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.2019.00016.

DeepTelos for ConceptBase: A contribution to the MULTI process challenge

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

***Abstract*—DeepTelos is a straightforward extension of the Telos modeling language to allow some form of multi-level modeling. A variant of Telos has been implemented in the ConceptBase system on top of a Datalog engine. Telos defines the concepts of instantiation, specialization and attribution/relations by means of axioms. In addition, the user can define new constructs by deductive rules, integrity constraints, and so-called query classes. In this paper, we tackle the process challenge formulated for the MULTI 2019 workshop to see to which extent DeepTelos is able to represent the requirements of this challenge.**

***Keywords*-multi-level modeling, Telos, process model, ConceptBase, Datalog**

I. Introduction

DeepTelos [1] was originally defined by just three deductive rules extending the existing Telos [2] constructs attribution, instantiation, and specialization. These 3 rules were later extended to five rules and one constraint to better integrate the derived specializations of DeepTelos and the existing Telos specialization axioms. The core idea of DeepTelos is to exploit the powertype pattern [3] via a rule: if a class c has a most general instance m, then any instance of c is a subclass of m. The most general instance m serves as a proxy for class c at one instantiation level lower. It has all instances of instances of c as its instances. The class m itself can have another most general instance $m1$, which serves as a proxy for m at even one instantiation level lower. So, the lattice of most general instance relations, denoted as (m IN c), spans a family of modeling levels. ConceptBase [4] is a multi-user database system for managing all kinds of models and metamodels. It implements its logical component (rules, constraints, queries) via a Datalog-neg engine. It also features a graphical user interface.

A. Multi-level rules and constraints

We shall use the latest revision 2 of DeepTelos for answering to the MULTI process challenge. This revision provides a better interplay of the DeepTelos rules with the built-in axioms for specialization, attribution/relations, and instantiation. Before listing the DeepTelos constructs and axioms, we need to introduce a mechanism of ConceptBase that allows to partially evaluate logical expressions that span of more than two instantiation levels. As an example, consider the following formula that expresses that a relation p is necessary (= multiplicity 1...*):

```
forall c,d/Proposition p/Proposition!necessary
   x,m/VAR  P(p,c,m,d) and (x in c) ==>
        exists y/VAR (y in d) and (x m y)
```

The proposition predicate P(p,c,m,d) expresses that there is a relation identified by p with label m between the objects c and d. The variables c and d stand for any instance of the class "Proposition", which is the most general class in Telos (also called the omega-level of Telos). The proposition p is an instance of Proposition!necessary (the necessary relation of "Proposition"). Then, if there is any instance x of c, it must provide a value y for the relation m: (x m y). While the details of the formula are not of great interest here, you note that the variable x and y are instances of classes c and d which themselves are variables (being instances of "Proposition"). This shows that the formula ranges over three levels: first, the level of "Proposition", second, the level of c and d, and third, the level of x and y. We call such formulas multi-level formulas. They are partially evaluated in ConceptBase for two reasons:

1) A formula with many such variables is costly to evaluate since some of the predicates match many facts in the database. For example (x in c) matches any instantiation fact.
2) Without partial evaluation, it is almost impossible to find stratifications of predicates in the presence of recursive rules with negation.

The partial evaluation picks a so-called binding path that provides values for the variables at class positions, here c and d. In this case, a possible binding path is (p in Proposition!necessary) and P(p,c,m,d). Any solution of this conjunction provides fillers for the variables p, c, m, and d. Assume, we have a definition of a class like

```
Employee with
   necessary, attribute
      name: String
end
```

Then, the following facts are true:

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.

P(id123,Employee,name,String) and (id123 in Proposition!necessary). By matching these facts with the binding path, we can partially evaluate the multi-level formula to

```
forall x/Employee exists y/String (x name y)
```

Multi-level rules and constraints are heavily used in DeepTelos. Note that there may be more than one binding path and the selection of the best candidate is in general not a trivial task. The partial evaluation maps a multi-level formula to a set of formulas, one for each solution of the binding path expression. Since the set of solutions can grow and shrink as the database is updated, ConceptBase needs to maintain the set of partially evaluated formulas.

A related approach to handle multi-level formulas is the Diagram Predicate Framework (DPF) [5]. Telos/ConceptBase adopt the minimal-model Datalog semantics while DPF is based on an algebraic specification of semantics over nodes and links of a multi-level graph. It should be noted that the Telos ”Proposition” object is subsuming nodes, attributes, relations, instantiations, and specializations. It allows to add for example attributes to an instantiation link.

B. DeepTelos Revision 2

DeepTelos Revision 2 is defined by 5 deductive rules and one constraint. The rules are defined in terms of two new constructs IN and ISA defined for any proposition (see appendix)

The first rule is the main rule: If there is a most general instance *m* of *c* (m IN c) and instance *x* of *c* and (*x* is not already derivable to be a (Telos) specialization of *m*, then (x ISA m) is derived, i.e. all such instances become (DeepTelos) specializations of the most general instance *m*. Note that the symbol arrow in the formula stands for the logical implication.

```
forall m,x,c/Proposition                          (mrule1)
  (x in c) and (m IN c) and not (x isA m)
    ==> (x ISA m)
```

The full definition of DeepTelos can be found at http://conceptbase.cc/deeptelos and can directly be used with the ConceptBase system available at http://conceptbase.cc. All sources of the DeepTelos solution for the MULTI 2019 ”Process” Challenge can be found at http://conceptbase.cc/multi2019challenge. Note that all figures in this paper followed by web links allow to directly upstart the model with the ConceptBase graph editor. Those figures are in fact screendumps of the ConceptBase graph editor. Instructions or provided on the above web page.

C. DeepTelos Car Example

To motivate the use of DeepTelos, consider the following simple scenario. A car has a model number and a mileage. In potency based approaches to multi-level modeling [6]–[9], one would have a meta class ”CarModel” (M2 level) with two attributes. The attribute model number would be applicable to instances of ”Car”, i.e. classes at M1 level (potency 1). The attribute mileage would be applicable to instances of instances of ”Car” (Mo level, potency 2). In DeepTelos, the potency levels are replaced by most general instances:

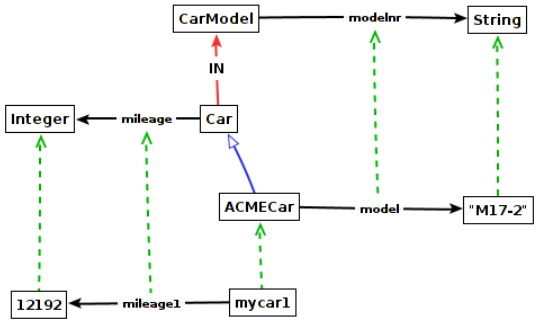

Fig. 1. DeepTelos solution for the car example.

The object ”CarModel” is a meta class with the most general instance ”Car”. The model number is applicable to instances of ”CarModel”, whereas the mileage is applicable to instances of ”Car”. In the above example, ”ACMECar” is defined as explicit (Telos) subclass of ”Car”. By DeepTelos rule ”mrule4” (see appendix), it is then also an instance of ”CarModel” and thus may use the ”modelnr” attribute. The object ”mycar1” is then an instance of ”ACMECar” (and thus of ”Car”) and can instantiate the mileage attribute. The concepts ”Car” and ”CarModel” shall be regarded as one aggregated concept. The purpose of the most-general instance ”Car” is to be able to define attributes like mileage that are applicable to all cars.

For supporting model reuse, we employ the module system of ConceptBase to organize the model definitions and separate variants of models. The module system in ConceptBase supports tree-like sub-module hierarchies where a sub-module ”sees” all definitions made in the super-module but not the definitions in sibling modules. There are two predefined modules in ConceptBase: the root module ”System” contains all built-in objects of Telos and ConceptBase, in particular the omega class ”Proposition”. The ”System” module has a submodule ”oHome”, which contains the definitions made by a user of ConceptBase.

We shall use the ”oHome” module for the definition of some useful formulas such as the necessary constraint discussed earlier. Inside, the ”oHome” module, we define the module DeepTelos, which includes the 5 rules and the one constraint discussed above plus some graphical types for a nicer visualization of DeepTelos most-general instance hierarchies. Then, the DeepTelos module contains a submodule ”ProcessModels” which contains the solution for the requirements P1-P19 of the challenge. Then, the submodule ”CodingProcess” of ”ProcessModels” contains the example process type of the MULTI ”Process” challenge. Finally, the submodules ”Process1”, ”Process2”, and ”Process3” of ”CodingProcess” contain example processes of the process type defined in ”CodingProcess”. The car example of figure 1 is stored in another sub-module of DeepTelos, thus sharing the definitions of DeepTelos, ”oHome” and ”System” but not of ”ProcessModels” and its submodules.

II. Analysis of the Multi process challenge

The challenge defines three main concepts. There are *tasks* and task types. Task types are used to define process models (roughly M1 level). The process models can be instantiated and deliver processes (M0 level). Thus, a process is the trace of the execution of a process model for a given case (e.g. to develop a software system). The second main concept is the *actor* type, resp. actor. Actors types are related to task types, e.g. to define the required competence of an actor to execute a task type. Thirdly, there are *artifacts* and artifact types. They define the inputs and outputs of task types (M1 level) and of their instances (M0 level). At the M1 level, one defines the types of inputs and outputs whereas the M0 level defines which actual input and output artifacts were used in a specific execution of the process. Some attributes and relations link objects at the same instantiation level, while others links objects at different abstraction levels. For example, one can authorize a single actor like AnnSmith (M0) to the sole person allowed to execute a given ask type such as "CodingInCobol". A particular such cross-level attribute is the language attribute to define the label of a concept in multiple languages. Our solution will allow to do so for any concept at any instantiation level!

While DeepTelos does not have predefined levels such as M0, M1, M2, M3 and so forth, it is still useful to roughly identify objects that a OMG-educated modeler would assign objects to.

M0: Here we find objects like "AnnSmith", task instances such as coding1 as instance of "CodingInCobol", which is started at a given date and ends at another date. You also find artifact instances such as "cobolprogram1". It should be noted that such artifacts can contain objects at a higher instantiation level. For example, a design document can contain a whole UML class diagram (M1) level. We have discussed this phenomenon earlier in the context of process-data diagrams. The solution there did however not use DeepTelos, but declared certain objects to be both instance and specialization of meta class.

M1: This level contains the specification of a process type such as the ACME coding process type of the challenge. This level roughly corresponds to a BPMN process model. In contrast to BPMN, our solution (and the challenge) also covers the execution of the process model at M0. In our solution, the concepts "Task", "Artifact", "Actor", and "Process" are all at M1 level.

M2: The M2 level defines constructs such as "ArtifactType" (having the most general instance "Artifact"), "TaskType" (most general instance "Task"), "ActorType" (most general instance "Actor"), and "ProcessType" (most general instance "Process"). Certain subclasses of these classes are also defined at this level such as "CriticalTaskType". We re-use the definition of a core BPMN language at the M2 level and pull it from the ConceptBase Forum. This simplifies the solution to the challenge since there is no need to re-invent the wheel.

M3: The M3 level contains the meta-meta classes "Node", "NodeOrLink" and the link object (label "connectedTo"). We re-use these definitions since the core BPMN language was defined with these meta-meta classes. Omega: We make heavy use of the omega-class "Proposition" as discussed in the introduction to define DeepTelos and to solve the "language" attribute. The omega level also contains the implementations of multiplicity constraints such as "necessary" (1..*) and single (0..1), see also the sources in the appendix.

The challenge demands a peculiar property of task types, namely the planned duration. In our solution, we shall allow to compare the planned duration (task type) with the actual duration of a task instance (derived from start and finish date). Function definitions are used for that purpose. We also define a handful of queries to analyze process models and their execution.

III. Model presentation: The constructs

This chapter introduces the solution of the MULTI process challenge by DeepTelos and ConceptBase. Most figures are accompanied with links that lead to the executable graph files to be processed by the ConceptBase graph editor.

A. The levels

Figure 2 shows the levels used for the solution. On the top left is the object Proposition defining the two DeepTelos relations IN (for relation a most general instance to its class) and ISA (for derived specialization relations). These two relations extend the Omega-level of Telos. All explicit objects in this diagram including all nodes and links are instances of "Proposition". The next level is established by the objects "Node", "NodeOrLink" and the "connectedTo" link of "NodeOrLink". In traditional metamodeling, these objects would be dedicated as metameta classes, i.e. members of the M3 level. It is used to define modeling languages such as BPMN. In the figure, the object "BPMN_Element" is the superclass of all constructs of a core BPMN metamodel that we utilize in this solution1. Since this metamodel does not cater for all constructs needed for the challenge, we also define a meta class "ProcessElementType", which subsumes all required constructs including the core BPMN constructs.

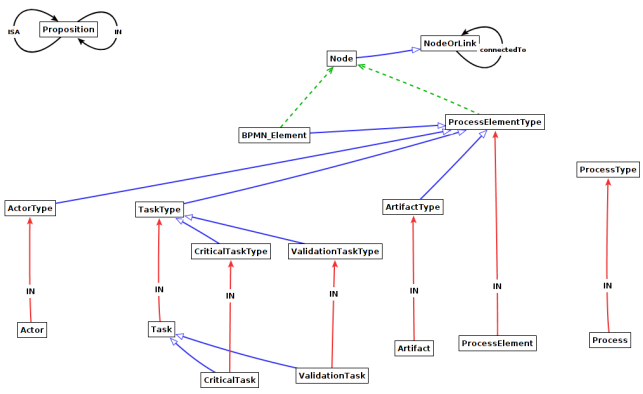

Fig. 2. The DeepTelos levels for the MULTI process challenge.

All subclasses of ”ProcessElementType” plus the class ”ProcessType” would be regarded as M2 level in an UML environment. The next levels is formed by the most general instances ”Actor” (of ”ActorType”), ”Task” (of ”TaskType”), ”Artifact” (of ”ArtifactType”), ”ProcessElement” (of ”ProcessElementType”) and ”Process” (of ”ProcessType”). Hence, ”Task” is a simple class (regared as M1 level in UML) and has all instances of instances of ”TaskType” as instance, as derived via the DeepTelos rules. The instances of the displayed most-general instances for the M0 level. We will see such instances in the section for the example processes. In summary, the solution features 4 UML-ish abstraction levels plus the omega level.

B. Requirements P1-P3

Figure 3 shows the solution to requirements P1-P4. ”ProcessType” is modeled as a container for ”ProcessElementType” which subsumes all required constructs (including gateways, start/end elements and task types). The ordering of the process element types is done by the next relation of ”BPMN_Element”. The two subclasses ”PlaceLike” and ”TransitionLike” are used to embed BPMN into a petri net semantics, which we do not use in this solution but can be inspected from the link given in the footnote. The figure also shows the object ”Process” as most general instance of ”ProcessType”. It also has a contains relation, which is the most general instance of the contains relation of ”ProcessType”. Hence, the use of the contains relation at the M2 level propagates to the M1 level. Just as we define process models as containers of process element types, we define process as containers of process elements.

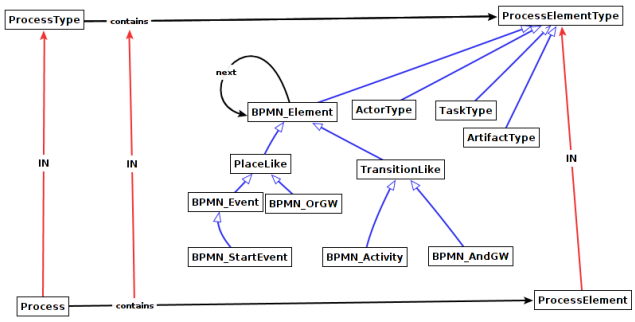

Fig. 3. Requirements P1-P3.

Note that the above link starts of the graph editor from which the source code can be inspected. The source code is also available via the link provided in the introduction.

C. Requirements P4-P6

These requirements introduce the first cross-level relations between ”Actor” (M1) and ”TaskType” (M2).

The relation ”creator” links a task type to the actor, who created it. Figure 5 also shows an instance of this relation. The second cross-level relation is ”executorset”, which is used to address require P6 (”task type may alternatively be assigned to a particular set of actors”). Note that all relations in Telos are by default set-valued. The relation ”executortype” links ”TaskType” and ”ActorType” (requirement P5).

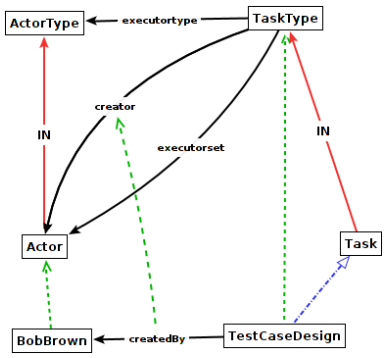

Fig. 4. Requirements P4-P6.

D. Requirements P7-P9

Task types use and produce artifact types. This is also propagated to tasks (which use and produce artifacts).

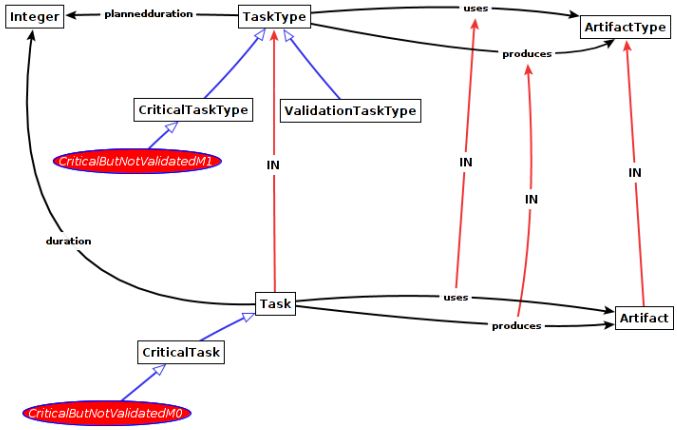

Fig. 5. Requirements P7-P9.

Critical task types and validation task types are modeled as subclasses of task type. The query class ”CriticalButNotValidatedM1” checks requirement P9. This query returns all critical task types that are not checked by a validation task type. A similar query ”CriticalButNotValidatedM0” is defined for the most general instance ”CriticalTask”. The code for both query classes in in the appendix.

The latter query operates at the M0 level, i.e. checks actual executions of the process rather than the process type. In addition constraints (see class ”CriticalTaskType” in the appendix) are used to address the actor requirements of P9.

Finally, figure 5 shows the planned duration (requirement P8) and the actual duration (property of Task). The latter is not demanded by the challenge but we found it useful to later check whether a task is delayed.

Requirement P10 (Each process type may be enacted multiple times.) is automatically fulfilled by Telos since any process

type (instance of ”ProcessType”) may itself have any number of instances. We shall later provide to instances of the ACME example process type. Requirement P11 is fulfilled by the ”contains” relation of ”Process”, see figure 4. ”Task” is a subclass of ”ProcessElement”. For requirement P13 (Tasks are associated with artifacts used and produced, along with performing actors.), we also refer to figure 4.

Requirement P12 (begin and end of a task) is implemented by two attributes ”begindate” and ”enddate” of ”Task”:

```
Task in Class with IN  class: TaskType
  attribute
    uses : Artifact; produces : Artifact;
    begindate : Integer; enddate : Integer;
    duration : Integer; executor : Actor
end
```

The full definition is in the appendix. It calculates the actual duration of a task by a deductive rule based on the function ”taskDuration”. Since the task type of a task has a ”plannedduration”, we can retrieve delayed tasks by a query class ”DelayedTask” (see appendix).

This approach also allows to define an derived attribute avgduration of TaskType, which is the average of all durations of its instances (not implemented here). This supports the idea of a datawarehouse-like aggregation of class-level attributes from instance-level attributes.

Requirement P14 is realized by the query class ”UnmatchedTask” (see appendix).

Requirements P15 and P16 is also fulfilled by Telos since each object may have multiple classes. Hence, any actor can have multiple actor types. A similar argument holds for requirement P16. The processes ”Process1” to ”Process3” have examples for such objects that have multiple classes.

E. Requirements P17-P18

The requirement demands that each actor who performs (=executes) a task must be authorized to do so. The requirement is related to the executortype and executorset relations of ”TaskType”.

We have no complete solution to this requirement since there are some default rules (when authorizations are not defined).

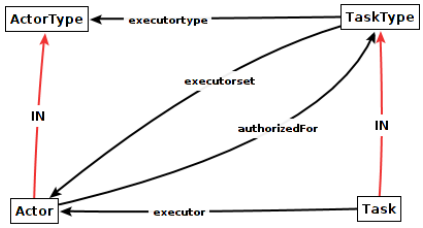

Fig. 6. Requirements P17.

Figure 6 shows the relevant defined relations. The authorization is then expressed by a Telos constraint ”isAuthorized” of class ”Actor” (see appendix).

Note that the query class ”AuthorizedTask” skips those tasks that have no authorization declared. The solution is only about half of what should be expected by a proper authorization schema. So, this solution is incomplete. We argue that it could be extended but foresee a rather complicated specification by Telos rules and constraints involving negated predicated to model defaults.

Requirement P18 (Actor types may specialize other actor types, in whichcase, all the rules that apply to instances of the specialized actor type must apply to instances of the specializing actor type.) is again easily fulfilled by Telos. All instances of subclasses are also instances of superclasses in Telos. Consequently, rules and constraints applicable to instances of the superclasses also apply to instances of the subclasses.

F. Requirements P19

This requirement demands that artifacts, actors, tasks and their types can have alternative names in international languages. We solve this by allowing any object to have such alternative names in any number of languages:

```
Proposition with
   attribute altname : String
end
```

So, we can for example specify:

```
Coding with
  altname
    de: "Programmierung"; se: "programmering";
    es: "programacion"; en: "coding"
end
```

The object ”Proposition” is at the omega level. A builtin axiom of Telos instantiates all explicit objects to ”Proposition”. So, we can apply the ”altname” construct to virtually any model element, regardless of the instantiation level. This discussion completes the first set of requirements P1-P19 of the MULTI process challenge. Subsequently, we present the software engineering process (an instance of ”ProcessType”) and the related requirements S1-S13.

IV. Model Presentation: Example Process

The example ACME software engineering process of the challenge is modelled using the capabilities of the core BPMN language implemented by a Telos metamodel:

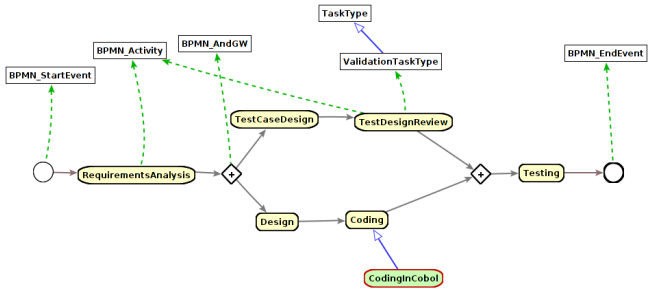

Fig. 7. The ACME process type in our solution.

he representation of the sequencing between the start element, the task types, the gateways, and the end element is done by using the next relation of BPMN_Element (see figure

3). The task type ”CodingInCobol” is defined as a subclass of ”Coding”. This is used for a variant of the process type. Figure 3 uses BPMN-style graphical shapes where appropriate.

A. Requirements S1-S4

These requirements read as:

- S1: A requirements analysis is performed by an analyst and produces a requirements specification.
- S2: A test case design is performed by a developer or test designer and produces test cases.
- S3: An occurrence of coding is performed by a developer and produces code. It must furthermore reference one or more programming languages employed.
- S4: Code must reference the programming language(s) in which it was written.

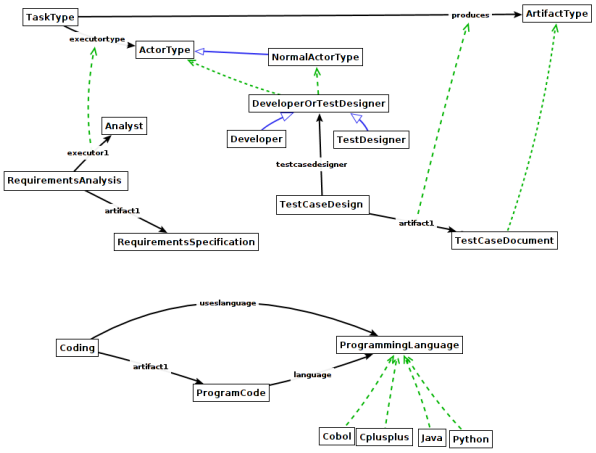

Fig. 8. Requirements S1-S4.

The upper part of figure 8 shows the constructs for ”TaskType”, ”ActorType” and ”ArtifactType” (M2 level) . The executor of the task is assigned using the ”executortype” relation of ”TaskType”. The produced artifact type, e.g. ”TestCaseDocument” (M1 level) uses the ”produces” relation of ”TaskType”. The artifact type ”ProgramCode” has a ”language” attribute. A similar attribute is defined for the ”Coding” task type.

To understand the instantians derived by Telos, consider the object ”DeveloperOrTestDesigner”. It is defined as an (explicit) instance of ”NormalActorType”, which is an explicit specialization of ”ActorType”. Via DeepTelos rule mrule4 (appendix) it is then also in instance of ”ActorType”. This instantiation then allows to use the ”executortype” relation for linking TestCaseDesign to its executor ”DeveloperOrTestDesigner”. Note that the figure does not show all instantiations to keep it readable.

B. Requirements S5-S7

These requirements demand that the task type ”CodingInCobol” always produces Cobol code and that Cobol is the languages used in this task type. Moreover Ann Smith is a developer and the only person authorized to execute this task type.

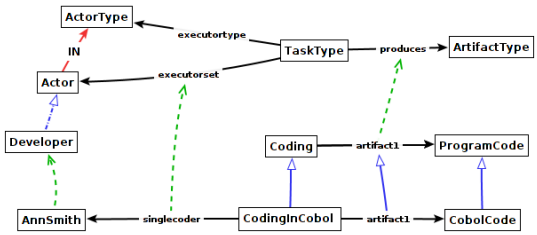

Fig. 9. Requirements S5-S7.

In the DeepTelos solution, the ”Coding” process produces ”ProgramCode” (relation artifact1). This is specialized to ”CobolCode” for ”CodingInCobol”. Ann Smith is the actor developer, who can execute ”CodingInCobol”. The language Cobol is prescribed to ”CodingInCobol” by a constraint:

```
CodingInCobol in BPMN_Activity isA Coding with
   constraint
    useCobol : $ forall cic/CodingInCobol
                   (cic useslanguage Cobol) $
  executorset singlecoder : AnnSmith
  produces artifact1 : CobolCode
end
```

C. Requirements S8-S10

The Testing task type shall be performed by a ”Tester” and it shall produce a ”TestReport” as artifact. A test report is associated to other software engineering artifacts produced by other tasks.

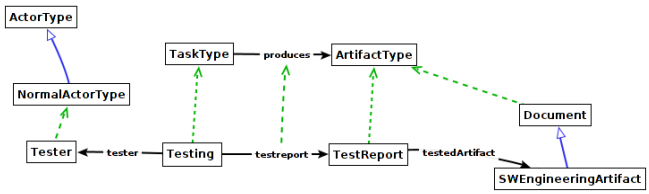

Fig. 10. Requirements S8-S9.

In the DeepTelos solution, ”SWEngineeringArtifact” is the superclass of all artifact types produces by the ACME process. It is a subclass of ”Document”, which is an instance of ”ArtifactType”. The DeepTelos rules then derive that ”SWEngineeringArtifact” is also an instance of ”ArtifactType”.

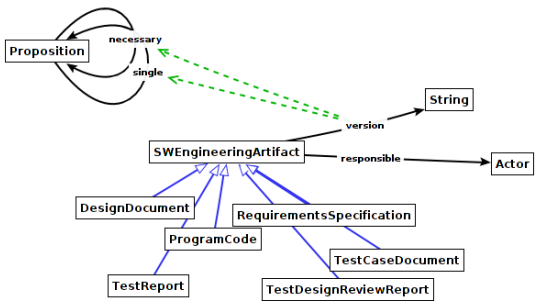

Fig. 11. Requirements S10.

Figure 11 shows the solution to requirement S10: Software engineering artifacts have a responsible actor and a version

number. The latter is defined to be necessary (1..*) and single-valued (0..1). These two cardinilities are defined by appropriate multi-level formulas in the oHome module as discussed in the introduction.

D. Requirements S11-S12

The actor Bob Brown is a tester and analyst. He is also the creator of all ACME task types. Furthermore, the expected duration of testing is 9 days.

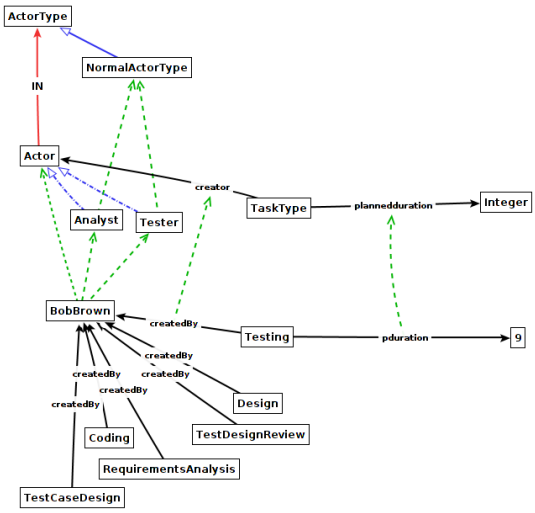

Fig. 12. Requirements S11-S12.

Again, the DeepTelos rules take care of the necessary derived instantiations and specializations: ”Analyst” and ”Tester” are both instances of ”NormalActorType”, which is a specialization of ”ActorType”. Then, both ”Analyst” and ”Tester” are derived instances of ”ActorType”, and consequently ”Analyst” and ”Tester” are derived specializations of ”Actor”. Then, ”BobBrown” becomes an instance of ”Actor” as well and can instantiate the ”createdby” attribute of ”TaskType”. ”BobBrown” is associated as creater of all ACME task types.

E. Requirements S13

The requirement reads as: ”Designing test cases is a critical task which must be performed by a senior analyst. Test cases must be validated by a test design review.”

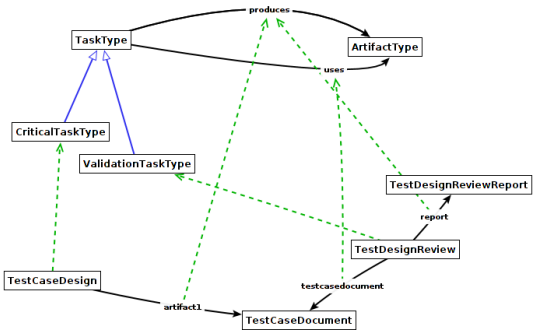

Fig. 13. Requirements S13.

The test case document is produced by ”TestCaseDesign” (a critical task) and used by ”TestDesignReview” (a validation task). The requirement that the executor must be a senior analyst is addressed by a constraint of class ”TestCaseDesign”:

```
forall tcd/Task a/Actor (tcd in TestCaseDesign)
    and (tcd executor a) ==> (a in SeniorAnalyst)
```

The complete definition of ”TestCaseDesign” is in the appendix. This completes the discussion of the ACME case requirements. Note that the definitions are mostly at the M1 level (simple classes). Actual executions of the ACME process type deliver instances of the tasks, i.e. objects at the M0 level. We provide three submodules ”Process1”, ”Process2”, and ”Process3” for such examples. The first is an example of the standard ACME process, the second uses the variant with ”CodingInCobol”, and the third adds contents to the software engineering artifacts such as actual lines of code to the instances ”CobolCode”. The third submodule ”Process3” shows how to trace dependencies between artifacts including dependencies between individual model elements, e.g. between specific lines of code and specific requirements.

V. Example Process Traces

Instances of tasks form traces of the executions of process types such as the ACME process. Actual instances of tasks have a begin date, a finish date, and are performed by suitable actors. They also produce and use instances of the artifact types specified in the ACME process type. The process instances are at the lowest abstraction level (M0), though artifacts may actually contain models (compare [10]) such as a UML class diagram. The links below point to the ConceptBase graphs for the three example process traces:

- http://conceptbase.sourceforge.net/multi2019challenge/process1.gel is an instance of ”ACMEPlainProcess”. The example violates the ”CriticalBut-NotValidatedM0” test.
- http://conceptbase.sourceforge.net/multi2019challenge/process2.gel is an instance of ”ACMECobolProcess”. This trace passes the ”CriticalBut-NotValidatedM0” test.
- http://conceptbase.sourceforge.net/multi2019challenge/process3.gel shows how transitive dependencies between artifacts and their contents are managed.

VI. Discussion

DeepTelos is a straightforward extension of Telos and defined by just 5 first-order rules interpreted by a Datalog engine.

A. Basic modeling constructs

The basic modeling constructs of Telos are instantiation (x in c), specialization (c isA d), and attribution/relations (x m y). DeepTelos adds two more basic constructs:

(m IN c) : This constructs declared the object m as most-general instance of the class c. Any instance of c shall then be a subclass of m, and vice versa.

(c ISA d) : This is a second construct to declare c as specialization of d. We technically need this construct because the ConceptBase implementation of (c isA d) forbids deriving specializations by a rule.

The two constructs and the five associated multi-level rules and one constraint realize a simple multi-level modeling environment. Basic modeling constructs are defined as attributes of the omega class "Proposition" in Telos/ConceptBase. Since this class can be extended by a user of ConceptBase, any user can add new modeling constructs. In the case of this challenge, we defined an attribute "altname" for "Proposition". This allows to assign alternative names to any Telos object, not just the objects used in this solution.

B. Employed levels

DeepTelos has no level numbers and no potencies. Instead, levels are introduced by declaring a relation like (Task IN TaskType). So, the instances of "TaskType" form one level and the instances of "Task" employe another level below "TaskType". The main levels of this solution are shown in figure 2. As discussed earlier, one can identify the 4 UML-ish levels M0 to M3 in our solution plus the omega level ("Proposition"). In this solution, we had no chain of most-general instances such as (m1 IN m2), (m2 in m3). This indicates that there are only potencies 1 and 2 used for attributes in our solution. We might have used "Node" as a level on top of "ProcessElementType" to have such a chain. However, it was not needed to solve the challenge, so we kept it out. DeepTelos spans levels by the (m IN c) predicate, but they are existing in parallel and have no static level number. As shown in http://conceptbase.cc/deeptelos, one can define objects: (M0Object IN M1Object), (M1Object IN M2Object), (M2Object IN M3Object) to force objects into UML-ish levels. There is however no apparent advantage to do so. On the contrary, one may run into problems to define cross-level relations.

C. Cross-level relationships and cross-level constraints

Such relationships were always possible in Telos. DeepTelos makes this feature even more useful, since such relations can be defined between objects that stand in most-general instance relation. For example, the object "TaskType" has a relation "createdBy" to "Actor". "Actor" is a most-general instance of "ActorType" and is related to "TaskType" by other (same-level) relations. As written earlier, we have formally no static level numbers. Instead, we (intuitively) derive the numeric level of an objects by the chain of instantiations.

For example, "BobBrown" has no instance but is instance of "Actor", "Actor" is (most-general) instance of "ActorType", which is an instance of "Node". Hence, we would associate "BobBrown" to level M0, "Actor" to M1, "ActorType" to M2, and "Node" to M3. In general such a calculation is not delivering unique level numbers in DeepTelos. But the intuition is still useful. Parallel to all these level is the omega level: objects of any level are also instances of "Proposition".

Cross-level constraints are also used in our solution, e.g. to define the "authorizedFor" relation:

```
Actor in Class with
  constraint isAuthorized :
    $ forall a/Actor t/Task T/AuthorizedTaskType
       (t in T) and (t executor a)
       ==> (a authorizedFor T) $ end
```

The variable *a* ranges over the "Actor" level (M1) and *T* over the "TaskType" level (M2). The relation "authorizedFor" is also a cross-level relationship. Since Telos regard all explicit information as objects, such constraints are technically not different from other constraints.

D. Integrity mechanism

Integrity constraints are specified as shown avove as first-order formulas, which are compiled to Datalog rules using the Lloyd-Topor algorithm. The semantics of Datalog ("perfect model semantics") computes the minimal Hernrand model of a Datalog program via a fixpoint engine. ConceptBase includes such an engine. Technically, the integrity constraints are formally negated to derive the violation of the constraint ("denial form"). If the violation can be derived, the corresponding update to the database (i.e. Telos models) is rejected. The Datalog engine uses the closed-world assumption for handling negated predicates. This yields a fast implementation but is not equivalent to classical negation. However, the advantage of the CWA is that is naturally computes the transitive closure of relations. Rules defining transitive closures play an important role in (software) modeling.

For example, dependencies between artifacts can be traced via a transitive closure. In this solution we also used queries to formulate constraints, see "CriticalBut-NotValidatedM0/M1" in section 2. Queries cannot be violiated but can be used to return violators as their answer. This is in many cases the preferred way in ConceptBase to define constraints. It allows to work with incomplete models that technically violate some constraints while being completed. A formal integrity constraint would forbid such models. A query can be called at any time and the modeler can then change the models to reduce the number of violators.

Multiplicity constraints are defined via the "necessary", "single" categories (compare figure 11). The constraints are maintained by ConceptBase. A violation leads to the rollback of the model update that first introduced the violation. Note that multiplicity constraints can be formulated at any abstraction level since ConcepBase represents all explicit model elements as objects. For example, classes and metaclasses are all objects (instances of "Proposition").

E. Abstraction

The solution is organized in modules. The upper-level modules have more abstract definitions than the lower level models. For example, the DeepTelos module defines the DeepTelos constructs. They can be used in any multi-level modeling project (e.g. the "CarExample"). This is a highly re-usable module and not domain-dependent. The "ProcessModels"

module contains the definitiond of processes (tasks, actors, artifacts, ..). This can be re-used to the extent to which this conceptualization of processes is deemed generic enough. In our solution, we integrated the BPMN meta model into this module to save some time. This is certainly not the only possible process modeling language, but it is a defacto standard. In the original BPMN meta model, a Petri net like semantics was inherited by the BPMN meta model: tasks could be triggered and that led to a token flow to the next task. We used an active rule (ECA) to implement the token flow semantics. However, this is disabled in this solution because the processes of the challenge have a more of a "colored petri net" semantics. The task instances (M1) could be produced by triggering an active rule that realizes the semantics of the corresponding task type (setting dates, input/output artifacts, executor, ...). Since this cannot be automated due to missing data, we abstained from realizing such dynamic semantics. Still, the "ProcessModels" module is abstract enough to be re-used on process modeling domains. The module "CodingProcess" defines the ACME process example. This is apparently not very re-usable in other domains.

F. Deep characterization

Consider as example the two levels (Task IN TaskType). The two levels are closely related via the DeepTelos rules. In particular each instance of "TaskType" becomes a subclass of "Task". As written earlier, DeepTelos replaces potencies by most-general instances. So, a property such as "duration" is defined at the object "Task", not "TaskType". One could see the pair "Task+TaskType" as a single abtract entity. Then, the attribute "duration" is indeed defined at an abstract level and characterizes objects at the UML-ish M0 level.

Another aspect of deep characterization are the multi-level rules used in this solution. They range over three or more instantiation levels and are partially evaluated to sets of rules ranging over just two levels. For example, the "necessary" construct defined the "1..*" multiplicity by a single formula. It characterizes all uses of the necessary construct.

Note that DeepTelos itself is defined by multi-level rules. They are also partially evaluated into many two-level rules. Still, they are only defined once and then used many times.

G. Reuse

This aspect was already discussed in the previous paragraphs. The re-use is supported by the module structure. The sources code of the modules (Telos sources) is shared at http://conceptbase.cc/multi2019challenge/SOURCES.

A module source can be directly inserted into a ConceptBase database and then be used. The sources are compiled to a set of facts/objects plus a set of (executable) rules. So, one can regard them as logical theories. Like with logical theories, one can just add them to one another. Of course, integrity constraints need to be regarded. In the case of this challenge, we also re-used existing Telos sources (BPMN) and integrated them into the DeepTelos concepts.

H. Semantics

ConceptBase uses a Datalog-neg engine to evaluate rules and constraints. The axioms for the basic constructs for specialization (c isA d), attribution/relations (x m y), and instantiation (x in c) are also expressed as rules and constraints. The semantics of a model in ConceptBase is the minimal fixpoint interpretation (=extension) for the logical predicates occuring in rules and constraints, as computed by the Datalog engine. The DeepTelos rules (R1) to (R5) are subject to the Datalog engine as well, providing in particular solutions to the DeepTelos specialization (c ISA d) and the instantiation predicate (x in c). Domain-specific rules and constraints are treated in the same way as any rule or constraint in ConceptBase. For example, there are rules for the predicate (a authorizedFor t). The extension specifies which actor is authorized for which task, which is evaluated at the M0 level (here: process traces). The extension is computed by ConceptBase.

The Telos specialization relationship (c isA d) is axiomatized in [11]. One of the axioms realizes the Nixon diamond pattern for attributes and relations: for any combination of an object x and a class attribute/relation label m, the class c that defines the most specific m is unique. This axiomatizaton supports *s*ubstitutability: wherever an instance of the superclass d is allowed, an instance of the subclass is also allowed. Note however that Telos does not support class methods. Substituability is limited to rules, constraints, and queries. The DeepTelos specialization (c ISA d) shall mostly behave like the Telos specialization, hough we did not yet transcribe all axioms for (c isA d) to (c ISA d). The only reason to introduce the DeepTelos specialization was an implementation limitation for the Telos specialization in ConceptBase: it does not support user-defined rules for the Telos specialization predicate. The Telos multiple specialization and multiple classification requires extra effort to map it to UML.

I. Incremental updates

ConceptBase supports incremental updates at any instantiation level at any time. The Telos axioms as implemented by ConceptBase [12] assign at least the builtin-class "Proposition" to any object. Hence, any object does have at least the class "Proposition" and can use the features of "Proposition" to assign attributes, relations, sub/superclasses, and classes/instances at any time. In principle, one could start to define objects at the UML M0 level first and attach its M1-level classes subsequently. Similarily, the M2-level of M1-level ovjects could be defined when the M1-level objects are already defined (as instances of Proposition).

In practice, the modeling of the different abstraction levels usually starts with the more abstract levels. But, the more abstract levels can be extended and modified as long as the builitin axioms and user-defined integrity constraints are fulfilled. Rules, constraints and queries can also be defined at any time. They can also be deleted and then be replaced by revisions at any time.

J. Lessons learned

One challenge with using ConceptBase was that recursive rules needed to be evaluated while an update to a model was processed. ConceptBase used to disable tabling (=cache of the extension of derived predicates) of the Datalog-engine during updates. That could lead to infinite loops since tabling is essential to avoid infinitite loops for recursive predicates. We addressed this loophole by temporarily re-activating tabling during updates. Another lesson learned was that the formula compiler of ConceptBase ignored the specialization facts derived by DeepTelos to check the typing of predicates. This forced us initially to some awkward definitions for some queries and redundant specialization facts. This weakness is now removed. The more most-general instances are defined, the more rules and constraints are generated by the partial evaluator. Hence, it may be more efficient not to partially evaluate the DeepTelos rules for very large models.

K. Further aspects

The partial evaluator generated about 150 two-level rules from the 6 multi-level rules. The number of generated rules depends on the number of instances and sub-classes associated to objects matching the predicate (m IN c). Still, one could use the two-level rules instread of the DeepTelos rules to carve out sub-sets of the modules, e.g. just the CodingProcess plus its sub-module.

Some additional features were added to the process model, in particular to check delayed tasks. This uses the ability of ConceptBase to evaluate arithmetic and function expressions.

ConceptBase was orginally developed as design repository for data-intensive applications, project DAIDA [13]. The metameta model of the design repository had the concepts ”DesignDecision” (=”Task”), ”DesignObject” (=”Artifact”), and ”DesignTool” (roughly ”Actor”). It did also feature all abstraction levels used in the challenge. The main drawback was the missing multi-level modeling aspect. This led to many instantiations of the produces/uses relations, while in our solution, we only need to define it for ”TaskType” and ”Task”. The DAIDA project also pioneered the fine-grain traceability of reequirements to code lines. An updated version of this feature is available at [12].

VII. Conclusions

We provided a solution for the MULTI process challenge. Except for the authorization constraint (P17), all requirements were met. We also provided an example process execution to highlight how the ACME process can be traced and monitored.

During the development of the solution, some weaknesses of the multi-level partial evaluator of ConceptBase became apparent. While ConceptBase allows to change objects at any abstraction level at any time (including deleting them), this has put the partial evaluator to its limits. It maintains a dependency graph between code that is generated from the formulas to maintain the executable rules. This graph can become cyclic and then prone to let the evaluator run into a loop. This can be avoided by defining the most-general instance relations at the beginning and then not change them afterwards. Still, it was an unpleasant experience. The partial evaluator is also a bit time consuming. It can take a few seconds to compile all modules. Once compiled, the execution is fast.

The solution does has some elegancy with it, e.g. directly using the terms actor, actor types etc. in solution. We find it still not obvious that we saved a lot of coding by the DeepTelos multi-level modeling approach. The advantage seems to be more in reuse rather than avoiding accidential complexity.

A key advantage of ConceptBase was that it naturally supports any number of instantiation levels. Another advantage is that the object ”Proposition” is fully accessible to users of ConceptBase to add new abstract constructs extending the attribution, instantiation and specialization constructs. Further, the Datalog compilation of rules is robust and delivers fast code.

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] J. J. Odell, *Advanced object-oriented analysis and design using UML*. Cambridge University Press, 1998, ch. Power types, pp. 23–32.

[4] 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

[5] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter, “A formal approach to the specification and transformation of constraints in MDE,” *J. Log. Algebr. Program.*, vol. 81, no. 4, pp. 422–457, 2012. [Online]. Available: https://doi.org/10.1016/j.jlap.2012.03.006

[6] 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

[7] J. de Lara, E. Guerra, R. Cobos, and J. Moreno-Llorena, “Extending deep meta-modelling for practical model-driven engineering,” *Comput. J.*, vol. 57, no. 1, pp. 36–58, 2014. [Online]. Available: http://dx.doi.org/10.1093/comjnl/bxs144

[8] C. Atkinson, M. Gutheil, and B. Kennel, “A flexible infrastructure for multilevel language engineering,” *IEEE Trans. Software Eng.*, vol. 35, no. 6, pp. 742–755, 2009. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TSE.2009.31

[9] U. Frank, “Multilevel modeling - toward a new paradigm of conceptual modeling and information systems design,” *Business & Information Systems Engineering*, vol. 6, no. 6, pp. 319–337, 2014. [Online]. Available: https://doi.org/10.1007/s12599-014-0350-4

[10] M. A. Jeusfeld, “A deductive view on process-data diagrams,” in *4th IFIP WG 8.1 Working Conference on Method Engineering, ME 2011, Paris, France, April 20-22, 2011. Proceedings*, 2011, pp. 123–137. [Online]. Available: https://doi.org/10.1007/978-3-642-19997-4_13

[11] ——, “Complete list of O-Telos axioms,” 2005, online: http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/d1228997/O-Telos-Axioms.pdf.

[12] ——, “Metamodeling and method engineering with ConceptBase,” in *Metamodeling for Method Engineering*, M. A. Jeusfeld, M. Jarke, and J. Mylopoulos, Eds. MIT Press, 2009, pp. 89–168.

[13] M. Jarke, Ed., *Database Application Engineering with DAIDA*, ser. Research Reports ESPRIT. Springer, 1993.

APPENDIX: SOURCE CODE

The appendix shows some excerpts of the source code. The complete source code (about 3000 lines of code) is available at http://conceptbase.cc/multi2019challenge/SOURCES.

```
{* excerpt from module oHome: *}

Proposition with
  attribute
    necessary: Proposition;
      {* multiplicity 1..*                      *}
    single: Proposition;
      {* multiplicity 0..1                      *}
    reflexive:  Proposition;
      {* any object is related to itself       *}
    transitive: Proposition;
      {* relation is closed under transitivity *}
    symmetric: Proposition;
       {* if x rel y then also y rel x         *}
    antisymmetric: Proposition;
       {* if x rel y and (y rel x) then x=y    *}
    asymmetric: Proposition
       {* if x rel y then not y rel x          *}
end


RelationSemantics in Class with
  constraint
    singleConstraint : $ forall c,d/Proposition
      p/Proposition!single x,m/VAR
      P(p,c,m,d) and (x in c) ==>
      (
       forall a1,a2/VAR
          (a1 in p) and (a2 in p) and Ai(x,m,a1)
          and Ai(x,m,a2) ==> (a1=a2)
      ) $;
  necConstraint : $ forall c,d/Proposition
      p/Proposition!necessary x,m/VAR
      P(p,c,m,d) and (x in c) ==>
      exists y/VAR (y in d) and (x m y) $;
   asym_IC: $ forall AC/Proposition!asymmetric
      C/Proposition x,y/VAR M/VAR
      P(AC,C,M,C) and (x in C) and (y in C) and
      (x M y)  ==> not (y M x) $;
   antis_IC:
      $ forall AC/Proposition!antisymmetric
       C/Proposition x,y/VAR M/VAR
       P(AC,C,M,C) and (x in C) and (y in C) and
       (x M y) and (y M x)  ==> (x = y) $
  rule
   trans_R: $ forall x,z,y,M/VAR
      AC/Proposition!transitive C/Proposition
      P(AC,C,M,C) and (x in C) and
      (y in C) and (z in C) and
      A_e(x,M,y) and (y M z) ==> (x M z) $;
   refl_R: $ forall x,M/VAR
      AC/Proposition!reflexive C/Proposition
      P(AC,C,M,C) and (x in C)
       ==> (x M x) $;
   symm_R: $ forall x,y,M/VAR
      AC/Proposition!symmetric C/Proposition
      P(AC,C,M,C) and (x in C) and (y in C) and
      A_e(x,M,y)  ==> (y M x) $
end


{* DeepTelos definition *}
Proposition with
   attribute
    ISA : Proposition;
    IN : Proposition
end
```

```
DeepTelosRules in Class with
   rule
    mrule1 : $ forall m,x,c/Proposition
               (x in c) and (m IN c) and
               not (x isA m)
               ==> (x ISA m) $;
    mrule2 : $ forall x,c,d/Proposition
               (c ISA d) and (x in c)
               ==> (x in d) $;
    mrule3 : $ forall c,d,m,n/Proposition
               (m IN c) and (n IN d) and (c ISA d)
               ==> (m ISA n) $;
    mrule4 : $ forall m,x,c/Proposition
               (m IN c) and (x isA m) and
               not (x in QueryClass)
               ==> (x in c) $;
    mrule5 : $ forall m,mx,x,c/Proposition
               (m IN c) and :(x isA mx):
               and (mx ISA m)
               and not (x in QueryClass)
               ==> (x in c) $
  constraint
    mconstr1 : $ forall x,m,c/Proposition
                 (m IN c) and (x in c)
                 ==> not (x in m) $
end


{* M3-like level *}

NodeOrLink with
   attribute
    connectedTo : NodeOrLink
end

Node isA NodeOrLink  end

NodeOrLink!connectedTo isA NodeOrLink  end

ModelType isA Node with
   attribute
    contains : NodeOrLink
end

ModelType!contains isA NodeOrLink!connectedTo  end


{* Queries, rules and constraints for
   the challenge (excerpt) *}

CriticalButNotValidatedM1 in QueryClass
 isA CriticalTaskType with
   constraint
    c3 : $ exists x/ArtifactType
           (this produces x) and
           not (exists vt/ValidationTaskType
                 (vt uses x)) $
end

CriticalButNotValidatedM0 in QueryClass
 isA CriticalTask with
   constraint
    c3 : $ exists x/Artifact
            (this produces x) and
           not (exists vt/ValidationTask
                (vt uses x)) $
end

CriticalTaskType in Node,Class isA TaskType with
   constraint
    c1 : $ forall t/CriticalTaskType at/ActorType
 (t executortype at) ==> (at in SeniorActorType) $;
    c2 : $ forall t/CriticalTaskType a/Actor
```

```
 (t executorset a) ==> (a in SeniorActor) $
end

UnmatchedTask in QueryClass isA ProperTask with
   computed_attribute
    T : ProperTaskType;
    PT : ProperProcessType;
    P : ProperProcess
  constraint
    cm : $
      (P contains this) and
      :(this in T): and (P in PT) and
      (PT <> Process) and (T <> Task) and
      not (PT contains T) $
end

taskDuration in Function isA Integer with
   parameter
    t : Task
  constraint
    cdur : $ exists d1,d2/Integer
       (t begindate d1) and (t enddate d2) and
         (this = d2-d1) $
end

Task in Class with
   IN
    class : TaskType
  attribute
    uses : Artifact;
    produces : Artifact;
    begindate : Integer;
    enddate : Integer;
    duration : Integer;
    executor : Actor
   rule
    durrule : $ forall t/Task d/Integer
     (d = taskDuration(t)) ==> (t duration d) $;
end

DelayedTask in QueryClass isA Task with
   constraint
    isDelayed : $ exists T/TaskType pd,d/Integer
                  (this in T) and
                  (T plannedduration pd) and
                  (this duration d) and (d > pd) $
end

{* P17 *}
AuthorizedTaskType in QueryClass
  isA TaskType with
   constraint
   catt : $ exists a/Actor (a authorizedFor this) $
end

Actor in Class with
  constraint
    isAuthorized :
       $ forall a/Actor t/Task T/AuthorizedTaskType
          (t in T) and (t executor a)
          ==> (a authorizedFor T) $
end


{* S13 *}
TestCaseDesign in CriticalTaskType with
   altname
    de : "Testszenarioentwurf"
  next
    n1 : TestDesignReview
  executortype
    testcasedesigner : DeveloperOrTestDesigner
  produces
    artifact1 : TestCaseDocument
  creator
    createdBy : BobBrown
  constraint
    s13a : $ forall tcd/Task a/Actor
       (tcd in TestCaseDesign) and
       (tcd executor a) ==> (a in SeniorAnalyst) $
end
```