Integrating Language and Ontology Engineering

Bruno Barroca[‡], Thomas Kühne[*], Hans Vangheluwe[†‡]

[†] University of Antwerp, Belgium
[‡] McGill University, Montréal, Canada
[*] Victoria University of Wellington, New Zealand

Keywords: Ontologies, FTG+PM, Properties, Meta-modeling Environments, Description Logics, Semantic Mapping, Verification spaces

Abstract. Creating new modeling environments has become a relatively low-cost investment thanks to meta modeling environments and language workbenches that can automatically synthesize environments from language specifications. However, the currently existing tools are focused on language syntax and execution/simulation rather than providing means to reason with semantic properties from the real world. It appears that reasoning opportunities as they arise in ontology research (e.g., based on description logics) are currently not exploited in the field of language engineering. In this vision paper, we explore the integration of the hitherto rather isolated areas of language engineering and ontology engineering in order to exploit the potential of using reasoning for models expressed in user-defined languages.

1 Introduction

The automated generation of modeling and programming environments can look back at a long history – starting in the 1980's [1] – of providing means to define the syntax and semantics of languages. Recently, influences from model-driven development made it easier to define semantics more flexibly by means of model-to-model transformations. However, most semantics definitions are still concerned with execution or simulation.

Despite the common trend of increasing the return on investment by improving languages and tools, it appears that the basic 'grammar-ware' principles of meta modeling environments has not changed in over 30 years. However, we can observe that the ability to chain transformations has enabled modelers to learn more about their models – and thus the systems under study – than would have been economically feasible in former days by defining a single transformation towards a particular platform: i.e., generating a compiler. The Formalism Transformation Graph + Process Modeling (FTG+PM) [2] approach from the area of multi-paradigm modeling is the most explicit example for this style of modeling in which the output of one transformation is used as the input for another one.

The crucial aspect of this approach is that from an initial set of models, we can have orthogonal transformations with totally different intents. In particular, each path in the transformation graph is designed to, from the initial set of models, illuminate a particular set of properties of the system under study/development, e.g., regarding termination, liveliness or even safety [3].

In this context, the main trend from the language engineering area, is to establish a trace between the property checking results (in the leaves of a given transformation chain) and the original model so that, for instance, the results can be suitably interpreted by the modeler in a particular domain. In this paper, we test yet another approach on this problem: we argue that one should explicitly associate the properties of interest as an ontological type (i.e., introduce ontological types explicitly as first-class citizens), and then test/check its conformance by means of transformation(s) to the respective semantic domains w.r.t. those properties.

To this end, we explore at the conceptual level, the integration of two hitherto unrelated areas: First, language engineering, with its support for model representation and transformation, and second, ontologies with their rich tool set for describing and inferring properties. In Section 2 we introduce a railway system, that we will use as a running example to illustrate both challenges and our proposed solution. In section 3, we present the context of the problem and the ingredients that contribute to our proposed solution which is presented in Section 4. Finally, we conclude in Section 5.

2 Case Study: The Railway System

Railway transportation is only one of the many sectors that have benefited from the application of computer automation techniques ranging from early automated railway control to train scheduling [4].

2.1 General Requirements

First and foremost a railway system needs to support transport of people or goods between endpoints. This basic function, however, is accompanied by additional constraints and boundary conditions. For example, a railway system must be affordable by end users and profitable for operators, placing certain constraints on energy consumption, personnel cost, etc. Moreover, as with any system where lives and or loss of large monetary investments are at stake, safety is a critical concern. Preventing trains from derailing or colliding is crucial for a railway system.

We observe that it is possible to separate the requirements into what happens, i.e., trains moving between endpoints, and how it happens, e.g., without collisions. Generally speaking, we can distinguish between the mechanics of a system (i.e., its execution semantics), and properties that can be associated with the mechanics (with verification semantics).

2.2 Execution Semantics

In the following, we will use a language to define a railway network (c.f. [5]) to describe the system mechanics. A railway network comprises a collection of railway tracks organized into sections and connecting different train stations. In order to avoid collisions, only one train is allowed in one section at a time. Signals control when trains may enter and leave sections. The combination of signals, their control logic, and track switches is called the interlocking system and its purpose is to prevent conflicting train movements.

The behavior of such an interlocking system is specified in a control table which embodies information about conflicting train routes. A formal description of the control table in combination with information about the physical railway layout could be regarded as a prescriptive model for the execution behavior of the interlocking system.

2.3 Verification Semantics

While one could use the execution behavior of an interlocking system to simulate the latter, this would only enable the detection of errors in the interlocking system through testing. In order to gain certainty regarding the absence of any potential conflicting train movements, it is necessary to formalize the respective safety requirement and ensure that it is satisfied by the interlocking system under study.

Somsak achieves the automated verification of an interlocking system by translating it into a colored petri net (CPN) which can then be automatically verified using CPN tools, a Petri Net model checker [5]. Each railway track represents a resource that should only be used by one train in order to avoid collisions, and can therefore be directly mapped to a place in the CPN formalism. Despite the fact that Somsak only verified three scenarios, this work is a good example of how domain models (such as interlocking systems) can be automatically and systematically verified by reusing effective algorithms developed in other domain such as CPNs.

Note that safety concerns are just one example for the utility of verification semantics. The railway operators may also be interested in the economics of running the railway system and thus may want to convince themselves that certain track layouts will achieve a desired throughput. In general, many such properties will have to hold for a given system and will be required to occur in conjunction, i.e., the system should be both economical to run and be safe to use.

3 Linguistic Models versus Ontologies

It is worthwhile pointing out that both execution and verification represent two different interpretations of one and the same model. Two different semantic domains are used in order to answer different sets of questions regarding the same model. The execution semantics tells us what may happen for a given interlocking

system, while the verification semantics tells us whether the entirety of all that may happen for a given interlocking system satisfies a particular property.

3.1 Natural vs. Verification Semantics

Intuitively, it appears that the execution semantics could be regarded as the natural "*meaning*" of an interlocking system, while the verification semantics appears to subject the interlocking system to a test. However, intuition is not reliable and in the following we thus strive to identify intrinsic differences between these two kinds of semantics.

Both kinds of semantics obviously fulfill the minimal criterion of being mappings that are functional and total [6]. At first sight, it may furthermore appear as if the two different kinds of semantics were on the same footing and thus interchangeable. In our example, there appears to be a symmetric relationship between

- the subset of safe interlocking systems,
- the subset of executions of safe interlocking systems, and
- the subset of elements in the verification space that satisfy the property "safe".

On closer examination, however, it becomes apparent that the roles of the two different semantics cannot be swapped as there is an asymmetric dependency. On the one hand, the space of all possible execution behaviors, as defined by the execution semantics, can be *partitioned* by the *properties* it may satisfy. In our example, the set of safe interlocking systems can be derived from the subset of "safe" elements in the verification space and only then find a subset of safe interlocking system executions in the overall set of all possible executions of interlocking systems. On the other hand, it is not true that the verification space can be usefully partitioned from the execution semantics. In our example, this is trivial, since following our assumptions, we have no other way to decide about the "safety" of a given interlocking system execution.

This asymmetric dependency may constitute an obstacle for purposes of our conceptual integration. Fortunately, there exist technologies are a good fit in order to

(a) enable the anchoring of natural semantics to models, and
(b) support the definition and organization of semantic properties

3.2 Language Engineering

A linguistic type model – often referred to as "metamodel" – is ideal for enabling the attachment of natural semantics to models. Following our example, if we define a "Railway DSL" grammar and well-formedness constraints combined in the form of a linguistic type model, then we can determine the structure of all elements that may occur in an interlocking system model. The linguistic type

model is thus ideally suited to be used as the basis for transformation definitions such as semantic mappings.

Existing metamodeling environments can synthesize complete environments for configuring scenarios – such as the topology of the railway network, positioning of signaling devices, the control table logic, etc. – and even use the linguistic type model as the basis for subsequent transformation (language) definitions [7].

Conformance of a model to a linguistic type model is typically granted by construction. In special cases of "freehand" (as opposed to syntax directed) editing and language evolution it may be necessary to check whether a model conforms to a linguistic type model, but in most cases the model is the result of using the linguistic type model as a generator, e.g., by structured editing or model generation.

3.3 Ontology Engineering

An ontology comprises and organizes a number of concepts and may use description logic to express concept properties and relationships. While ontologies in general contain very different kinds of concepts – of which so-called moment types (properties) are only a particular subset – the technology associated with them appears to be eminently suited to accommodate a taxonomy of properties derived from the various verification semantics.

The most popular ontology language is the Web Ontology Language (OWL), which along Common Logic (CL), and the Resource Description Framework (RDF) is included in the Ontology Definition MetaModel (ODM) proposed by the Object Management Group (OMG). The most advanced kind of reasoning in ontology engineering is achieved by means of description logics (DLs). DLs is a family of knowledge representation languages, which are used as logical formalisms to support reasoning, e.g., in the context of the Semantic Web. DLs are less expressive than first order logics, hence they are amenable to decidable and efficient reasoning mechanisms.

The most interesting feature of these languages is their ability to infer new knowledge, i.e., make implicit knowledge explicit. DLs language extensions span from introducing the notion of time as partial-order relations such as in temporal extensions [8]; introducing vagueness or incomplete concepts, such as in fuzzy logics extensions [9]; introducing the notion of concepts with probability values [10]; to introducing the ability to express possibility of event occurrence, such as on possibilistic extension [11].

Conformance of a model to an ontological type, in contrast to linguistic type model conformance, is never granted by construction. A conformance check always requires the application of a certain interpretation – i.e., a semantic mapping whose choice depends on the specific property that is to be validated against – and then the subsequent ascertainment of whether or not the property (or properties) associated with an ontological type hold(s) for the element that is the result of the semantic mapping.

In general, ontological types are agnostic to the particular domain they are applied to. For instance, in our example the safety property essentially embod-

ies "*absence of collisions*" and could also be applied to a assembly line scenario in which workpieces are transported and merged by conveyor belts. A different semantic mapping would be required from an assembly line model into the verification space, but ascertaining whether the collision-free property holds based on the respective element in the verification space by using a Petri Net model checker would be identical to the railway example. For that, just consider that instead of trains to Petri Net tokens, we would now have products to Petri Net tokens; and instead of railway tracks to Petri Net Places, we would now have conveyor belts to Petri Net Places; and the property to check is whether there exist some situation in the entire set of possible situations where there are two Tokens in the same Place.

While the above described language agnosticism allows ontological types and their definition to be reused, it also means that it is not straightforward to associated further ontological types to a model: for instance, consider the situation where the only classification information known for a given model is a single ontological type. The latter can classify models from a whole range of languages so it is not clear which semantic mapping (of many potentially applicable) would have to be applied in order to validate it (the model) against another ontological type. In contrast, if the linguistic type of a model is known (typically by-construction in most modeling environments) then it is trivial to determine which semantic mappings are available for it.

4 Integrating Languages with Ontologies

So far we have identified a particular subkind of semantics with associated properties that gives rise to ontological types (i.e., verification semantics), and furthermore observed that linguistic types complement the latter. In the following, we describe what a complete framework that encompasses language engineering and ontology engineering could look like. Such an integrated approach is clearly desirable since it allows us to not execute, simulate, and test models w.r.t. the respective referents in the real world, but also to go beyond and reason about their properties.

For instance, in our example, we could specify a railway ontology based on notions concerning the economic efficiency and/or safety of railway topologies. We could then infer the most economical, but also safe, train schedules using for instance design space exploration techniques.

A complete framework for modeling with both linguistic and ontological types, would typically use multiple formalism integration – in the style of FTG+PM – and would have a number of desirable features:

(a) While transforming models across different formalisms, we should be able to trace the properties that are being lost, preserved or created during such transformations;
(b) It is rather likely that ontological types will be reusable across domains (e.g., safety in the context of rail transportation and collision-free schedules

in the context of assembly lines). This will greatly increase the return on investment for developing the respective analysis approaches.

4.1 Conceptual Framework

We start our conceptualization from the well known model developed in [12] that explores how linguistic symbols are related to the objects they represent.

Based on this conceptualization, our first attempt is presented in Figure 1 on the left, where 'System' replaces the real world object, the 'Ontological Type' replaces thoughts (also called reference or Concept), and finally the 'Linguistic Type' replaces symbol (also called 'Sign'). Moreover, we present in the center of the triangle, the model element, which in the modeling world is the first-class entity that relates the System under study/development with both Ontological and Linguistic types.

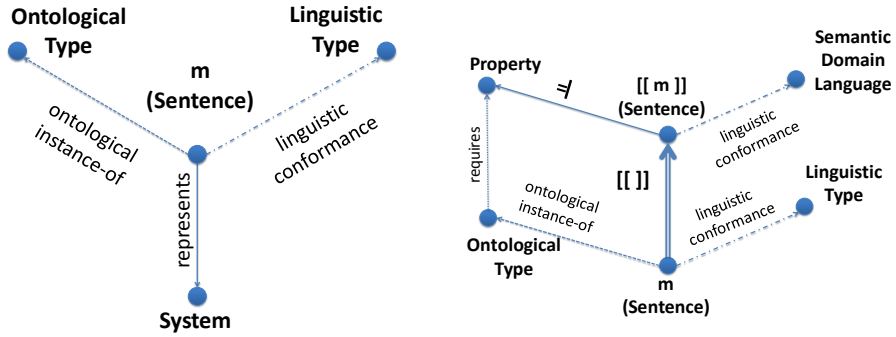

Fig. 1. A base conceptual framework (on the left), and a refined version (on the right).

The Figure 1 on the right, extends our first attempt, where we unfold the 'ontological instance-of' relation between a given model m and its ontological type (denoted in the Figure as 'Ontological Type'). Notice that we start from a situation where although linguistic types are taken for granted (i.e., by-construction) in modeling environments, the same is not true for ontological types. The unfolding of this relation is therefore done by means of a semantic mapping $[\![]\!]$ to a verification platform where a set of properties (denoted in the Figure as 'Property') relevant w.r.t. that given 'Ontological Type' can be verified. This relevance is stressed in the Figure with the relation 'requires', which means that a given 'Ontological Type' depends or includes as part of its intension, a given set of properties.

One way of transversing the left side of this diagram is therefore, by assuming that we know which Properties are required by a given Ontological Type, so that we can choose the most appropriate verification platform and devise a semantic mapping $[\![]\!]$ which can be realized by means of a model transformation. This model transformation is then able to automatically transform arbitrary

models m given that they do conform (in the linguistic sense) to a given meta-model (or grammar) depicted in the Figure as 'Linguistic Type'. The resulting transformed model denoted as $[\![m]\!]$ by construction of the transformation itself should also conform linguistically to the language from the verification platform (denoted as 'Semantic Domain Language'). The circle in this diagram is closed by the satisfaction relationship (denoted as $\models$) between $[\![m]\!]$ and the Property is established, which means that we can now conclude that the given model m is indeed an ontological instance of the given Ontological Type.

It is important to mention here that in practice, there might be several different properties that a given Ontological Type may require. Therefore, we can expect to have several different orthogonal semantic mappings $[\![]\!]$ using possibly different kinds of verification mechanisms (and their associated semantics) in order to be able to finally establish that ontological instance relationship between a model and an ontological type.

4.2 Framework Instantiation

After describing the general terms of our conceptual framework, we will now look at its instantiation in the particular case of our railway transportation example.

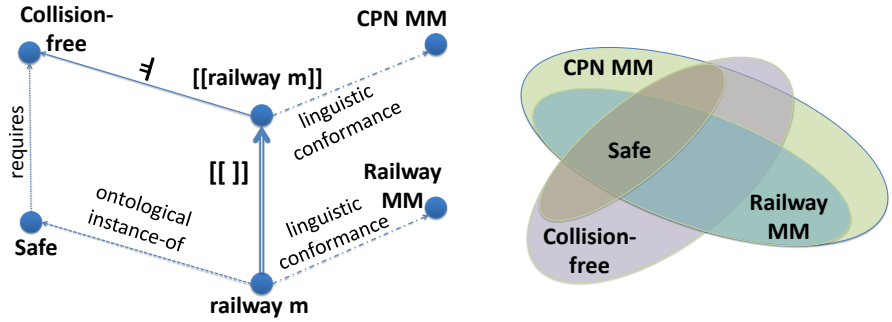

Fig. 2. Instantiation of the conceptual framework on the railway transportation example (on the left), and the respective solution-partition space (on the right).

At the bottom left of Figure 2 (on the left), we see that the system under study is an interlocking system in conjunction with a train scheduling system. The model *railway m* at the bottom right, represents this system. The linguistic type of *railway m* is the meta-model "Railway MM" to which it conforms syntactically. At the top right is an interpretation of *railway m* that was chosen in order to enable a reachability analysis. Such an interpretation can be achieved by means of a model transformations of model *railway m* into a corresponding Colored Petri Net $[\![railway\ m]\!]$. The latter also has its own linguistic type (the meta-model "CPN MM") to which it conforms syntactically.

The analysis of $[\![railway\ m]\!]$ is performed by unfolding the complete state-space (e.g., by using the model-checking engine of a CPN tool). This state-space is then queried for a property "$Collision - free$", e.g., checking whether any collision scenarios exist. This property "$Collision - free$" is depicted at the top-left. Model $railway\ m$ can be said to be an ontological instance of the ontological type "Safe" (i.e., be called "safe") if and only if its interpretation $[\![railway\ m]\!]$ satisfies property $Collision - free$.

Finally, we show in Figure 2 (on the right) the solution space partitioning complements the commuting diagram on the left. Notice that solutions are systems in the real world. On the one hand, given the 'requires' relation, the set of safe solutions is a subset of collision-free solutions: in other words, 'safe' is a stronger concept which may depend not only on being collision-free but also from other concepts. On the other hand, given the semantic mapping $[\![]\!]$, the set of solutions represented by models conforming to the Railway MM will always be a subset of the total set of the solutions represented by models conforming to the Colored Petri Nets CPN MM. Finally, given the above sets, we conclude that the set defined by the commuting diagram on the left is defined by the intersection of all of the sets defined on the right: i.e., a solution which has both a Railway model representation and its respective CPN representation, which is proven to be collision-free, and therefore an ontological instance of type Safe.

5 Conclusions

In this vision paper, we have proposed to integrate the technological spaces of language engineering and ontology engineering in a manner that strengthens language engineering to include concepts and techniques from ontology engineering. While modelers have already been successfully checking models for semantic properties with various approaches in the past, to the best of our knowledge our approach is the first to introduce ontological types with semantic properties as first-class citizens and proposes to arrange them in taxonomies, thus exploiting semantic relationships.

We have identified differences between ordinary, "natural" semantics and verification semantics, observing that semantics of the latter kind are agnostic to languages and partition sentences in a language. We have chosen to refer to types that achieve such partitions as "ontological types", referencing the fact that they are based on semantic properties, rather than on syntactic conformance.

We could only touch upon the potential of using model exploration and inference engines to leverage a taxonomy of ontological types to a tool that supports the identification of models that satisfy properties in multiple dimensions.

Nevertheless, in this paper we have contributed towards finding optimal ways of combining linguistic type models with ontologies w.r.t. previous attempts [13, 14], which are rather more focused (with again) more syntactic issues than conceptual ones. We also believe that our proposal is suitable to shed further light on the most precise characterization of ontological vs linguistic classification as presented in [15]. However, our use of the prefix "ontological" should not be

construed as meaning that our notion of "ontological types" is exactly the same as the "ontological types" discussed in [15]. While there is certainly large overlap, it remains to be seen whether our notion full subsumes the other, or rather represent as subset of that may be best characterized as "moment ontological types".

References

1. T. Teitelbaum and T. Reps, "The Cornell Program Synthesizer: A syntax-directed programming environment," *Commun. ACM*, vol. 24, pp. 563–573, Sept. 1981.
2. S. Mustafiz, J. Denil, L. Lúcio, and H. Vangheluwe, "The FTG+PM framework for multi-paradigm modelling: An automotive case study," in *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, MPM '12, (New York, NY, USA), pp. 13–18, ACM, 2012.
3. L. Lucio, S. Mustafiz, J. Denil, H. Vangheluwe, and M. Jukšs, "FTG+PM: An integrated framework for investigating model transformation chains," in *SDL Forum*, pp. 182–202, 2013.
4. J. Pachl, *Railway Operation and Control*. VTD Rail Publishing, 2nd ed., 2009.
5. S. Vanit-Anunchai, "Modelling railway interlocking tables using coloured petri nets," in *Coordination Models and Languages* (D. Clarke and G. Agha, eds.), vol. 6116 of *Lecture Notes in Computer Science*, pp. 137–151, Springer Berlin Heidelberg, 2010.
6. D. Harel and B. Rumpe, "Meaningful modeling: What's the semantics of "semantics"?," *IEEE Computer*, vol. 37, no. 10, pp. 64–72, 2004.
7. T. Kühne, G. Mezei, E. Syriani, H. Vangheluwe, and M. Wimmer, "Systematic transformation development," *Electronic Communications of the EASST*, vol. 21, 2009.
8. A. Artale, E. Franconi, M. Mosurovic, F. Wolter, and M. Zakharyaschev, "Temporal description logic," in *Handbook of Time and Temporal Reasoning in Artificial Intelligence*, pp. 96–105, MIT Press, 2001.
9. U. Straccia, "A fuzzy description logic for the semantic web," in *Fuzzy Logic and the Semantic Web. Capturing Intelligence, Elsevier (2006) 73–90.*
10. Z. Ding and Y. Peng, "A probabilistic extension to ontology language OWL," in *In Proceedings of the 37th Hawaii International Conference On System Sciences (HICSS-37), Big Island*, 2004.
11. G. Qi, J. Z. Pan, and Q. Ji, "A possibilistic extension of description logics," in *In Proc. of DL'07, 2007.*
12. C. Ogden and I. A. Richards, "The meaning of meaning: A study of the influence of language upon thought and of the science of symbolism.," *8th ed. 1923. Reprint New York: Harcourt Brace Jovanovich*, 1923.
13. B. Henderson-Sellers, "Bridging metamodels and ontologies in software engineering," *J. Syst. Softw.*, vol. 84, pp. 301–313, Feb. 2011.
14. E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, and M. Wimmer, "Lifting metamodels to ontologies - a step to the semantic integration of modeling languages," in *In Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2006*, pp. 528–542, Springer, 2006.
15. T. Kühne, "Matters of (meta-) modeling," *Software and Systems Modeling*, vol. 5, no. 4, pp. 369–385, 2006.