove A1 is an instance of the Galileo family of satellites. Via DeepTelos rules, it becomes also an instance of satellite model (and thus can use the properties that are defined there). Giove A1 is also an explicit specialization of GalileoSatellite (an instance of GalileoFamily). The instance I1 of the Giove A1 is then also an instance of GalileoSatellite and satellite. The derived instantiations are needed for being able to instantiate the multi-level attributes as discussed in figure 9.

In figure 9, navigation satellites have a property transmitterWatt (for sending the GPS signal). All satellite models have a specified start mass and a planned orbit. The GIOVE A1 satellite uses the first two attributes. All Galileo satellites are designed for the same orbit (3612). The instance I1 of GIOVE A1 happens to fly on the planned orbit (though it could also fly on another orbit due to a malfunction in the rocket or the gyroscopic control).

Figure 10 shows the use of the multi-level satellite model to design the configuration of satellites. The satellite model Giove A1 has 1 gyroscopic device (product model GyroMasterV17) and three identical reaction wheels (model ReaFastV45). The instance I1 instantiates these parts according to the satellite model Giove A1. The Galileo family of satellites has among

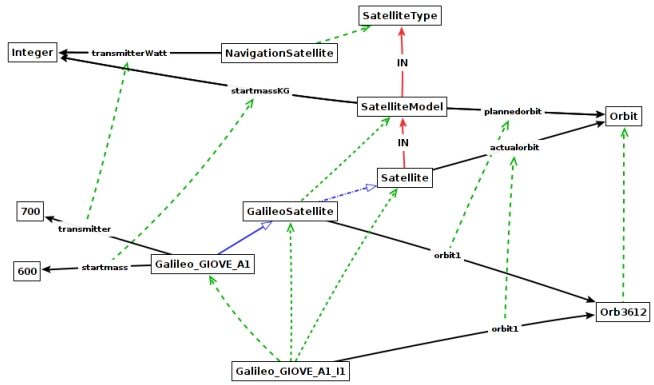


Fig. 9. Multi-level properties of satellites.

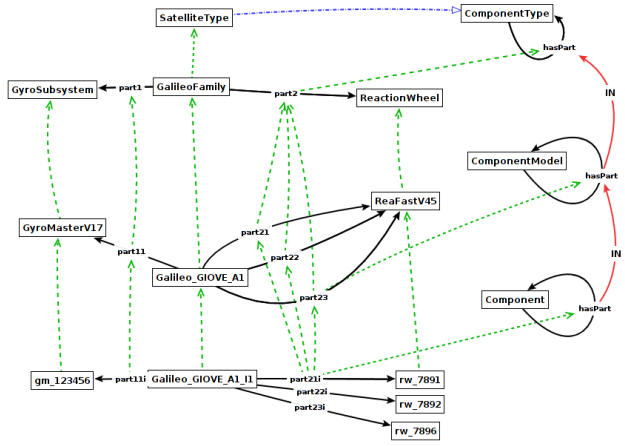


Fig. 10. Configuration of satellites.

others a gyro sub system and reaction wheels (possibly more than one). On the right hand side, you see the ”hasPart” relation for component types. The two part types of GalileoFamily are instances of the ”hasPart” relation of ”ComponentType”. By using the DeepTelos construct (m IN c) for this relation, we can inherit the ”hasPart” relation downwards to ”ComponentModel” and ”Component”. The generated rules of DeepTelos make sure that the lower level ”hasPart” relations are instantiated via these rules. Thus, one can check use the ”hasPart” relation of ”Component” to list all parts of a component like Galileo_GIOVE_A1_I1.

The final step of the demonstration is to show the two-level rules generated from the 5 DeepTelos rules. The following query shall compute it:

```
GeneratedRules in QueryClass isA Class with
  computed_attribute genrule : MSFOLrule
  constraint c1 : $ (this rule genrule) and
          exists mlrule/MSFOLrule
            (DeepTelosRules rule mlrule)
            and  :(genrule isA mlrule): $
end
```

It turns out that the satellite example leads to about 100 such generated rules.

## V. Summary

We demonstrated the use of DeepTelos for multi-level modeling. DeepTelos is defined by just 5 multi-level rules plus one constraint. The key idea is the use of a special construct (m IN c) that related a class *c* to its ”most-general instance” *m*. Instead of numbers as potencies, DeepTelos uses actual names for the levels such as ”Component” (level 1), ”ComponentModel” (level 2), and so on. The multi-level attributes are then defined at the right level of such a hierarchy of levels. Several levels can co-exist and be related to each other. We also note that cross-level relations are supported such as the planned versus actual orbits of satellite models/satellites.

The demonstration can be easily replayed by installing ConceptBase (see link at the beginning) of the paper and then opening the links to the graph files below the figures in this paper. We did not compare DeepTelos in this paper to other multi-level modeling tools. We plan however to represent the multi-level challenge models (bicycle, process models) in DeepTelos to facilitate a comparison. DeepTelos does have its limitations. If a model has a deep specialization hierarchy, then a large number of formulas are generated from the DeepTelos formulas. This slows down the compilation but also the runtime performance, e.g. for adding an instance to some class. It should be noted that all levels can be updated incrementally, which can be a costly operation when many instances already exist.

## VI. Acknowledgements

Special thanks to Bernd Neumayr, who co-developed DeepTelos. Further thanks to Philipp Martin Fischer, with whom I attended the Dagstuhl Seminar 17492 and who inspired the satellite example used in this demonstration. Finally, many thanks to René Soiron, who helped implement the compiler for multi-level rules in the early 1990-ties.

## References

[1] M. A. Jeusfeld and B. Neumayr, “DeepTelos: Multi-level modeling with most general instances,” in *Conceptual Modeling - 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings*, 2016, pp. 198–211. [Online]. Available: https://doi.org/10.1007/978-3-319-46397-1_15

[2] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, “Telos: Representing knowledge about information systems,” *ACM Trans. Inf. Syst.*, vol. 8, no. 4, pp. 325–362, 1990. [Online]. Available: http://doi.acm.org/10.1145/102675.102676

[3] M. Jarke, R. Gallersdörfer, M. A. Jeusfeld, M. Staudt, and S. Eherer, “ConceptBase - a deductive object base for meta data management,” *J. Intell. Inf. Syst.*, vol. 4, no. 2, pp. 167–192, 1995. [Online]. Available: http://dx.doi.org/10.1007/BF00961873

[4] C. Atkinson and T. Kühne, “The essence of multilevel metamodeling,” in *UML 2001 - The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001, Proceedings*, 2001, pp. 19–33. [Online]. Available: https://doi.org/10.1007/3-540-45441-1_3

[5] J. P. A. Almeida, U. Frank, and T. Kühne, “Multi-level modelling (dagstuhl seminar 17492),” *Dagstuhl Reports*, vol. 7, no. 12, pp. 18–49, 2017. [Online]. Available: https://doi.org/10.4230/DagRep.7.12.18

[6] S. Ceri, G. Gottlob, and L. Tanca, “What you always wanted to know about Datalog (and never dared to ask),” *IEEE Trans. Knowl. Data Eng.*, vol. 1, no. 1, pp. 146–166, 1989. [Online]. Available: https://doi.org/10.1109/69.43410